



Guida per gli sviluppatori

# AWS Encryption SDK



# AWS Encryption SDK: Guida per gli sviluppatori

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

---

# Table of Contents

Che cosa è la AWS Encryption SDK? .....	1
Sviluppato in repository open source .....	2
Compatibilità con librerie e servizi di crittografia .....	3
Support e manutenzione .....	4
Ulteriori informazioni .....	4
Invio di feedback .....	5
Concetti .....	6
Crittografia envelope .....	7
Chiave di dati .....	9
Chiave di avvolgimento .....	9
Portachiavi e fornitori di chiavi principali .....	10
Contesto di crittografia .....	11
Messaggio crittografato .....	13
Suite di algoritmi .....	13
Responsabile di materiali crittografici .....	14
Crittografia simmetrica e asimmetrica .....	14
Impegno chiave .....	15
Politica di impegno .....	16
Firme digitali .....	18
Come funziona l'SDK .....	18
In che modo AWS Encryption SDK crittografa i dati .....	19
Come AWS Encryption SDK decripta un messaggio crittografato .....	19
Suite di algoritmi supportate .....	20
Consigliato: AES-GCM con derivazione della chiave, firma e impegno .....	21
Altre suite di algoritmi supportate .....	22
Interagire con AWS KMS .....	24
Le best practice .....	26
Configurazione dell'SDK .....	30
Selezione di un linguaggio di programmazione .....	30
Selezione dei tasti di avvolgimento .....	30
Utilizzo di più regioni AWS KMS keys .....	32
Scelta di una suite di algoritmi .....	53
Limitazione delle chiavi di dati crittografate .....	62
Creazione di un filtro di scoperta .....	66

Impostazione di una politica di impegno .....	69
Lavorare con dati in streaming .....	69
Memorizzazione nella cache delle chiavi dati .....	70
Utilizzo dei keyring .....	71
Come funzionano i keyring .....	72
Compatibilità dei keyring .....	73
Requisiti diversi per i portachiavi di crittografia .....	74
Keyring e fornitori di chiavi master compatibili .....	74
Scegliere un portachiavi .....	75
Portachiavi AWS KMS .....	76
AWS KMS Portachiavi gerarchici .....	95
Keyring non elaborati AES .....	120
Keyring non elaborato RSA .....	124
Keyring multipli .....	130
Linguaggi di programmazione .....	135
C .....	135
Installazione .....	136
Utilizzo dell'SDK per C .....	137
Esempi .....	142
.NET .....	149
Installazione e creazione .....	151
Debug .....	152
Portachiavi AWS KMS .....	152
Contesto di crittografia richiesto (CMM) .....	155
Esempi .....	158
Java .....	166
Prerequisiti .....	166
Installazione .....	168
Portachiavi AWS KMS .....	169
Contesto di crittografia richiesto (CMM) .....	172
Esempi .....	174
JavaScript .....	187
Compatibilità .....	188
Installazione .....	190
Modules .....	191
Esempi .....	194

Python .....	200
Prerequisiti .....	201
Installazione .....	201
Esempi .....	202
Interfaccia a riga di comando .....	213
Installazione dell'interfaccia a riga di comando .....	215
Come utilizzare l'interfaccia a riga di comando .....	218
Esempi .....	233
Sintassi e riferimento parametri .....	257
Versioni .....	271
Caching della chiave dei dati .....	275
Come utilizzare il caching della chiave di dati .....	276
Utilizzo della memorizzazione nella cache delle chiavi dati: S tep-by-step .....	277
Esempio di caching della chiave di dati: crittografare una stringa .....	285
Impostazione delle soglie di sicurezza della cache .....	301
Dettagli di caching della chiave dei dati .....	303
In che modo funziona il caching della chiave dei dati .....	303
Creazione di una cache di materiali crittografici .....	306
Creazione di un responsabile della cache di materiali crittografici .....	307
Cosa c'è in una voce della cache della chiave di dati? .....	308
Contesto di crittografia: Come selezionare le voci della cache .....	309
La mia applicazione utilizza chiavi dati memorizzate nella cache? .....	309
Esempio di caching della chiave dei dati .....	310
Risultati della cache locale .....	311
Esempio di codice .....	312
Modello AWS CloudFormation .....	324
Versioni del AWS Encryption SDK .....	339
C .....	340
C# /.NET .....	340
Interfaccia a riga di comando (CLI) .....	341
Java .....	343
JavaScript .....	345
Python .....	347
Dettagli della versione .....	348
Versioni precedenti alla 1.7. x .....	348
Versione 1.7. x .....	349

Versione 2.0. x .....	352
Versione 2.2. x .....	353
Versione 2.3. x .....	354
Migrazione della tuaAWS Encryption SDK .....	355
Come migrare e implementare .....	357
Fase Aggiornare l'applicazione alla versione più recente 1.xversione .....	357
Fase Aggiornare l'applicazione alla versione più recente .....	358
Aggiornamento dei provider di chiaviAWS KMS principali .....	360
Migrazione alla modalità .....	361
Migrazione alla modalità di rilevamento .....	364
Aggiornamento in corsoAWS KMSportachiavi .....	368
Impostazione della politica di impegno .....	370
Come impostare la tua politica di impegno .....	372
Risoluzione dei problemi relativi alla migrazione alle versioni più recenti .....	379
Oggetti obsoleti o rimossi .....	380
Conflitto di configurazione Politica di impegno e suite di algoritmi .....	380
Conflitto di configurazione Politica di impegno e testo cifrato .....	381
Convalida dell'impegno chiave non riuscita .....	382
Altri errori di crittografia .....	382
Altri errori di decrittografia .....	382
Considerazioni sul rollback .....	383
Domande frequenti .....	384
Riferimento .....	389
Riferimenti a formati di messaggi .....	389
Struttura dell'intestazione .....	390
Struttura corpo .....	398
Struttura piè di pagina .....	404
Esempi di formati di messaggi .....	404
Dati incorniciati (formato messaggio versione 1) .....	405
Dati incorniciati (formato messaggio versione 2) .....	409
Dati non inclusi in frame (formato messaggio versione 1) .....	411
Riferimento AAD del corpo .....	415
Riferimenti agli algoritmi .....	416
Riferimento al vettore di inizializzazione .....	421
AWS KMSDettagli tecnici del portachiavi gerarchico .....	422
Cronologia dei documenti .....	424

---

Aggiornamenti recenti .....	424
Aggiornamenti precedenti .....	427
.....	cdxxix

# Che cosa è la AWS Encryption SDK?

AWS Encryption SDK è una libreria di crittografia lato client progettata per rendere ancora più semplice la crittografia e la decrittazione dei dati utilizzando best practice e standard di settore. Questo servizio consente di concentrarsi sulle funzionalità principali dell'applicazione, piuttosto che su come crittografare e decrittografare i dati nel migliore dei modi. AWS Encryption SDK viene fornito gratuitamente con la licenza Apache.

AWS Encryption SDK risponde a domande come le seguenti:

- Quale algoritmo di crittografia devo usare?
- Come o in che modo è consigliabile utilizzare tale algoritmo?
- Come posso generare la chiave di crittografia?
- Come posso proteggere la chiave di crittografia e dove è possibile archivarla?
- Come posso rendere i miei dati crittografati portatili?
- Come posso assicurarmi che il destinatario previsto possa leggere i miei dati crittografati?
- Come posso garantire che i miei dati crittografati non vengano modificati nel periodo che intercorre tra la scrittura e la lettura?
- Come posso usare le chiavi dati che AWS KMS restituiscono?

Con AWS Encryption SDK, definisci un [provider di chiavi master](#) (Java e Python) o un [portachiavi](#) (C, C#.NET e JavaScript) che determina quali chiavi di wrapping utilizzare per proteggere i tuoi dati. Quindi è possibile crittografare e decrittare i dati tramite semplici procedure fornite da AWS Encryption SDK. AWS Encryption SDK si occuperà di tutte le altre operazioni.

Eliminando AWS Encryption SDK, puoi dedicare più risorse alla creazione di una soluzione di crittografia rispetto alla funzionalità di base della tua applicazione. AWS Encryption SDK risponde a queste domande fornendo i seguenti oggetti.

Un'implementazione predefinita conforme alle best practice di crittografia

Per impostazione predefinita, AWS Encryption SDK genera una chiave di dati univoca per ogni oggetto dati che crittografa. Questo segue le best practice di crittografia sull'uso di chiavi di dati univoche per ciascuna operazione di crittografia.

AWS Encryption SDK consente di crittografare i dati utilizzando un algoritmo della chiave simmetrico, sicuro e autenticato. Per ulteriori informazioni, consulta [the section called “Suite di algoritmi supportate”](#).

Un framework per proteggere le chiavi di dati con chiavi di avvolgimento

AWS Encryption SDK protegge le chiavi dati che crittografano i dati crittografandole con una o più chiavi di avvolgimento. Fornendo un framework per crittografare le chiavi di dati con più di una chiave di wrapping, AWS Encryption SDK contribuisce a rendere portatili i dati crittografati.

Ad esempio, crittografa i dati con un input AWS KMS e una AWS KMS key chiave dal tuo HSM locale. Puoi utilizzare una delle chiavi di wrapping per decrittografare i dati, nel caso in cui una non sia disponibile o il chiamante non sia autorizzato a utilizzare entrambe le chiavi.

Un messaggio formattato che memorizza le chiavi di dati crittografati con i dati crittografati

AWS Encryption SDK memorizza i dati crittografati e la chiave di dati crittografati insieme in un [messaggio crittografato](#) che utilizza un formato di dati definito. Questo significa che non è necessario tenere traccia o proteggere le chiavi di dati che crittografano i dati poiché AWS Encryption SDK lo fa per te.

Alcune implementazioni linguistiche AWS Encryption SDK richiedono un Account AWS, ma AWS Encryption SDK non lo richiedono Account AWS e non dipendono da alcun servizio. AWS Ne hai bisogno Account AWS solo se scegli di utilizzarlo per [AWS KMS keys](#) proteggere i tuoi dati.

## Sviluppato in repository open source

AWS Encryption SDK È sviluppato in repository open source su GitHub. È possibile utilizzare questi repository per visualizzare il codice, leggere e segnalare problemi e trovare informazioni specifiche sull'implementazione del linguaggio.

- SDK di crittografia AWS per C — [aws-encryption-sdk-c](#)
- AWS Encryption SDK per .NET: [aws-encryption-sdk-net](#) directory del aws-encryption-sdk-da fny repository.
- AWS CLI di crittografia — [aws-encryption-sdk-cli](#)
- SDK di crittografia AWS per Java — [aws-encryption-sdk-java](#)
- SDK di crittografia AWS per JavaScript — [aws-encryption-sdk-javascript](#)
- SDK di crittografia AWS per Python — [aws-encryption-sdk-python](#)

## Compatibilità con librerie e servizi di crittografia

AWS Encryption SDK è supportato in diversi [linguaggi di programmazione](#). Tutte le implementazioni linguistiche sono interoperabili. È possibile crittografare con un'implementazione di una lingua e decrittografare con un'altra. L'interoperabilità potrebbe essere soggetta a vincoli linguistici. In tal caso, questi vincoli sono descritti nell'argomento relativo all'implementazione della lingua. Inoltre, durante la crittografia e la decrittografia, è necessario utilizzare keyring compatibili o chiavi master e provider di chiavi master. Per informazioni dettagliate, consultare [the section called “Compatibilità dei keyring”](#).

Tuttavia, AWS Encryption SDK non può interagire con altre librerie. Poiché ogni libreria restituisce dati crittografati in un formato diverso, non è possibile crittografare con una libreria e decrittare con un'altra.

### Client di crittografia DynamoDB e crittografia lato client Amazon S3

[AWS Encryption SDK Non possono decrittografare i dati crittografati dal DynamoDB Encryption Client o dalla crittografia lato client Amazon S3. Queste librerie non possono decrittografare il messaggio crittografato restituito.](#) AWS Encryption SDK

### AWS Key Management Service (AWS KMS)

AWS Encryption SDK Possono utilizzare [chiavi dati per proteggere AWS KMS keysi dati](#), incluse le chiavi KMS multiregionali. Ad esempio, puoi configurare per AWS Encryption SDK crittografare i tuoi dati in uno o più AWS KMS keys sistemi. Account AWS Tuttavia, è necessario utilizzare AWS Encryption SDK per decrittare tali dati.

[AWS Encryption SDK Non possono decrittografare il testo cifrato restituito da Encrypt o dalle operazioni. AWS KMS ReEncrypt](#) Analogamente, l'operazione AWS KMS [Decritta](#) non può decrittare il [messaggio crittografato](#) AWS Encryption SDK restituito.

AWS Encryption SDK [Supporta solo chiavi KMS con crittografia simmetrica](#). Non è possibile utilizzare una [chiave KMS asimmetrica](#) per la crittografia o l'accesso a. AWS Encryption SDK Genera le proprie chiavi di firma ECDSA per le [suite di algoritmi](#) che firmano i messaggi.

Per informazioni su come decidere quale libreria o servizio utilizzare, vedi [Come scegliere uno strumento o un servizio di crittografia in AWS Servizi](#) e strumenti di crittografia.

## Support e manutenzione

AWS Encryption SDK utilizza la stessa [politica di manutenzione](#) utilizzata dall'AWSSDK e dagli strumenti, comprese le fasi di controllo delle versioni e del ciclo di vita. Come [procedura ottimale](#), si consiglia di utilizzare l'ultima versione disponibile di AWS Encryption SDK per il linguaggio di programmazione in uso e di eseguire l'aggiornamento non appena vengono rilasciate nuove versioni. Quando una versione richiede modifiche significative, come l'aggiornamento da AWS Encryption SDK versioni precedenti alla 1.7. x alle versioni 2.0. x e versioni successive, forniamo [istruzioni dettagliate](#) per aiutarti.

Ogni implementazione del linguaggio di programmazione AWS Encryption SDK è sviluppata in un GitHub repository open source separato. È probabile che il ciclo di vita e la fase di supporto di ciascuna versione varino tra i repository. Ad esempio, una determinata versione di AWS Encryption SDK potrebbe trovarsi nella fase di disponibilità generale (supporto completo) in un linguaggio di programmazione, ma la end-of-support fase in un linguaggio di programmazione diverso. Ti consigliamo di utilizzare una versione completamente supportata ogni volta che è possibile ed evitare versioni che non sono più supportate.

Per trovare la fase del ciclo di vita delle AWS Encryption SDK versioni per il tuo linguaggio di programmazione, consulta il `SUPPORT_POLICY.rst` file in ogni AWS Encryption SDK repository.

- SDK di crittografia AWS per C— [Support\\_policy.rst](#)
- AWS Encryption SDK per .NET — [Support\\_Policy.rst](#)
- AWS CLI di crittografia — [Support\\_policy.rst](#)
- SDK di crittografia AWS per Java— [Support\\_policy.rst](#)
- SDK di crittografia AWS per JavaScript— [Support\\_policy.rst](#)
- SDK di crittografia AWS per Python— [Support\\_policy.rst](#)

Per ulteriori informazioni, consulta la [politica di manutenzione degli AWS SDK Versioni del AWS Encryption SDK e degli strumenti](#) nella Guida di riferimento agli SDK e agli AWS strumenti.

## Ulteriori informazioni

Se hai bisogno di ulteriori informazioni su AWS Encryption SDK e la crittografia lato client, prova queste origini.

- Per un aiuto su termini e nozioni utilizzati in questo SDK, consulta [Concetti in AWS Encryption SDK](#).
- Per le linee guida sulle migliori pratiche, consulta [Best practice per AWS Encryption SDK](#).
- Per ulteriori informazioni sul funzionamento di questo SDK, consulta [Come funziona l'SDK](#).
- Per esempi che mostrano come configurare le opzioni in AWS Encryption SDK, vedere [Configurazione della AWS Encryption SDK](#).
- Per informazioni tecniche dettagliate consulta [Riferimento](#).
- Per le specifiche tecniche di AWS Encryption SDK, vedere le [AWS Encryption SDK Specifiche](#) in GitHub.
- Per le risposte alle tue domande sull'utilizzo di AWS Encryption SDK, leggi e pubblica sul [forum di discussione di AWS Crypto Tools](#).

Per informazioni sulle implementazioni di AWS Encryption SDK in diversi linguaggi di programmazione.

- C: Vedi [SDK di crittografia AWS per C](#) la [documentazione AWS Encryption SDK C](#) e il [aws-encryption-sdk-repository](#) su GitHub.
- C#.NET: vedi [AWS Encryption SDK per .NET](#) e attiva la [aws-encryption-sdk-net](#) directory del repository. [aws-encryption-sdk-dafny](#) GitHub
- Interfaccia a riga di comando: vedi [Interfaccia a riga di comando AWS Encryption SDK](#), [leggi i documenti](#) per la CLI di AWS crittografia e [aws-encryption-sdk-cli](#) il repository su. GitHub
- Java: vedi [SDK di crittografia AWS per Java](#), AWS Encryption SDK [Javadoc](#) e il repository attivo. [aws-encryption-sdk-java](#) GitHub
- JavaScript: Vedi [the section called "JavaScript"](#) e accendi il repository. [aws-encryption-sdk-javascript](#) GitHub
- Python: vedi [SDK di crittografia AWS per Python](#), la [documentazione di AWS Encryption SDK Python](#) e il repository su. [aws-encryption-sdk-python](#) GitHub

## Invio di feedback

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

- Se scopri una potenziale vulnerabilità della sicurezza in AWS Encryption SDK, ti preghiamo di [informare i responsabili della sicurezza di AWS](#). Non create un problema pubblico. GitHub
- Per fornire un feedback su AWS Encryption SDK, segnala un problema nell' GitHub archivio del linguaggio di programmazione che stai utilizzando.
- Per fornire feedback su questa documentazione, utilizzate i collegamenti Feedback disponibili in questa pagina. Puoi anche segnalare un problema o contribuire al [aws-encryption-sdk-docs](#) repository open source di questa documentazione. GitHub

## Concetti in AWS Encryption SDK

Questa sezione introduce i concetti utilizzati in AWS Encryption SDK, e fornisce un glossario e un riferimento. È progettato per aiutarti a capire come AWS Encryption SDK funziona e i termini che utilizziamo per descriverlo.

Serve aiuto?

- Scopri come AWS Encryption SDK utilizza la [crittografia a busta](#) per proteggere i tuoi dati.
- Scopri gli elementi della crittografia in busta: [le chiavi dati](#) che proteggono i tuoi dati e le chiavi di [avvolgimento che proteggono le tue chiavi](#) dati.
- Scopri i [portachiavi e i fornitori di chiavi principali](#) che determinano le chiavi di avvolgimento da utilizzare.
- Scopri il [contesto di crittografia](#) che aggiunge integrità al tuo processo di crittografia. È facoltativo, ma è una best practice che consigliamo.
- Scopri il [messaggio crittografato](#) restituito dai metodi di crittografia.
- Quindi sei pronto per utilizzarlo AWS Encryption SDK nel tuo [linguaggio di programmazione](#) preferito.

Argomenti

- [Crittografia envelope](#)
- [Chiave di dati](#)
- [Chiave di avvolgimento](#)
- [Portachiavi e fornitori di chiavi principali](#)
- [Contesto di crittografia](#)
- [Messaggio crittografato](#)

- [Suite di algoritmi](#)
- [Responsabile di materiali crittografici](#)
- [Crittografia simmetrica e asimmetrica](#)
- [Impegno chiave](#)
- [Politica di impegno](#)
- [Firme digitali](#)

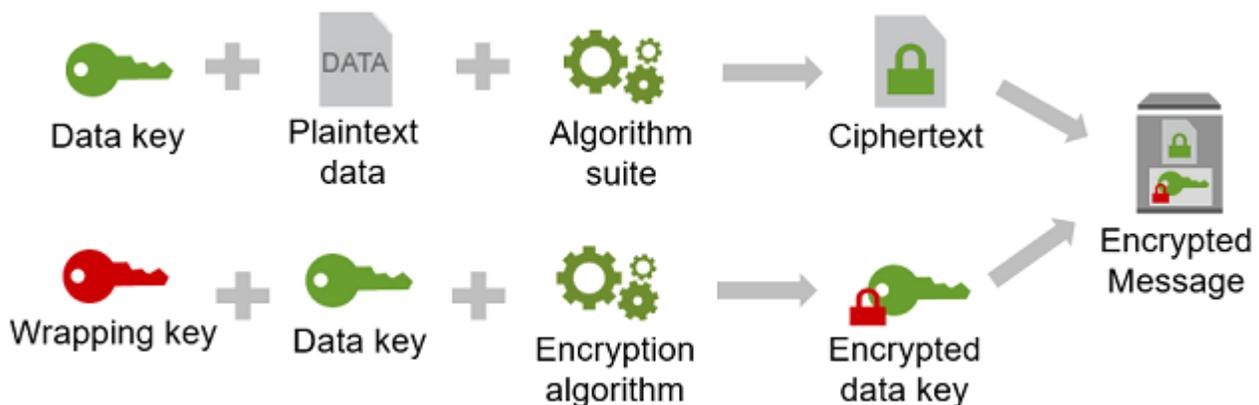
## Crittografia envelope

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

### Protezione delle chiavi dei dati

AWS Encryption SDK crittografa ogni messaggio con una chiave dati unica. Quindi crittografa la chiave dati sotto la chiave di avvolgimento specificata. Memorizza la chiave dati crittografata con i dati crittografati nel messaggio crittografato che restituisce.

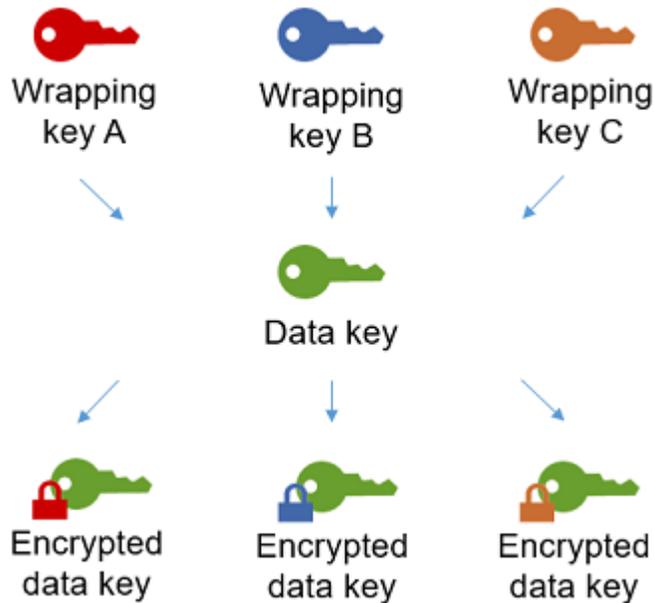
Per specificare la chiave di avvolgimento, si utilizza un [portachiavi o un fornitore](#) di chiavi [principali](#).



## Crittografia degli stessi dati con più chiavi di avvolgimento

È possibile crittografare la chiave dati con più chiavi di wrapping. Potresti voler fornire chiavi di avvolgimento diverse per utenti diversi, oppure chiavi di avvolgimento di tipi diversi o in posizioni diverse. Ciascuna delle chiavi di wrapping crittografa la stessa chiave di dati. AWS Encryption SDK memorizza tutte le chiavi dati crittografate con i dati crittografati nel messaggio crittografato.

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



## Abbinare i punti di forza di più algoritmi

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

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

## Chiave di dati

Una chiave di dati è una chiave di crittografia che l'AWS Encryption SDK utilizza per crittografare i dati. Ogni chiave di dati corrisponde a un array di byte conforme ai requisiti per le chiavi di crittografia. Se non scegli il [caching della chiave di dati](#), l'AWS Encryption SDK utilizza una chiave di dati univoca per crittografare ogni messaggio.

Non è necessario specificare, generare, implementare, estendere, proteggere o utilizzare chiavi dati. L'AWS Encryption SDK si occupa di tutte queste attività quando chiami le operazioni di crittografia e decrittazione.

Per proteggere le chiavi dati, le AWS Encryption SDK crittografa utilizzando una o più chiavi di crittografia a chiave note come chiavi di [wrapping o chiavi master](#). Dopo che l'AWS Encryption SDK ha utilizzato le chiavi di dati di testo normale per crittografare i dati, le elimina dalla memoria appena possibile. Quindi, archivia le chiavi di dati con i dati crittografati nel [messaggio crittografato](#) restituito dalle operazioni di crittografia. Per informazioni dettagliate, consultare [the section called "Come funziona l'SDK"](#).

### Tip

In AWS Encryption SDK, distinguiamo le chiavi dei dati dalle chiavi di crittografia dei dati. Molteplici [suite di algoritmi](#) supportate, tra cui la suite predefinita, utilizzano una [funzione di derivazione della chiave](#) che impedisce alla chiave dei dati di toccare i limiti crittografici. La funzione di derivazione della chiave richiede la chiave di dati come input e restituisce una chiave di crittografia dei dati effettivamente utilizzata per crittografare i dati. Per questo motivo, abbiamo spesso detto che i dati sono crittografati "in" una chiave dei dati anziché "da" una chiave di dati.

Ogni chiave di dati crittografata include metadati, incluso l'identificatore della chiave di wrapping che l'ha crittografata. Questi metadati facilitano l'identificazione di chiavi di wrapping valide AWS Encryption SDK durante la decrittografia.

## Chiave di avvolgimento

Una chiave di wrapping è una chiave di crittografia a chiave AWS Encryption SDK utilizzata per crittografare la [chiave dati che crittografa i dati](#). Ogni chiave di dati in testo semplice può essere crittografata con una o più chiavi di wrapping. [Sei tu a determinare quali chiavi di wrapping vengono utilizzate per proteggere i tuoi dati quando configuri un portachiavi o un fornitore di chiavi master.](#)

**Note**

La chiave di avvolgimento si riferisce alle chiavi di un portachiavi o di un fornitore di chiavi principali. La chiave master è in genere associata alla `MasterKey` classe di cui si crea un'istanza quando si utilizza un provider di chiavi master.

AWS Encryption SDKSupporta diverse chiavi di wrapping di uso comune, come AWS Key Management Service (AWS KMS) simmetriche (incluse le chiavi [KMS multiregione](#)), [chiavi AES-GCM AWS KMS keys](#)(Advanced Encryption Standard/Galois Counter Mode) non elaborate e chiavi RSA non elaborate. Puoi anche estendere o implementare le tue chiavi di wrapping.

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

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

Se non disponi di un sistema di gestione delle chiavi, ti consigliamo AWS KMS. AWS Encryption SDKSi integra con AWS KMS per aiutarti a proteggere e utilizzare le tue chiavi di imballaggio. Tuttavia, AWS Encryption SDK non richiede alcun AWS AWS servizio.

## Portachiavi e fornitori di chiavi principali

Per specificare le chiavi di wrapping utilizzate per la crittografia e la decrittografia, si utilizza un portachiavi (C, C# / .NET e JavaScript) o un provider di chiavi master (Java, Python, CLI). È possibile utilizzare i portachiavi e i provider di chiavi master che fornisce o progettare implementazioni personalizzate. AWS Encryption SDK AWS Encryption SDKFornisce portachiavi e fornitori di chiavi principali compatibili tra loro soggetti a vincoli linguistici. Per informazioni dettagliate, consultare [Compatibilità dei keyring](#).

Un keyring genera, crittografa e decrittografa le chiavi di dati. Quando definisci un portachiavi, puoi specificare le chiavi di [avvolgimento che crittografano le tue chiavi dati](#). La maggior parte dei portachiavi specifica almeno una chiave di avvolgimento o un servizio che fornisce e protegge le chiavi di avvolgimento. È inoltre possibile definire un portachiavi senza chiavi avvolgenti o un

portachiavi più complesso con opzioni di configurazione aggiuntive. Per informazioni sulla scelta e sull'utilizzo dei portachiavi che definisce, consulta AWS Encryption SDK. [Utilizzo dei keyring](#) I portachiavi sono supportati in C, C#/.NET e nella versione JavaScript 3.x di SDK di crittografia AWS per Java

Un fornitore di chiavi principali è un'alternativa a un portachiavi. Il provider di chiavi master restituisce le chiavi di wrapping (o chiavi master) specificate. Ogni chiave master è associata a un provider di chiavi master, ma un provider di chiavi master in genere fornisce più chiavi master. I provider di chiavi principali sono supportati in Java, Python e nella AWS CLI di crittografia.

È necessario specificare un portachiavi (o un provider di chiavi principali) per la crittografia. È possibile specificare lo stesso portachiavi (o fornitore di chiavi principali) o uno diverso per la decrittografia. Durante la crittografia, AWS Encryption SDK utilizza tutte le chiavi di wrapping specificate per crittografare la chiave dati. Durante la decrittografia, AWS Encryption SDK utilizza solo le chiavi di wrapping specificate per decrittografare una chiave dati crittografata. [Specificare le chiavi di wrapping per la decrittografia è facoltativo, ma è una procedura consigliata.](#) AWS Encryption SDK

Per informazioni dettagliate sulla specificazione delle chiavi di wrapping, consulta. [Selezione dei tasti di avvolgimento](#)

## Contesto di crittografia

Per migliorare la sicurezza delle operazioni di crittografia, includi un [contesto di crittografia](#) in tutte le richieste di crittografia dei dati. L'utilizzo di un contesto di crittografia è facoltativo, ma viene consigliato come best practice.

Un contesto di crittografia è un set di coppie nome-valore che contiene dati autenticati aggiuntivi arbitrari e non segreti. Il contesto di crittografia può contenere qualsiasi tipo di dati scelto, ma in genere consiste in dati utili in registrazioni e monitoraggi, ad esempio i dati relativi al tipo di file, allo scopo o al proprietario. Durante la crittografia dei dati, il contesto di crittografia viene vincolato a livello crittografico ai dati crittografati, per cui dovrai utilizzare lo stesso contesto per decrittarli. L'AWS Encryption SDK include il contesto di crittografia in testo normale nell'intestazione del [messaggio crittografato](#) restituito.

Il contesto di crittografia AWS Encryption SDK utilizzato è costituito dal contesto di crittografia specificato e da una coppia di chiavi pubblica aggiunta dal [gestore dei materiali crittografici](#) (CMM). Nello specifico, ogni volta che utilizzi un [algoritmo di crittografia con firma](#), il CMM aggiunge una coppia nome-valore al contesto di crittografia, costituita da un nome riservato, `aws-crypto-public-key` e un valore che rappresenta la chiave di verifica pubblica. Il nome `aws-crypto-`

`public-key` nel contesto di crittografia è riservato dall'AWS Encryption SDK e non può essere utilizzato come nome di qualsiasi altra coppia nello stesso contesto. Per informazioni dettagliate, consulta [AAD](#) in Riferimenti a formati di messaggi.

Il contesto di crittografia nell'esempio seguente include due coppie del contesto specificato nella richiesta e una coppia di chiavi pubblica aggiunta dal CMM.

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

Per decrittografare i dati, è necessario passare il messaggio crittografato. Poiché l'AWS Encryption SDK può estrarre il contesto di crittografia dall'intestazione del messaggio crittografato, non devi fornirlo separatamente. Tuttavia, grazie al contesto di crittografia puoi verificare di stare decrittando il messaggio corretto.

- Nell'[interfaccia a riga di comando \(CLI\) dell'AWS Encryption SDK](#), se fornisci un contesto di crittografia in un comando di decrittazione, la CLI verifica che i valori siano presenti nel contesto del messaggio crittografato prima di restituire i dati di testo normale.
- In altre implementazioni del linguaggio di programmazione, la risposta di decrittografia include il contesto di crittografia e i dati in chiaro. La funzione di decrittazione nell'applicazione deve sempre verificare che il contesto di crittografia nella risposta di decrittazione includa il contesto della relativa richiesta (o in un sottoinsieme) prima di restituire i dati di testo normale.

#### Note

[Con la versione 4. x di AWS Encryption SDK per .NET e versione 3. x di SDK di crittografia AWS per Java](#), è possibile richiedere un contesto di crittografia in tutte le richieste di crittografia con il contesto di crittografia richiesto CMM.

Quando scegli un contesto di crittografia, ricordati che non è segreto. Il contesto di crittografia viene visualizzato in testo semplice nell'intestazione del messaggio [crittografato](#) restituito. AWS Encryption SDK Se stai usando AWS Key Management Service, il contesto di crittografia può comparire in testo non crittografato nei record e nei log di controllo, ad esempio AWS CloudTrail.

[Per esempi di invio e verifica di un contesto di crittografia nel codice, consulta gli esempi relativi al linguaggio di programmazione preferito.](#)

## Messaggio crittografato

Quando si crittografano i dati con AWS Encryption SDK, viene restituito un messaggio crittografato.

[Un messaggio crittografato è una struttura di dati formattata portatile che include i dati crittografati insieme a copie crittografate delle chiavi dati, l'ID dell'algoritmo e, facoltativamente, un contesto di crittografia e una firma digitale.](#) Le operazioni di crittografia in AWS Encryption SDK restituiscono un messaggio crittografato, mentre le operazioni di decrittazione richiedono un messaggio crittografato come input.

Abbinando i dati crittografati e le relative chiavi di dati crittografati è possibile semplificare l'operazione di decrittografia e ti consente di evitare di archiviare e gestire chiavi di dati crittografate in modo indipendente rispetto ai dati che crittografano.

Per informazioni tecniche sui messaggi crittografati, vedi [Formato dei messaggi crittografati](#).

## Suite di algoritmi

AWS Encryption SDK utilizza una suite di algoritmi per crittografare e firmare i dati contenuti nel [messaggio crittografato](#) restituito dalle operazioni di crittografia e decrittografia. AWS Encryption SDK supporta diverse [suite di algoritmi](#). Tutte le suite supportate utilizzano Advanced Encryption Standard (AES) come algoritmo primario e lo combinano con altri algoritmi e valori.

AWS Encryption SDK stabilisce una suite di algoritmi consigliata come predefinita per tutte le operazioni di crittografia. L'impostazione predefinita potrebbe cambiare con il migliorare di standard e best practice. È possibile specificare una suite di algoritmi alternativa nelle richieste di crittografia dei dati o durante la creazione di un [gestore di materiali crittografici \(CMM\)](#), ma a meno che non sia richiesta un'alternativa per la propria situazione, è preferibile utilizzare quella predefinita. L'impostazione predefinita attuale è AES-GCM con una [funzione di derivazione delle extract-and-expand chiavi \(HKDF\) basata su HMAC](#), [key commitment](#), una firma [Elliptic Curve Digital Signature Algorithm \(ECDSA\)](#) e una chiave di crittografia a 256 bit.

Se la tua applicazione richiede prestazioni elevate e gli utenti che crittografano i dati e quelli che decifrano i dati sono altrettanto affidabili, potresti prendere in considerazione la possibilità di specificare una suite di algoritmi senza firma digitale. Tuttavia, consigliamo vivamente una suite di algoritmi che includa l'impegno delle chiavi e una funzione di derivazione delle chiavi. Le suite di algoritmi senza queste funzionalità sono supportate solo per la compatibilità con le versioni precedenti.

## Responsabile di materiali crittografici

Il gestore dei materiali crittografici (CMM) assembla i materiali crittografici utilizzati per crittografare e decrittografare i dati. I materiali di crittografia includono chiavi di dati crittografati e testo non crittografato e una chiave facoltativa di firma dei messaggi. Non interagisci mai direttamente con la CMM. I metodi di crittografia e decrittazione lo gestiscono per te.

È possibile utilizzare la CMM predefinita o la CMM di [memorizzazione nella cache AWS Encryption SDK fornita oppure scrivere una CMM](#) personalizzata. E puoi specificare una CMM, ma non è obbligatoria. Quando specifichi un portachiavi o un fornitore di chiavi principali, AWS Encryption SDK crea automaticamente una CMM predefinita. La CMM predefinita ottiene i materiali di crittografia o decrittografia dal portachiavi o dal provider di chiavi master specificato. Ciò potrebbe comportare una chiamata a un servizio crittografico, come [AWS Key Management Service](#) (AWS KMS).

Poiché la CMM funge da collegamento tra il AWS Encryption SDK e un portachiavi (o fornitore di chiavi principali), è il punto ideale per la personalizzazione e l'estensione, ad esempio il supporto per l'applicazione delle policy e la memorizzazione nella cache. AWS Encryption SDK fornisce un CMM di caching per supportare il [caching della chiave di dati](#).

## Crittografia simmetrica e asimmetrica

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

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

AWS Encryption SDK [Utilizza la crittografia a busta](#). Crittografa i dati con una chiave dati simmetrica. Crittografa la chiave dati simmetrica con una o più chiavi di avvolgimento simmetriche o asimmetriche. Restituisce un [messaggio crittografato](#) che include i dati crittografati e almeno una copia crittografata della chiave dati.

### Crittografia dei dati (crittografia simmetrica)

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

## Crittografia della chiave dati (crittografia simmetrica o asimmetrica)

Il [portachiavi](#) o il [fornitore di chiavi master](#) fornito per un'operazione di crittografia e decrittografia determina il modo in cui la chiave dati simmetrica viene crittografata e decrittografata. Puoi scegliere un portachiavi o un provider di chiavi master che utilizza la crittografia simmetrica, ad esempio un portachiavi, o uno che utilizza la crittografia asimmetrica, come un AWS KMS portachiavi RSA non elaborato o. `JceMasterKey`

## Impegno chiave

AWS Encryption SDK supporta key commitment (a volte nota come robustezza), una proprietà di sicurezza che garantisce che ogni testo cifrato possa essere decrittografato solo in un singolo testo in chiaro. A tale scopo, key commitment garantisce che solo la chiave dati che ha crittografato il messaggio verrà utilizzata per decrittografarlo. [La crittografia e la decrittografia con impegno chiave è una best practice. AWS Encryption SDK](#)

La maggior parte dei cifrari simmetrici moderni (incluso AES) crittografano un testo in chiaro con un'unica chiave segreta, ad esempio la chiave [dati univoca](#) utilizzata per crittografare ogni messaggio di testo in chiaro. AWS Encryption SDK La decrittografia di questi dati con la stessa chiave di dati restituisce un testo in chiaro identico all'originale. La decrittografia con una chiave diversa di solito fallisce. Tuttavia, è possibile decrittografare un testo cifrato con due chiavi diverse. In rari casi, è possibile trovare una chiave in grado di decrittografare alcuni byte di testo cifrato in un testo semplice diverso, ma comunque comprensibile.

AWS Encryption SDK Crittografano sempre ogni messaggio di testo in chiaro con un'unica chiave di dati. Potrebbe crittografare quella chiave dati con più chiavi di wrapping (o chiavi master), ma le chiavi di wrapping crittografano sempre la stessa chiave dati. Tuttavia, un [messaggio crittografato sofisticato creato manualmente potrebbe effettivamente contenere diverse chiavi di dati, ognuna crittografata da una chiave di wrapping diversa](#). Ad esempio, se un utente decrittografa il messaggio crittografato, restituisce 0x0 (falso) mentre un altro utente che decrittografa lo stesso messaggio crittografato ottiene 0x1 (vero).

Per evitare questo scenario, supporta l'impegno chiave durante la crittografia e la decrittografia AWS Encryption SDK. Quando AWS Encryption SDK crittografano un messaggio con impegno chiave, associa criticograficamente la chiave di dati univoca che ha prodotto il testo cifrato alla stringa di impegno chiave, un identificatore di chiave di dati non segreto. Quindi memorizza la stringa di impegno chiave nei metadati del messaggio crittografato. Quando decifra un messaggio con impegno chiave, AWS Encryption SDK verifica che la chiave dati sia l'unica chiave per quel

messaggio crittografato. Se la verifica della chiave dati non riesce, l'operazione di decrittografia ha esito negativo.

Il supporto per Key Commitment è stato introdotto nella versione 1.7. x, che può decrittografare i messaggi con un impegno chiave, ma non lo farà con un impegno chiave. È possibile utilizzare questa versione per implementare completamente la capacità di decrittografare il testo cifrato con impegno chiave. Versione 2.0. x include il supporto completo per Key Commitment. Per impostazione predefinita, crittografa e decrittografa solo con l'impegno della chiave. Questa è una configurazione ideale per le applicazioni che non necessitano di decrittografare il testo cifrato crittografato dalle versioni precedenti di AWS Encryption SDK.

Sebbene la crittografia e la decrittografia con impegno chiave siano una best practice, lasciamo che sia tu a decidere quando utilizzarla e a regolare il ritmo con cui adottarla. A partire dalla versione 1.7. x, AWS Encryption SDK supporta una [politica di impegno](#) che imposta la [suite di algoritmi predefinita](#) e limita le suite di algoritmi che possono essere utilizzate. Questa politica determina se i dati vengono crittografati e decrittografati con un impegno chiave.

Key Commitment si traduce in un [messaggio crittografato leggermente più grande \(+ 30 byte\)](#) e richiede più tempo per l'elaborazione. Se l'applicazione è molto sensibile alle dimensioni o alle prestazioni, è possibile scegliere di disattivare l'impegno chiave. Ma fatelo solo se necessario.

Per ulteriori informazioni sulla migrazione alle versioni 1.7. x e 2.0. x, incluse le loro principali funzionalità di impegno, vedi [Migrazione della tua AWS Encryption SDK](#). Per informazioni tecniche su Key Commitment, vedere [the section called “Riferimenti agli algoritmi”](#) e [the section called “Riferimenti a formati di messaggi”](#).

## Politica di impegno

[Una policy di impegno è un'impostazione di configurazione che determina se l'applicazione esegue la crittografia e la decrittografia con un impegno chiave. La crittografia e la decrittografia con impegno chiave è una best practice. AWS Encryption SDK](#)

La politica di impegno ha tre valori.

### Note

Potrebbe essere necessario scorrere orizzontalmente o verticalmente per visualizzare l'intera tabella.

## Impegno, politica, valori

Valore	Crittografa con impegno chiave	Crittografa senza impegno chiave	Deccripta con impegno chiave	Deccripta senza impegno chiave
ForbidEncryptAllowDecrypt				
RequireEncryptAllowDecrypt				
RequireEncryptRequireDecrypt				

L'impostazione della politica di impegno è stata introdotta nella AWS Encryption SDK versione 1.7. x. È valido in tutti i [linguaggi di programmazione](#) supportati.

- `ForbidEncryptAllowDecrypt` crittografa con o senza impegno chiave, ma non crittografa con impegno chiave. Questo è l'unico valore valido per la politica di impegno nella versione 1.7. x e viene utilizzato per tutte le operazioni di crittografia e decrittografia. È progettato per preparare tutti gli host che eseguono l'applicazione alla decrittografia con impegno chiave prima che incontrino un testo cifrato crittografato con impegno chiave.
- `RequireEncryptAllowDecrypt` crittografa sempre con impegno chiave. Può decrittare con o senza impegno chiave. Questo valore, introdotto nella versione 2.0. x, consente di iniziare a crittografare con l'impegno delle chiavi, ma di decrittografare comunque i testi cifrati precedenti senza l'impegno di chiave.
- `RequireEncryptRequireDecrypt` crittografa e decrittografa solo con un impegno chiave. Questo valore è l'impostazione predefinita per la versione 2.0. x. Usa questo valore quando sei certo che tutti i tuoi testi cifrati siano crittografati con l'impegno della chiave.

L'impostazione della politica di impegno determina quali suite di algoritmi è possibile utilizzare. A partire dalla versione 1.7. x, AWS Encryption SDK supporta [suite di algoritmi](#) per l'impegno chiave,

con e senza firma. Se specificate una suite di algoritmi che è in conflitto con la vostra politica di impegno, AWS Encryption SDK restituisce un errore.

Per informazioni sull'impostazione della politica di impegno, consulta [Impostazione della politica di impegno](#).

## Firme digitali

Per garantire l'integrità di un messaggio digitale durante lo spostamento tra i sistemi, è possibile applicare una firma digitale al messaggio. Le firme digitali sono sempre asimmetriche. La chiave privata viene utilizzata per creare la firma e aggiungerla al messaggio originale. Il destinatario utilizza una chiave pubblica per verificare che il messaggio non sia stato modificato dopo la firma.

AWS Encryption SDK crittografa i dati utilizzando un algoritmo di crittografia autenticato, AES-GCM, e il processo di decrittografia verifica l'integrità e l'autenticità di un messaggio crittografato senza utilizzare una firma digitale. Tuttavia, poiché AES-GCM utilizza chiavi simmetriche, chiunque sia in grado di decrittografare la chiave dati utilizzata per decrittografare il testo cifrato potrebbe anche creare manualmente un nuovo testo cifrato crittografato, causando potenziali problemi di sicurezza. Ad esempio, se si utilizza una AWS KMS chiave come chiave di avvolgimento, ciò significa che è possibile per un utente con autorizzazioni KMS Decrypt creare testi cifrati crittografati senza chiamare KMS Encrypt.

Per evitare questo problema, AWS Encryption SDK supporta l'aggiunta di una firma Elliptic Curve Digital Signature Algorithm (ECDSA) alla fine dei messaggi crittografati. Quando viene utilizzata una suite di algoritmi di firma, AWS Encryption SDK genera una chiave privata temporanea e una coppia di chiavi pubbliche per ogni messaggio crittografato. AWS Encryption SDK memorizza la chiave pubblica nel contesto di crittografia della chiave dati e scarta la chiave privata, e nessuno può creare un'altra firma che verifichi con la chiave pubblica. Poiché l'algoritmo associa la chiave pubblica alla chiave dati crittografata come dati autenticati aggiuntivi nell'impostazione del messaggio, un utente che può solo decrittografare i messaggi non può alterare la chiave pubblica.

La verifica della firma comporta un notevole costo in termini di prestazioni per la decrittografia. Se gli utenti che crittografano i dati e gli utenti che decifrano i dati sono altrettanto affidabili, prendi in considerazione l'utilizzo di una suite di algoritmi che non includa la firma.

## Come funziona AWS Encryption SDK

[I flussi di lavoro di questa sezione spiegano come crittografa i dati e decrittografa i AWS Encryption SDK messaggi crittografati.](#) Questi flussi di lavoro descrivono il processo di base utilizzando le

funzionalità predefinite. Per i dettagli sulla definizione e l'utilizzo di componenti personalizzati, consulta l' [GitHub](#) archivio per ogni implementazione [linguistica](#) supportata.

AWS Encryption SDK Utilizza la crittografia a busta per proteggere i dati. Ogni messaggio è crittografato con una chiave dati unica. Quindi la chiave dati viene crittografata dalle chiavi di wrapping specificate. Per decrittografare il messaggio crittografato, AWS Encryption SDK utilizza le chiavi di wrapping specificate per decrittografare almeno una chiave di dati crittografata. Quindi può decrittografare il testo cifrato e restituire un messaggio in testo semplice.

Hai bisogno di aiuto con la terminologia che utilizziamo in? AWS Encryption SDK Per informazioni, consulta [the section called “Concetti”](#).

## In che modo AWS Encryption SDK crittografa i dati

AWS Encryption SDK Fornisce metodi per crittografare stringhe, array di byte e flussi di byte. Per esempi di codice, consultate l'argomento Esempi in ogni sezione. [Linguaggi di programmazione](#)

1. Crea un [portachiavi](#) (o [fornitore di chiavi principali](#)) che specifichi le chiavi di avvolgimento che proteggono i tuoi dati.
2. Passa il portachiavi e i dati in testo semplice a un metodo di crittografia. [Ti consigliamo di passare in un contesto di crittografia opzionale e non segreto.](#)
3. Il metodo di crittografia richiede al portachiavi i materiali di crittografia. Il portachiavi restituisce chiavi di crittografia dei dati univoche per il messaggio: una chiave dati in testo semplice e una copia di tale chiave dati crittografata da ciascuna delle chiavi di wrapping specificate.
4. Il metodo di crittografia usa la chiave di dati di testo non crittografato per crittografare i dati, quindi elimina la chiave di dati di testo non crittografato. Se si fornisce un contesto di crittografia (una [procedura AWS Encryption SDK consigliata](#)), il metodo di crittografia associa criticograficamente il contesto di crittografia ai dati crittografati.
5. Il metodo di crittografia restituisce un [messaggio crittografato](#) che contiene i dati crittografati, le chiavi dei dati crittografati e altri metadati, incluso il contesto di crittografia, se ne hai utilizzato uno.

## Come AWS Encryption SDK decripta un messaggio crittografato

AWS Encryption SDK Fornisce metodi che decrittografano il [messaggio crittografato](#) e restituiscono testo in chiaro. Per esempi di codice, consultate l'argomento Esempi in ogni sezione. [Linguaggi di programmazione](#)

Il [portachiavi](#) (o [provider di chiavi master](#)) che decrittografa il messaggio crittografato deve essere compatibile con quello utilizzato per crittografare il messaggio. Una delle sue chiavi di wrapping deve essere in grado di decrittografare una chiave di dati crittografata nel messaggio crittografato. Per informazioni sulla compatibilità con i portachiavi e i fornitori di chiavi principali, vedere [the section called “Compatibilità dei keyring”](#)

1. Crea un portachiavi o un fornitore di chiavi master con chiavi avvolgenti in grado di decrittografare i tuoi dati. Puoi utilizzare lo stesso portachiavi che hai fornito per il metodo di crittografia o uno diverso.
2. [Passate il messaggio crittografato](#) e il portachiavi a un metodo di decrittografia.
3. Il metodo di decrittografia richiede al portachiavi o al fornitore della chiave principale di decrittografare una delle chiavi di dati crittografate nel messaggio crittografato. Trasferisce le informazioni dal messaggio crittografato, incluse le chiavi di dati crittografate.
4. Il keyring utilizza le chiavi di wrapping per decrittare una delle chiavi di dati crittografate. Se ha esito positivo, la risposta include la chiave di dati in testo semplice. Se nessuna delle chiavi di wrapping specificate dal keyring o dal provider di chiavi master è in grado di decrittografare una chiave dati crittografata, la chiamata di decrittografia ha esito negativo.
5. Il metodo di decrittografia utilizza la chiave dati in testo semplice per decrittografare i dati, scarta la chiave di dati in testo semplice e restituisce i dati in chiaro.

## Suite di algoritmi supportate in AWS Encryption SDK

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

La suite di algoritmi AWS Encryption SDK utilizza l'algoritmo Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM), noto come AES-GCM, per crittografare i dati grezzi. LaAWS Encryption SDKsupporta chiavi di crittografia a 256 bit, 192 bit e 128 bit. La lunghezza del vettore di inizializzazione (IV) è sempre 12 byte. La lunghezza del tag di autenticazione è sempre 16 byte.

Per impostazione predefinita, ilAWS Encryption SDKutilizza una suite di algoritmi con AES-GCM con un HMACextract-and-expandfunzione di derivazione chiave ([HKDF](#)), firma e chiave di crittografia a 256 bit. Se il file[politica impegno](#)richiede[chiave impegno](#), ilAWS Encryption SDKseleziona una suite di algoritmi che supporta anche l'impegno chiave; in caso contrario, seleziona una suite di algoritmi con derivazione e firma delle chiavi, ma non un impegno chiave.

## Consigliato: AES-GCM con derivazione della chiave, firma e impegno

LaAWS Encryption SDKconsiglia una suite di algoritmi che deriva una chiave di crittografia AES-GCM fornendo una chiave di crittografia dei dati a 256 bit a quella basata su HMACextract-and-expandfunzione di derivazione delle chiavi (HKDF). LaAWS Encryption SDKaggiunge una firma ECDSA (Elliptic Curve Digital Signature Algorithm). Per supportare[chiave impegno](#), questa suite di algoritmi deriva anche astringa di impegno chiave— un identificatore della chiave di dati non segreto, memorizzato nei metadati del messaggio crittografato. Questa stringa di impegno chiave è derivata anche da HKDF utilizzando una procedura simile alla derivazione della chiave di crittografia dei dati.

### AWS Encryption SDKSuite di algoritmi

Algoritmo di crittografia	Lunghezza della chiave di crittografia dei dati (in bit)	Algoritmo di derivazione della chiave	Algoritmo di firma	Scadenza impegno
AES-GCM	256	HKDF con SHA-384	ECDSA con P-384 e SHA-384	HKDF con SHA-512

L'HKDF consente di evitare il riutilizzo accidentale di una chiave di crittografia dei dati e riduce il rischio di utilizzo eccessivo di una chiave di dati.

Per la firma, questa suite di algoritmi utilizza ECDSA con un algoritmo di funzione hash crittografica (SHA-384). ECDSA viene utilizzato per impostazione predefinita, anche quando non è specificato dalla policy per la chiave master sottostante. [Firma di messaggi](#) verifica che il mittente del messaggio sia stato autorizzato a crittografare i messaggi e fornisce il non ripudio. È particolarmente utile quando la policy di autorizzazione per una chiave master consente a un set di utenti di crittografare i dati e a un altro set di utenti di decrittografarli.

Le suite di algoritmi con impegno chiave assicurano che ogni testo cifrato decifri in un solo testo normale. Lo fanno convalidando l'identità della chiave dati utilizzata come input per l'algoritmo di crittografia. Durante la crittografia, queste suite di algoritmi derivano una stringa di impegno chiave. Prima di decrittografare, convalidano che la chiave dati corrisponda alla stringa di impegno chiave. In caso contrario, la chiamata di decrittografia ha esito negativo.

## Altre suite di algoritmi supportate

L'AWS Encryption SDK supporta le seguenti suite di algoritmi alternative per la compatibilità con le versioni precedenti, di cui, in generale, sconsigliamo l'utilizzo. Tuttavia, riconosciamo che la firma può ostacolare in modo significativo le prestazioni, quindi offriamo una suite di commit chiave con derivazione delle chiavi per questi casi. Per le applicazioni che devono rendere più significativi compromessi sulle prestazioni, continuiamo a offrire suite prive di firma, impegno chiave e derivazione delle chiavi.

### AES-GCM senza impegno

Le suite di algoritmi senza impegno chiave non convalidano la chiave dati prima della decrittografia. Di conseguenza, queste suite di algoritmi potrebbero decrittografare un singolo testo cifrato in diversi messaggi in testo normale. Tuttavia, perché le suite di algoritmi con impegno chiave producono un [messaggio crittografato leggermente più grande \(+30 byte\)](#) e richiedere più tempo per l'elaborazione, potrebbero non essere la scelta migliore per ogni applicazione.

La AWS Encryption SDK supporta una suite di algoritmi con derivazione delle chiavi, impegno chiave, firma e uno con derivazione chiave e impegno chiave, ma non firma. Non è consigliabile utilizzare una suite di algoritmi senza impegno. Se necessario, consigliamo una suite di algoritmi con derivazione delle chiavi e impegno chiave, ma non firmare. Tuttavia, se il profilo delle prestazioni dell'applicazione supporta l'utilizzo di una suite di algoritmi, l'utilizzo di una suite di algoritmi con impegno chiave, derivazione di chiavi e firma è una buona pratica.

### AES-GCM senza firma

Le suite di algoritmi senza firma non forniscono la firma ECDSA che offre autenticità e non ripudio. Utilizzare queste suite solo quando gli utenti che crittografano i dati e quelli che decrittografano i dati sono ugualmente affidabili.

Quando si utilizza una suite di algoritmi senza firmare, si consiglia di sceglierne una con derivazione chiave e impegno chiave.

### AES-GCM senza derivazione della chiave

Le suite di algoritmi senza derivazione della chiave utilizzano la chiave di crittografia dei dati come chiave di crittografia AES-GCM, anziché utilizzare una funzione di derivazione della chiave per ottenere una chiave univoca. Sconsigliamo l'utilizzo di questa suite per generare testo cifrato, ma il AWS Encryption SDK lo supporta per motivi di compatibilità.

Per ulteriori informazioni su come queste suite sono rappresentate e utilizzate nella libreria, vedi [the section called “Riferimenti agli algoritmi”](#).

# Utilizzo dell'AWS Encryption SDK con AWS KMS

Per utilizzarlo AWS Encryption SDK, è necessario configurare i [portachiavi](#) o i [fornitori di chiavi master](#) con chiavi di confezionamento. Se non un'infrastruttura di chiavi non è disponibile, consigliamo di utilizzare [AWS Key Management Service \(AWS KMS\)](#). Molti degli esempi di codice in esso contenuti AWS Encryption SDK richiedono un [AWS KMS key](#).

Per interagire AWS KMS, è AWS Encryption SDK necessario l'AWS SDK del linguaggio di programmazione preferito. La libreria AWS Encryption SDK client funziona con gli AWS SDK per supportare le chiavi master archiviate in AWS KMS.

Per prepararsi a utilizzare AWS Encryption SDK con AWS KMS

1. Creazione di un Account AWS Per istruzioni, consultare [Come creare e attivare un nuovo account Amazon Web Services?](#) nel AWS Knowledge Center.
2. Creazione di una crittografia simmetrica AWS KMS key. Per assistenza, consulta [Creating Keys](#) nella Guida per gli AWS Key Management Service sviluppatori.

## Tip

Per usare la AWS KMS key in modo programmatico, è necessario l'ID chiave o l'Amazon Resource Name (ARN) della AWS KMS key. Per assistenza nell'individuazione dell'ID o dell'ARN di un utente AWS KMS key, consulta [Finding the Key ID e ARN](#) nella Guida per gli AWS Key Management Service sviluppatori.

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

Per creare un utente IAM con una chiave di accesso, vedi [Creazione di utenti IAM](#) nella Guida per l'utente IAM.

Per generare credenziali di sicurezza temporanee, vedi [Richiesta di credenziali di sicurezza temporanee](#) nella Guida per l'utente IAM.

4. Imposta AWS le tue credenziali utilizzando le istruzioni contenute in [AWS SDK for Java](#), [AWS SDK for JavaScript](#), [AWS SDK for Python \(Boto\)](#) o [AWS SDK for C++](#) (per C) e l'ID della chiave di accesso e la chiave di accesso segreta che hai generato nel passaggio 3. Se hai generato credenziali temporanee, dovrai anche specificare il token di sessione.

Questa procedura consente al SDK AWS di firmare le richieste ad AWS per te. I codici di esempio in AWS Encryption SDK che interagiscono con AWS KMS presuppongono che tu abbia completato questa fase.

5. Scarica e installa AWS Encryption SDK. Per scoprire come, consulta le istruzioni di installazione per il [linguaggio di programmazione](#) che desideri utilizzare.

# Best practice per AWS Encryption SDK

La AWS Encryption SDK è progettato per rendere ancora più semplice la protezione dei dati utilizzando best practice e standard di settore. Sebbene molte best practice siano selezionate per te nei valori predefiniti, alcune pratiche sono facoltative ma consigliate ogni volta che è pratico.

## Usa la versione più recente

Quando avvii AWS Encryption SDK, usa l'ultima versione offerta nel tuo preferito [Linguaggio](#). Se hai utilizzato il AWS Encryption SDK, esegui l'aggiornamento a ogni versione più recente il prima possibile. Questo ti assicura di utilizzare la configurazione consigliata e di sfruttare le nuove proprietà di sicurezza per proteggere i tuoi dati. Per i dettagli sulle versioni supportate, comprese le linee guida per la migrazione e la distribuzione, vedere [Support e manutenzione](#) e [Versioni del AWS Encryption SDK](#).

Se una nuova versione depreca gli elementi nel tuo codice, sostituiscili il prima possibile. Gli avvisi di deprecazione e i commenti sul codice in genere consigliano una buona alternativa.

Per rendere gli aggiornamenti significativi più facili e meno soggetti a errori, di tanto in tanto forniamo una versione temporanea o di transizione. Usa queste release e la relativa documentazione di accompagnamento per assicurarti di poter aggiornare l'applicazione senza interrompere il flusso di lavoro di produzione.

## Usa i valori predefiniti

La AWS Encryption SDK progetta le best practice nei suoi valori predefiniti. Quando possibile, usali. Per i casi in cui l'impostazione predefinita non è pratica, forniamo alternative, come suite di algoritmi senza firma. Offriamo anche opportunità di personalizzazione agli utenti avanzati, come portachiavi personalizzati, fornitori di chiavi master e gestori di materiali crittografici (CMM). Usa queste alternative avanzate con cautela e fai verificare le tue scelte da un tecnico della sicurezza ogni volta che è possibile.

## Utilizzo di un contesto di crittografia

Per migliorare la sicurezza delle operazioni di crittografia, includi [Contesto di crittografia](#) `dataKeyContext` un valore significativo in tutte le richieste di crittografia dei dati. L'utilizzo di un contesto di crittografia è facoltativo, ma viene consigliato come best practice. Un contesto di crittografia fornisce dati aggiuntivi autenticati (AAD) per la crittografia autenticata AWS Encryption SDK. Sebbene non sia un segreto, il contesto di crittografia può aiutarti [proteggere l'integrità e l'autenticità](#) dei tuoi dati crittografati.

NellaAWS Encryption SDK, si specifica un contesto di crittografia solo durante la crittografia. Durante la decrittografia, ilAWS Encryption SDKutilizza il contesto di crittografia nell'intestazione del messaggio crittografatoAWS Encryption SDKrestituisce. Prima che l'applicazione restituisca i dati di testo normale, verifica che il contesto di crittografia utilizzato per crittografare il messaggio sia incluso nel contesto di crittografia utilizzato per decrittare il messaggio. Per ulteriori informazioni, consulta gli esempi nel tuo linguaggio di programmazione.

Quando utilizzi l'interfaccia a riga di comando,AWS Encryption SDKverifica il contesto di crittografia per te.

## Proteggi i tuoi incarti

LaAWS Encryption SDKgenera una chiave di dati univoca per crittografare ogni messaggio di testo normale. Quindi crittografa la chiave dati con le chiavi di wrapping fornite. Se le chiavi di avvolgimento vengono perse o eliminate, i dati crittografati non sono recuperabili. Se le tue chiavi non sono protette, i tuoi dati potrebbero essere vulnerabili.

Utilizzare chiavi di wrapping protette da un'infrastruttura di chiavi sicura, ad esempio[AWS Key Management Service](#)(AWS KMS). Quando utilizzi chiavi AES o RSA raw, usa una fonte di casualità e storage durevole che soddisfa i tuoi requisiti di sicurezza. Generazione e archiviazione delle chiavi di wrapping in un modulo di sicurezza hardware (HSM) o in un servizio che fornisce HSM, ad esempioAWS CloudHSM, è una best practice.

Usa i meccanismi di autorizzazione della tua infrastruttura di chiavi per limitare l'accesso alle tue chiavi di wrapping solo agli utenti che lo richiedono. Implementare i principi delle best practice, come il privilegio minimo. Quando utilizziAWS KMS keys, utilizza le policy delle chiavi e le policy IAM[Principi di best practice](#).

## Specifica le tue chiavi di avvolgimento

È sempre una best practice[specifica le tue chiavi di avvolgimento](#)esplicitamente durante la decrittografia e la crittografia. Quando lo fai, ilAWS Encryption SDKusa solo le chiavi specificate. Questa pratica ti assicura di utilizzare solo le chiavi di crittografia che intendi. PerAWS KMSavvolgendo le chiavi, migliora anche le prestazioni impedendo di utilizzare inavvertitamente chiavi in un altroAccount AWS o Regione, o il tentativo di decrittografare con chiavi che non hai il permesso di utilizzare.

Durante la crittografia, i portachiavi e i fornitori di chiavi master cheAWS Encryption SDKi materiali di consumo richiedono di specificare le chiavi a capo. Usano tutte e solo le chiavi di avvolgimento specificate. È inoltre necessario specificare le chiavi di wrapping durante la crittografia e la decrittografia con portachiavi AES grezzi, portachiavi RSA grezzi e JCEMasterKeys.

Tuttavia, durante la decrittografia con AWS KMS portachiavi e fornitori di chiavi master, non è necessario specificare le chiavi di avvolgimento. La AWS Encryption SDK può ottenere l'identificatore della chiave dai metadati della chiave di dati crittografata. Tuttavia, la specifica delle chiavi è una best practice che ti consigliamo.

Per supportare questa best practice quando si lavora con AWS KMS Wrapping delle chiavi, ti consigliamo quanto segue:

- Utilizza AWS KMS portachiavi che specificano le chiavi di avvolgimento. Durante la crittografia e la decrittografia, questi portachiavi utilizzano solo le chiavi di wrapping specificate dall'utente.
- Quando utilizzi AWS KMS master keys e master key provider, utilizzano i costruttori in modalità rigorosa introdotti in [versione 1.7.x](#) della AWS Encryption SDK. Creano provider che crittografano e decrittografano solo con le chiavi di wrapping specificate. I costruttori per i fornitori di chiavi master che decrittografano sempre con qualsiasi chiave di wrapping sono deprecati nella versione 1.7.x e cancellato nella versione 2.0.x.

Quando si specifica AWS KMS il wrapping delle chiavi per la decrittografia non è pratico, è possibile utilizzare i provider di scoperta. La AWS Encryption SDK in C e JavaScript supporta [AWS KMS Keyring](#). I fornitori di chiavi master con una modalità di rilevamento sono disponibili per Java e Python nelle versioni 1.7.x e versioni successive. Questi provider di rilevamento, che vengono utilizzati solo per la decrittografia con AWS KMS avvolgendo le chiavi, dirigere esplicitamente il AWS Encryption SDK per utilizzare qualsiasi chiave di wrapping che crittografia una chiave dati.

Se è necessario utilizzare un provider di rilevamento, utilizzare il relativo filtro funzionalità per limitare le chiavi di avvolgimento che utilizzano. Ad esempio, le ricette [AWS KMS Keyring](#) usa solo le chiavi di avvolgimento in un particolare Regione AWS. Puoi inoltre configurare AWS KMS Keyring AWS KMS [Provider di chiavi](#) usare solo il [Wrapping](#) in particolare Account AWS. Inoltre, come sempre, utilizza le policy delle chiavi e le policy IAM per controllare l'accesso AWS KMS chiavi di avvolgimento.

## Usa le firme digitali

È consigliabile utilizzare una suite di algoritmi con la firma. [Firme digitali](#) verificare che il mittente del messaggio sia stato autorizzato a inviare il messaggio e proteggere l'integrità del messaggio. Tutte le versioni AWS Encryption SDK usa suite di algoritmi con la firma per impostazione predefinita.

Se i tuoi requisiti di sicurezza non includono le firme digitali, puoi selezionare una suite di algoritmi senza firme digitali. Tuttavia, si consiglia di utilizzare le firme digitali, soprattutto quando un gruppo di utenti crittografa i dati e un gruppo diverso di utenti decrittografa tali dati.

## Scadenza impegno

È consigliabile utilizzare la funzione di sicurezza dell'impegno chiave. Verificando l'identità dell'[Chiave](#) che ha crittografato i tuoi dati, [Scadenza impegno](#) impedisce di decrittografare qualsiasi testo cifrato che potrebbe generare più di un messaggio in chiaro.

La [AWS Encryption SDK](#) fornisce supporto completo per la crittografia e la decrittografia con impegno chiave a partire da [versione 2.0.x](#). Per impostazione predefinita, tutti i tuoi messaggi sono crittografati e decrittografati con impegno chiave. [Versione 1.7.x](#) del [AWS Encryption SDK](#) può decrittografare testi cifrati con impegno chiave. È progettato per aiutare gli utenti delle versioni precedenti a distribuire la versione 2.0.x con successo.

Supporto per l'impegno chiave include [nuove suite di algoritmi](#) e un [nuovo formato](#) che produce un testo cifrato di soli 30 byte più grande di un testo cifrato senza impegno chiave. Il design riduce al minimo il suo impatto sulle prestazioni in modo che la maggior parte degli utenti possa godere dei vantaggi dell'impegno chiave. Se la tua applicazione è molto sensibile alle dimensioni e alle prestazioni, potresti decidere di utilizzare il [Scadenza impegno](#) impostazione per disabilitare l'impegno chiave o consentire il [AWS Encryption SDK](#) per decrittografare i messaggi senza impegno, ma farlo solo se è necessario.

## Limita il numero di chiavi di dati crittografati

È una best practice per [limitare il numero di chiavi di dati crittografati](#) nei messaggi decrittografati, in particolare nei messaggi provenienti da fonti non attendibili. La decrittografia di un messaggio con numerose chiavi dati crittografate che non è possibile decrittografare può causare ritardi prolungati, aumentare le spese, limitare l'applicazione e altri che condividono il tuo account e potenzialmente esaurire l'infrastruttura delle chiavi. Senza limiti, un messaggio crittografato può avere fino a 65.535 ( $2^{16} - 1$ ) chiavi dati crittografate. Per dettagli, consulta [Limitazione delle chiavi di dati crittografate](#).

Per ulteriori informazioni su [AWS Encryption SDK](#) e le caratteristiche di sicurezza che sono alla base di queste best practice, consulta [Crittografia lato client Esplicit KeyIds Scadenza impegno](#) nella [AWS Blog](#).

# Configurazione della AWS Encryption SDK

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

Quando configuri l'implementazione, esamina le AWS Encryption SDK [migliori pratiche](#) e implementane quante più possibile.

## Argomenti

- [Selezione di un linguaggio di programmazione](#)
- [Selezione dei tasti di avvolgimento](#)
- [Utilizzo di più regioni AWS KMS keys](#)
- [Scelta di una suite di algoritmi](#)
- [Limitazione delle chiavi di dati crittografate](#)
- [Creazione di un filtro di scoperta](#)
- [Impostazione di una politica di impegno](#)
- [Lavorare con dati in streaming](#)
- [Memorizzazione nella cache delle chiavi dati](#)

## Selezione di un linguaggio di programmazione

AWS Encryption SDK È disponibile in più [linguaggi di programmazione](#). Le implementazioni del linguaggio sono progettate per essere completamente interoperabili e per offrire le stesse funzionalità, sebbene possano essere implementate in modi diversi. In genere, si utilizza la libreria compatibile con l'applicazione. Tuttavia, è possibile selezionare un linguaggio di programmazione per una particolare implementazione. Ad esempio, se preferisci lavorare con i [portachiavi](#), puoi scegliere il SDK di crittografia AWS per C o il SDK di crittografia AWS per JavaScript.

## Selezione dei tasti di avvolgimento

AWS Encryption SDK Genera una chiave dati simmetrica unica per crittografare ogni messaggio. A meno che non si utilizzi la [memorizzazione nella cache delle chiavi dati](#), non è necessario configurare, gestire o utilizzare le chiavi dati. Lo AWS Encryption SDK fa per te.

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

Per specificare le chiavi di wrapping per la crittografia e la decrittografia, si utilizza un portachiavi (C e JavaScript) o un provider di chiavi master (Java, Python, Encryption CLI). AWS È possibile specificare una chiave di wrapping o più chiavi di wrapping dello stesso tipo o di tipi diversi. Se utilizzi più chiavi di wrapping per racchiudere una chiave dati, ogni chiave di wrapping crittograferà una copia della stessa chiave dati. Le chiavi dati crittografate (una per chiave di wrapping) vengono archiviate con i dati crittografati nel messaggio crittografato che restituiscono. AWS Encryption SDK Per decrittografare i dati, è AWS Encryption SDK necessario innanzitutto utilizzare una delle chiavi di wrapping per decrittografare una chiave dati crittografata.

Per specificare una chiave AWS KMS key in un portachiavi o un provider di chiavi master, utilizza un identificatore di chiave supportato. AWS KMS Per i dettagli sugli identificatori di chiave per una AWS KMS chiave, consulta [Identificatori chiave](#) nella Guida per gli sviluppatori. AWS Key Management Service

- Quando si esegue la crittografia con SDK di crittografia AWS per Java, SDK di crittografia AWS per JavaScript SDK di crittografia AWS per Python, o l'AWS Encryption CLI, è possibile utilizzare qualsiasi identificatore di chiave valido (ID chiave, ARN chiave, nome alias o alias ARN) per una chiave KMS. Quando si esegue la crittografia con SDK di crittografia AWS per C, è possibile utilizzare solo un ID chiave o un ARN di chiave.

Se si specifica un nome alias o un alias ARN per una chiave KMS durante la crittografia, salva AWS Encryption SDK la chiave ARN attualmente associata a quell'alias; non salva l'alias. Le modifiche all'alias non influiscono sulla chiave KMS utilizzata per decrittografare le chiavi dati.

- Quando si esegue la decrittografia in modalità rigorosa (in cui si specificano chiavi di wrapping particolari), è necessario utilizzare una chiave ARN per l'identificazione. AWS KMS keys Questo requisito si applica a tutte le implementazioni di linguaggio di AWS Encryption SDK.

Quando si esegue la crittografia con un AWS KMS portachiavi, AWS Encryption SDK memorizza l'ARN della chiave AWS KMS key nei metadati della chiave dati crittografata. Durante la decrittografia in modalità rigorosa, AWS Encryption SDK verifica che la stessa chiave ARN sia presente nel portachiavi (o nel provider della chiave principale) prima di tentare di utilizzare la

chiave di wrapping per decrittografare la chiave dati crittografata. Se si utilizza un identificatore di chiave diverso, non lo riconoscerà né AWS Encryption SDK utilizzerà, anche se gli identificatori si riferiscono alla AWS KMS key stessa chiave.

Per specificare una [chiave AES non elaborata](#) o una [coppia di chiavi RSA non elaborata](#) come chiave di wrapping in un portachiavi, è necessario specificare uno spazio dei nomi e un nome. In un provider di chiavi master, `Provider ID` è l'equivalente dello spazio dei nomi e il `Key ID` è l'equivalente del nome. Durante la decrittografia, è necessario utilizzare lo stesso identico spazio dei nomi e lo stesso nome per ogni chiave di wrapping non elaborata utilizzata durante la crittografia. Se utilizzate un namespace o un nome diverso, non riconosceranno né AWS Encryption SDK utilizzeranno la chiave di wrapping, anche se il materiale della chiave è lo stesso.

## Utilizzo di più regioni AWS KMS keys

È possibile utilizzare i tasti multiregionali AWS Key Management Service (AWS KMS) come chiavi di avvolgimento in AWS Encryption SDK. Se si esegue la crittografia con una chiave multiregionale in una Regione AWS, è possibile decrittografare utilizzando una chiave multiregionale correlata in un'altra Regione AWS. Il supporto per le chiavi multiregione è stato introdotto nella versione 2.3. x della AWS Encryption SDK e versione 3.0. x della CLI di AWS crittografia.

AWS KMS Le chiavi multiregionali sono un insieme di AWS KMS keys chiavi diverse Regioni AWS che hanno lo stesso materiale chiave e lo stesso ID di chiave. È possibile utilizzare queste chiavi correlate come se fossero la stessa chiave in regioni diverse. Le chiavi multiregionali supportano scenari di disaster recovery e backup comuni che richiedono la crittografia in una regione e la decrittografia in un'altra regione senza effettuare una chiamata interregionale a AWS KMS. Per informazioni sulle chiavi multiregionali, consulta [Using Multiregion Keys](#) nella [Developer Guide](#). AWS Key Management Service

Per supportare le chiavi multiregionali, sono AWS Encryption SDK inclusi portachiavi compatibili con AWS KMS più regioni e fornitori di chiavi principali. Il nuovo simbolo multiregionale presente in ogni linguaggio di programmazione supporta sia chiavi a regione singola che a più regioni.

- Per le chiavi a regione singola, il simbolo che supporta più regioni si comporta esattamente come il portachiavi a regione singola e il provider di chiavi master. AWS KMS tenta di decrittografare il testo cifrato solo con la chiave a regione singola che ha crittografato i dati.

- Per le chiavi multiregionali, il simbolo che supporta più regioni tenta di decrittografare il testo cifrato con la stessa chiave multiregionale che ha crittografato i dati o con la relativa chiave multiregionale nella regione specificata.

Nei portachiavi e nei provider di chiavi master compatibili con più regioni che utilizzano più di una chiave KMS, puoi specificare più chiavi singole e multiregionali. Tuttavia, è possibile specificare solo una chiave per ogni set di chiavi multiregionali correlate. Se specificate più di un identificatore di chiave con lo stesso ID chiave, la chiamata al costruttore ha esito negativo.

È inoltre possibile utilizzare una chiave multiregionale con i portachiavi standard a regione singola e i provider di AWS KMS chiavi principali. Tuttavia, è necessario utilizzare la stessa chiave multiregionale nella stessa regione per crittografare e decrittografare. I portachiavi a regione singola e i provider di chiavi master tentano di decrittografare il testo cifrato solo con le chiavi che hanno crittografato i dati.

Gli esempi seguenti mostrano come crittografare e decrittografare i dati utilizzando chiavi multiregionali e i nuovi portachiavi e fornitori di chiavi master compatibili con più regioni. Questi esempi crittografano i dati nella `us-east-1` regione e decrittografano i dati nella regione utilizzando chiavi multiregionali correlate in ciascuna regione. `us-west-2` Prima di eseguire questi esempi, sostituisci la chiave multiregionale di esempio ARN con un valore valido tratto dal tuo Account AWS

## C

Per crittografare con una chiave multiregionale, utilizzate il `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` metodo per creare un'istanza del portachiavi. Specificate una chiave multiregionale.

Questo semplice esempio non include un [contesto di crittografia](#). Per un esempio che utilizza un contesto di crittografia in C, vedi [Crittografia e decrittazione di stringhe](#).

Per un esempio completo, consulta [kms\\_multi\\_region\\_keys.cpp](#) nel SDK di crittografia AWS per C repository su 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

Per eseguire la crittografia con una chiave multiregionale nella regione Stati Uniti orientali (Virginia settentrionale) (us-east-1), create un'istanza di un `CreateAwsKmsMrkKeyringInput` oggetto con un identificatore di chiave per la chiave multiregionale e un client per la regione specificata. AWS KMS Quindi utilizzate il metodo per creare il portachiavi. `CreateAwsKmsMrkKeyring()`

Il `CreateAwsKmsMrkKeyring()` metodo crea un portachiavi con esattamente una chiave multiregionale. Per crittografare con più chiavi di avvolgimento, inclusa una chiave multiregionale, utilizzate il metodo. `CreateAwsKmsMrkMultiKeyring()`

Per un esempio completo, consulta [AwsKmsMrkKeyringExample.cs](#) nel repository for.NET AWS Encryption SDK su. 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

Questo esempio crittografa il `hello.txt` file con una chiave multiregionale nella regione `us-east-1`. Poiché l'esempio specifica un ARN chiave con un elemento `Region`, questo esempio non utilizza l'attributo `region` del `--wrapping-keys` parametro.

Quando l'ID di chiave della chiave di wrapping non specifica una regione, è possibile utilizzare l'attributo `region` di `--wrapping-keys` per specificare la regione, ad esempio. `--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

Per crittografare con una chiave multiregionale, crea un'istanza `AwsKmsMrkAwareMasterKeyProvider` e specifica una chiave multiregionale.

Per un esempio completo, vedi [BasicMultiRegionKeyEncryptionExample.java](#) nel repository su SDK di crittografia AWS per Java 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

Per crittografare con una chiave multiregionale, utilizzate il `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` metodo per creare il portachiavi e specificare una chiave multiregionale.

Per un esempio completo, vedi [kms\\_multi\\_region\\_simple.ts](#) nel repository su SDK di crittografia AWS per JavaScript GitHub

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

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

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

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

/* Instantiate an AWS KMS client
 * The SDK di crittografia AWS per 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

Per crittografare con una chiave multiregionale, utilizzate il metodo per creare il portachiavi e specificare una chiave multiregionale. `buildAwsKmsMrkAwareStrictMultiKeyringNode()`

Per un esempio completo, vedi [kms\\_multi\\_region\\_simple.ts](#) nel repository su SDK di crittografia AWS per JavaScript GitHub

```

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

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

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

```

```

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

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

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

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

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

```

## Python

Per crittografare con una chiave multiregionale, utilizzate il metodo e specificate una AWS KMS chiave multiregionale. `MRKAwareStrictAwsKmsMasterKeyProvider()`

Per un esempio completo, consulta [mrk\\_aware\\_kms\\_provider.py nel repository](#) su SDK di crittografia AWS per Python 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
)
```

Quindi, sposta il testo cifrato nella regione. us-west-2 Non è necessario crittografare nuovamente il testo cifrato.

Per decrittografare il testo cifrato in modalità rigorosa nella us-west-2 regione, istanziate il simbolo multiregionale con la chiave ARN della chiave multiregione correlata nella regione. us-west-2 Se si specifica la chiave ARN di una chiave multiregionale correlata in una regione diversa (inclusa la zona in cui è stata crittografata)us-east-1, il simbolo Multi-Region-Aware effettuerà una chiamata interregionale a tale scopo. AWS KMS key

Durante la decrittografia in modalità rigorosa, il simbolo che supporta più regioni richiede una chiave ARN. Accetta solo una chiave ARN da ogni set di chiavi multiregione correlate.

Prima di eseguire questi esempi, sostituisci la chiave multiregionale di esempio ARN con un valore valido tratto dal tuo. Account AWS

## C

Per decrittografare in modalità rigorosa con una chiave multiregionale, utilizzate il `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` metodo per creare un'istanza del portachiavi. Specificate la chiave multiregionale correlata nella regione locale (us-west-2).

Per un esempio completo, consulta [kms\\_multi\\_region\\_keys.cpp nel repository](#) su. SDK di crittografia AWS per C 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

Per decrittografare in modalità rigorosa con un'unica chiave multiregionale, utilizzate gli stessi costruttori e metodi utilizzati per assemblare l'input e creare il portachiavi per la crittografia. Crea un'istanza di un `CreateAwsKmsMrkKeyringInput` oggetto con la chiave ARN di una chiave multiregionale correlata e un AWS KMS client per la regione Stati Uniti occidentali (Oregon) (`us-west-2`). Quindi utilizza il `CreateAwsKmsMrkKeyring()` metodo per creare un portachiavi multiregione con una chiave KMS multiregionale.

Per un esempio completo, consulta [AwsKmsMrkKeyringExample.cs nel repository for.NET](#) su AWS Encryption SDK 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

Per decrittografare con la relativa chiave multiregionale nella regione us-west-2, utilizzate l'attributo key del --wrapping-keys parametro per specificare l'ARN della chiave.

```
# 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

Per decrittografare in modalità rigorosa, crea un'istanza `AwsKmsMrkAwareMasterKeyProvider` e specifica la chiave multiregionale correlata nella regione locale (`us-west-2`).

[Per un esempio completo, consulta `.java` nel repository su.](#)

[BasicMultiRegionKeyEncryptionExample](#) SDK di crittografia AWS per Java 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

Per decrittografare in modalità rigorosa, utilizzate il `buildAwsKmsMrkAwareStrictMultiKeyringBrowser()` metodo per creare il portachiavi e specificare la chiave multiregionale correlata nella regione locale (us-west-2).

[Per un esempio completo, vedi `kms\_multi\_region\_simple.ts` nel repository su SDK di crittografia AWS per JavaScript GitHub](#)

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

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

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

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

/* Instantiate an AWS KMS client
 * The SDK di crittografia AWS per 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

Per decrittografare in modalità rigorosa, utilizzate il `buildAwsKmsMrkAwareStrictMultiKeyringNode()` metodo per creare il portachiavi e specificare la relativa chiave multiregionale nella regione locale (`us-west-2`).

[Per un esempio completo, vedi `kms\_multi\_region\_simple.ts` nel repository su SDK di crittografia AWS per JavaScript GitHub](#)

```

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

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

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

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

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

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

```

## Python

Per decrittografare in modalità rigorosa, utilizzate il metodo per creare il provider della chiave principale. `MRKAwareStrictAwsKmsMasterKeyProvider()` Specificare la chiave multiregionale correlata nella regione locale (us-west-2).

Per un esempio completo, consulta [mrk\\_aware\\_kms\\_provider.py nel repository](#) su. SDK di crittografia AWS per Python 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
)
```

È inoltre possibile decrittografare in modalità di rilevamento con AWS KMS chiavi multiregionali. Durante la decrittografia in modalità di rilevamento, non ne viene specificata alcuna. AWS KMS keys (Per informazioni sui portachiavi Single-Region AWS KMS Discovery, consulta.) [Utilizzo di un keyring di individuazione AWS KMS](#)

Se la crittografia è stata eseguita con una chiave multiregionale, il simbolo che supporta più regioni in modalità di individuazione tenterà di decrittografare utilizzando una chiave multiregionale correlata nella regione locale. Se non ne esiste nessuna, la chiamata ha esito negativo. In modalità di

rilevamento, non AWS Encryption SDK tenterà di effettuare una chiamata interregionale per la chiave multiregionale utilizzata per la crittografia.

### Note

Se si utilizza un simbolo che supporta più regioni in modalità di rilevamento per crittografare i dati, l'operazione di crittografia non riesce.

L'esempio seguente mostra come decrittografare con il simbolo che supporta più regioni in modalità di scoperta. Poiché non si specifica un AWS KMS key, AWS Encryption SDK deve ottenere la regione da una fonte diversa. Quando possibile, specifica la regione locale in modo esplicito. Altrimenti, AWS Encryption SDK ottiene la regione locale dalla regione configurata nell'AWSSDK per il tuo linguaggio di programmazione.

Prima di eseguire questi esempi, sostituisci l'ID account di esempio e la chiave multiregionale ARN con valori validi dal tuo Account AWS

## C

Per decrittografare in modalità di rilevamento con una chiave multiregionale, utilizzate il `Aws::Cryptosdk::KmsMrkAwareSymmetricKeyring::Builder()` metodo per creare il portachiavi e il metodo per creare il `Aws::Cryptosdk::KmsKeyring::DiscoveryFilter::Builder()` filtro di rilevamento. Per specificare la regione locale, definisci una `ClientConfiguration` e specificala nel client. AWS KMS

Per un esempio completo, consulta [kms\\_multi\\_region\\_keys.cpp](#) nel SDK di crittografia AWS per C repository su 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

Per creare un portachiavi di rilevamento compatibile con più regioni in AWS Encryption SDK for .NET, crea un'istanza di un `CreateAwsKmsMrkDiscoveryKeyringInput` oggetto che utilizza un AWS KMS client per un determinato client e un filtro di rilevamento opzionale che limiti le chiavi KMS a una partizione e a un account particolari Regione AWS. AWS Quindi chiamate il metodo con l'oggetto di input. `CreateAwsKmsMrkDiscoveryKeyring()` Per un esempio

completo, consulta [AwsKmsMrkDiscoveryKeyringExample.cs](#) nel repository AWS Encryption SDK for.NET su. GitHub

Per creare un portachiavi Discovery compatibile con più regioni per più di un portachiavi Regione AWS, usa il `CreateAwsKmsMrkDiscoveryMultiKeyring()` metodo per creare un portachiavi multi-regione oppure utilizzalo per creare diversi portachiavi Discovery con riconoscimento multiregionale e `CreateAwsKmsMrkDiscoveryKeyring()` poi usa il metodo per combinarli in un portachiavi multiplo. `CreateMultiKeyring()`

Per [AwsKmsMrkDiscoveryMultiKeyringExampleun](#) esempio, consulta `.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

Per decrittografare in modalità di rilevamento, utilizzate l'attributo `discovery` del parametro. `--wrapping-keys` Gli attributi `discovery-account` e `discovery-partition` creano un filtro di scoperta facoltativo, ma consigliato.

Per specificare la regione, questo comando include l'attributo `region` del parametro. `--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

Per specificare la regione locale, utilizzate il `builder().withDiscoveryMrkRegion` parametro. Altrimenti, AWS Encryption SDK ottiene la regione locale dalla regione configurata in [AWS SDK for Java](#).

Per un esempio completo, consulta [DiscoveryMultiRegionDecryptionExample.java](#) nel SDK di crittografia AWS per Java repository su. 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

Per decrittografare in modalità di rilevamento con una chiave simmetrica multiregionale, utilizzate il metodo `AwsKmsMrkAwareSymmetricDiscoveryKeyringBrowser()`

Per un esempio completo, vedi [kms\\_multi\\_region\\_discovery.ts](#) nel repository su SDK di crittografia AWS per JavaScript GitHub

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

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

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

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

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

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

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

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

## JavaScript Node.js

Per decrittografare in modalità di scoperta con una chiave multiregionale simmetrica, utilizzate il metodo `AwsKmsMrkAwareSymmetricDiscoveryKeyringNode()`

Per un esempio completo, vedi [kms\\_multi\\_region\\_discovery.ts](#) nel repository su SDK di crittografia AWS per JavaScript GitHub

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

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

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

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

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

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

```

        discoveryFilter,
    })

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

```

## Python

Per decrittografare in modalità di scoperta con una chiave multiregionale, usa il metodo.

`MRKAwareDiscoveryAwsKmsMasterKeyProvider()`

Per un esempio completo, consulta [mrk\\_aware\\_kms\\_provider.py nel repository](#) su SDK di crittografia AWS per Python 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
)

```

## Scelta di una suite di algoritmi

AWS Encryption SDKSupporta diversi [algoritmi di crittografia simmetrica e asimmetrica](#) per crittografare le chiavi di dati con le chiavi di wrapping specificate. [Tuttavia, quando utilizza tali chiavi](#)

[dati per crittografare i dati, l'AWS Encryption SDK impostazione predefinita è una suite di algoritmi consigliata che utilizza l'algoritmo AES-GCM con derivazione delle chiavi, firme digitali e impegno delle chiavi.](#) Sebbene la suite di algoritmi predefinita sia probabilmente adatta alla maggior parte delle applicazioni, è possibile scegliere una suite di algoritmi alternativa. Ad esempio, alcuni modelli di fiducia sarebbero soddisfatti da una suite di algoritmi senza [firme digitali](#). Per informazioni sulle suite di algoritmi AWS Encryption SDK supportate, vedere [Suite di algoritmi supportate in AWS Encryption SDK](#).

Gli esempi seguenti mostrano come selezionare una suite di algoritmi alternativa durante la crittografia. Questi esempi selezionano una suite di algoritmi AES-GCM consigliata con derivazione delle chiavi e impegno delle chiavi, ma senza firme digitali. Quando esegui la crittografia con una suite di algoritmi che non include firme digitali, utilizza la modalità di decrittografia solo senza segno durante la decrittografia. Questa modalità, che fallisce se incontra un testo cifrato firmato, è particolarmente utile durante la decrittografia in streaming.

## C

Per specificare una suite di algoritmi alternativa in, è necessario creare una CMM in modo SDK di crittografia AWS per C esplicito. Quindi utilizzate il `aws_cryptosdk_default_cmm_set_alg_id` con la CMM e la suite di algoritmi selezionata.

```
/* 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;
}

```

Per decrittografare dati crittografati senza firme digitali, usa.

**AWS\_CRYPTOSDK\_DECRYPT\_UNSIGNED** Ciò causa l'esito negativo della decrittografia se viene rilevato testo cifrato firmato.

```

/* 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

Per specificare una suite di algoritmi alternativa in per.NET, specificate la AWS Encryption SDK proprietà di un oggetto. `AlgorithmSuiteId` [EncryptInput](#) La proprietà AWS Encryption SDK for .NET include [costanti](#) che è possibile utilizzare per identificare la suite di algoritmi preferita.

The AWS Encryption SDK for .NET non dispone di un metodo per rilevare il testo cifrato firmato durante la decrittografia in streaming perché questa libreria non supporta lo streaming di dati.

```

// 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

Durante la crittografia del `hello.txt` file, questo esempio utilizza il `--algorithm` parametro per specificare una suite di algoritmi senza firme digitali.

```

# 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

```

Durante la decrittografia, questo esempio utilizza il parametro. `--decrypt-unsigned` Questo parametro è consigliato per garantire la decrittografia di testo cifrato non firmato, in particolare con la CLI, che trasmette sempre input e output.

```

# 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

Per specificare una suite di algoritmi alternativa, utilizzate il metodo.

`AwsCrypto.builder().withEncryptionAlgorithm()` Questo esempio specifica una suite di algoritmi alternativa senza firme digitali.

```

// 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();

```

Quando trasmettete dati in streaming per la decrittografia, utilizzate il `createUnsignedMessageDecryptingStream()` metodo per assicurarvi che tutto il testo cifrato che state decrittando non sia firmato.

```
// 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

Per specificare una suite di algoritmi alternativa, utilizzate il parametro con un valore enum.  
**suiteId** 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, })
```

Durante la decrittografia, utilizzate il metodo standard. `decrypt` SDK di crittografia AWS per JavaScript nel browser non è disponibile una `decrypt-unsigned` modalità perché il browser non supporta lo 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

Per specificare una suite di algoritmi alternativa, utilizzate il `suiteId` parametro con un valore `AlgorithmSuiteIdentifier` enum.

```
// Specify an algorithm suite without signing

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

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

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

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

Per decrittografare dati crittografati senza firme digitali, utilizzate Stream.

`decryptUnsignedMessage` Questo metodo fallisce se incontra testo cifrato firmato.

```
// 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

Per specificare un algoritmo di crittografia alternativo, utilizzate il `algorithm` parametro con un valore 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
)
```

Quando decifrate messaggi crittografati senza firme digitali, utilizzate la modalità `decrypt-unsigned streaming`, specialmente quando decifrate durante lo 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
```

## Limitazione delle chiavi di dati crittografate

È possibile limitare il numero di chiavi dati crittografate in un messaggio crittografato. Questa funzionalità basata sulle best practice può aiutarti a rilevare un portachiavi configurato in modo errato durante la crittografia o un testo cifrato dannoso durante la decrittografia. Inoltre, evita chiamate inutili, costose e potenzialmente esaustive all'infrastruttura chiave. La limitazione delle chiavi di dati crittografate è particolarmente utile quando si decifrano messaggi da una fonte non attendibile.

Sebbene la maggior parte dei messaggi crittografati disponga di una chiave dati crittografata per ogni chiave di wrapping utilizzata nella crittografia, un messaggio crittografato può contenere fino a 65.535 chiavi dati crittografate. Un malintenzionato potrebbe creare un messaggio crittografato con migliaia di chiavi di dati crittografate, nessuna delle quali può essere decrittografata. Di conseguenza, AWS Encryption SDK tenterebbe di decrittografare ogni chiave di dati crittografata fino a esaurire le chiavi di dati crittografate contenute nel messaggio.

Per limitare le chiavi dati crittografate, utilizzate il parametro `MaxEncryptedDataKeys`. Questo parametro è disponibile per tutti i linguaggi di programmazione supportati a partire dalla versione 1.9. x e 2.2. x del AWS Encryption SDK. È facoltativo e valido per la crittografia e la decrittografia. Gli esempi seguenti decrittografano i dati crittografati con tre diverse chiavi di wrapping. Il `MaxEncryptedDataKeys` valore è impostato su 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

Per limitare le chiavi di dati crittografate in AWS Encryption SDK for .NET, crea un'istanza di un client AWS Encryption SDK per .NET e imposta il relativo `MaxEncryptedDataKeys` parametro opzionale sul valore desiderato. Quindi, chiama il `Decrypt()` metodo sull'istanza AWS Encryption SDK configurata.

```
// 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)
```

## Creazione di un filtro di scoperta

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

la chiave KMS che l'ha crittografata, indipendentemente da chi possiede o ha accesso a quella chiave KMS.

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

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

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

Gli esempi in questa sezione mostrano come creare un filtro di scoperta. Prima di utilizzare il codice, sostituite i valori di esempio con valori validi per la partizione Account AWS and.

## C

Per un esempio completo, vedere [kms\\_discovery.cpp](#) in. SDK di crittografia AWS per 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

Per un esempio completo, vedere [DiscoveryFilterExample.cs](#) in the AWS Encryption SDK for.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

Per un esempio completo, [DiscoveryDecryptionExamplevedi.java](#) in. SDK di crittografia AWS per Java

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

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

## JavaScript (Node and Browser)

Per esempi completi, vedere [kms\\_filtered\\_discovery.ts \(Node.js\)](#) e [kms\\_multi\\_region\\_discovery.ts \(Browser\)](#) in. SDK di crittografia AWS per JavaScript

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

```
}
```

## Python

[Per un esempio completo, vedere `discovery\_kms\_provider.py` in `SDK di crittografia AWS per 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",
)
```

## Impostazione di una politica di impegno

[Una policy di impegno è un'impostazione di configurazione che determina se l'applicazione esegue la crittografia e la decrittografia con un impegno chiave. La crittografia e la decrittografia con impegno chiave è una procedura consigliata. AWS Encryption SDK](#)

L'impostazione e la modifica della politica di impegno è un passaggio fondamentale nella [migrazione](#) dalle versioni 1.7. x e versioni precedenti della versione AWS Encryption SDK 2.0. x e versioni successive. Questa progressione è spiegata in dettaglio nell'[argomento sulla migrazione](#).

Il valore predefinito della politica di impegno nelle versioni più recenti di AWS Encryption SDK (a partire dalla versione 2.0). x) `RequireEncryptRequireDecrypt`, è ideale per la maggior parte delle situazioni. Tuttavia, se è necessario decrittografare un testo cifrato che è stato crittografato senza l'impegno di una chiave, potrebbe essere necessario modificare la politica di impegno in `RequireEncryptAllowDecrypt`. Per esempi su come impostare una politica di impegno in ogni linguaggio di programmazione, vedi. [Impostazione della politica di impegno](#)

## Lavorare con dati in streaming

Quando trasmetti dati in streaming per la decrittografia, tieni presente che AWS Encryption SDK restituisce testo in chiaro decrittografato dopo il completamento dei controlli di integrità, ma prima della verifica della firma digitale. Per evitare che venga restituito o utilizzato testo non crittografato fino alla verifica della firma, consigliamo di memorizzare nel buffer il testo non crittografato trasmesso in streaming fino al completamento dell'intero processo di decrittografia.

Questo problema si verifica solo quando si esegue lo streaming di testo cifrato per la decrittografia e solo quando si utilizza una suite di algoritmi, come la suite di algoritmi predefinita, che include firme digitali.

Per semplificare il buffering, alcune implementazioni del AWS Encryption SDK linguaggio, ad esempio SDK di crittografia AWS per JavaScript in Node.js, includono una funzionalità di buffering come parte del metodo di decrittografia. La AWS Encryption CLI, che trasmette sempre input e output, ha introdotto un `--buffer` parametro nelle versioni 1.9. x e 2.2. x. In altre implementazioni linguistiche, è possibile utilizzare le funzionalità di buffering esistenti. (The AWS Encryption SDK for .NET non supporta lo streaming).

Se utilizzi una suite di algoritmi senza firme digitali, assicurati di utilizzare la `decrypt-unsigned` funzionalità in ogni implementazione linguistica. Questa funzionalità decrittografa il testo cifrato ma fallisce se incontra testo cifrato firmato. Per informazioni dettagliate, consultare [Scelta di una suite di algoritmi](#).

## Memorizzazione nella cache delle chiavi dati

In generale, il riutilizzo delle chiavi dati è sconsigliato, ma AWS Encryption SDK offre un'opzione di memorizzazione nella [cache delle chiavi dati](#) che consente un riutilizzo limitato delle chiavi dati. La memorizzazione nella cache delle chiavi di dati può migliorare le prestazioni di alcune applicazioni e ridurre le chiamate all'infrastruttura chiave. Prima di utilizzare la memorizzazione nella cache delle chiavi di dati in produzione, regolate le [soglie di sicurezza](#) e verificate che i vantaggi superino gli svantaggi del riutilizzo delle chiavi di dati.

# Utilizzo dei keyring

ISDK di crittografia AWS per C, iSDK di crittografia AWS per JavaScript, i e i SDK di crittografia AWS per Java per.NET utilizzano i portachiavi AWS Encryption SDK per eseguire la crittografia delle [buste](#). I keyring generano, crittografano e decrittano le chiavi di dati. I portachiavi determinano l'origine delle chiavi dati univoche che proteggono ogni messaggio e delle chiavi di [avvolgimento che crittografano tale chiave](#) di dati. Puoi specificare un keyring durante la crittografia e lo stesso keyring o uno diverso durante la decrittazione. Puoi utilizzare i keyring forniti dall'SDK oppure scrivere keyring personalizzati compatibili.

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

Ti consigliamo di utilizzare un portachiavi che protegga le tue chiavi di wrapping ed esegua operazioni crittografiche entro un limite sicuro, come il AWS KMS portachiavi, che utilizza that never leave () unencrypted. AWS KMS keys [AWS Key Management Service](#) AWS KMS Puoi anche scrivere un portachiavi che utilizzi chiavi di wrapping archiviate nei tuoi moduli di sicurezza hardware (HSM) o protette da altri servizi di chiavi master. Per informazioni dettagliate, consulta l'argomento [Interfaccia di keyring](#) nella Specifica di AWS Encryption SDK.

I portachiavi svolgono il ruolo di [chiavi master e fornitori di chiavi master](#) nella SDK di crittografia AWS per Java e nella SDK di crittografia AWS per Python CLI di AWS crittografia. Se utilizzi implementazioni linguistiche diverse per crittografare e AWS Encryption SDK decrittografare i dati, assicurati di utilizzare portachiavi e fornitori di chiavi master compatibili. Per informazioni dettagliate, consultare [Compatibilità dei keyring](#).

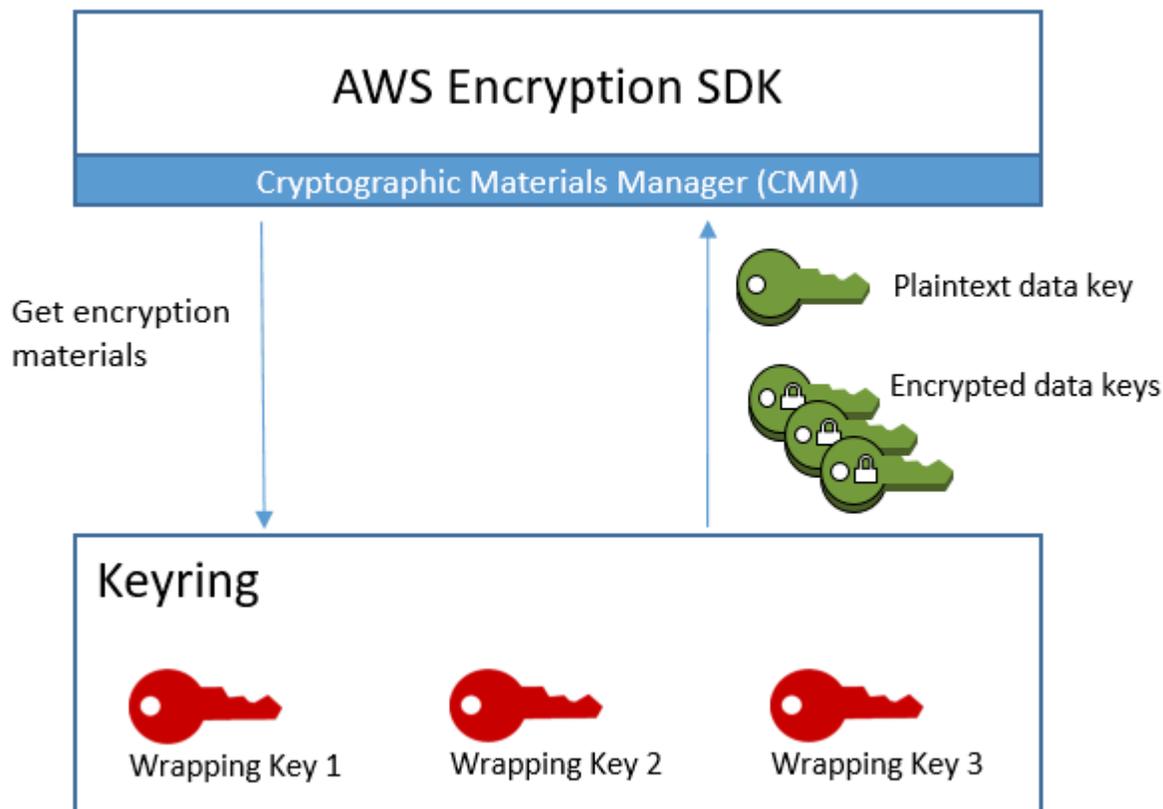
In questo argomento viene illustrato come utilizzare la caratteristica keyring di AWS Encryption SDK e come scegliere un keyring. [Per esempi di creazione e utilizzo dei portachiavi, consulta C e gli argomenti. JavaScript](#)

## Argomenti

- [Come funzionano i keyring](#)
- [Compatibilità dei keyring](#)
- [Scegliere un portachiavi](#)

## Come funzionano i keyring

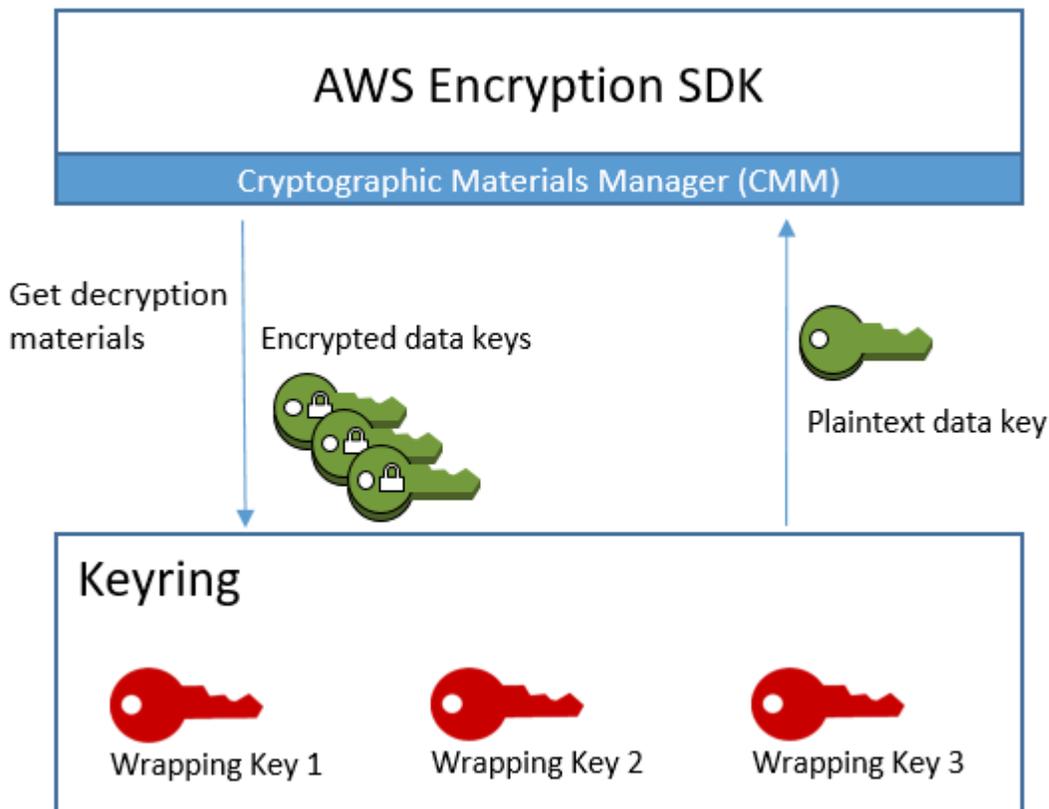
Quando esegui la crittografia dei dati, il AWS Encryption SDK chiede al keyring i materiali di crittografia. Il portachiavi restituisce una chiave dati in testo semplice e una copia della chiave dati crittografata da ciascuna delle chiavi di avvolgimento del portachiavi. AWS Encryption SDK utilizza la chiave di testo semplice per crittografare i dati, quindi distrugge la chiave di dati in testo semplice. Quindi, AWS Encryption SDK restituisce un [messaggio crittografato che include le chiavi di dati crittografate](#) e i dati crittografati.



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

AWS Encryption SDK Passa le chiavi di dati crittografate dal messaggio crittografato al portachiavi e chiede al portachiavi di decrittografarle tutte. Il keyring utilizza le chiavi di wrapping per decrittare una delle chiavi di dati crittografate e restituisce una chiave di dati di testo normale. AWS Encryption SDK utilizza la chiave di dati di testo normale per decrittare i dati. Se nessuna delle chiavi di wrapping

nel keyring è in grado di decrittare una qualsiasi delle chiavi di dati crittografate, l'operazione di decrittazione non riesce.



Puoi utilizzare un singolo keyring o combinarne più di uno dello stesso tipo o di tipi diversi in un [keyring multiplo](#). Quando esegui la crittografia dei dati, il keyring multiplo restituisce una copia della chiave di dati crittografata da tutte le chiavi di wrapping di tutti i keyring che costituiscono il keyring multiplo. È possibile decrittografare i dati utilizzando un portachiavi con una qualsiasi delle chiavi di avvolgimento del portachiavi multiplo.

## Compatibilità dei keyring

Sebbene le diverse implementazioni linguistiche di presentino alcune differenze architettoniche, AWS Encryption SDK sono completamente compatibili, soggette a vincoli linguistici. È possibile crittografare i dati utilizzando un'implementazione linguistica e decrittografarli con qualsiasi altra implementazione linguistica. Tuttavia, è necessario utilizzare la stessa chiave di wrapping o quella corrispondente per crittografare e decrittografare le chiavi dati. Per informazioni sui vincoli del linguaggio, consulta l'argomento relativo all'implementazione di ciascun linguaggio, ad esempio [the section called "Compatibilità"](#) nell'argomento SDK di crittografia AWS per JavaScript.

## Requisiti diversi per i portachiavi di crittografia

Nelle implementazioni AWS Encryption SDK linguistiche diverse dalla SDK di crittografia AWS per C, tutte le chiavi di inserimento in un portachiavi di crittografia (o portachiavi multiplo) o in un provider di chiavi master devono essere in grado di crittografare la chiave dati. Se una chiave di wrapping non riesce a crittografare, il metodo di crittografia fallisce. Di conseguenza, il chiamante deve disporre delle [autorizzazioni necessarie](#) per tutte le chiavi del portachiavi. Se si utilizza un portachiavi Discovery per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia non riesce.

L'eccezione è la SDK di crittografia AWS per C, in cui l'operazione di crittografia ignora un portachiavi di rilevamento standard, ma ha esito negativo se si specifica un portachiavi di rilevamento multiarea, da solo o in un portachiavi multiregionale.

## Keyring e fornitori di chiavi master compatibili

La tabella seguente mostra quali chiavi master e provider di chiavi master sono compatibili con i portachiavi forniti. AWS Encryption SDK Qualsiasi incompatibilità minore dovuta a vincoli linguistici è spiegata nell'argomento relativo all'implementazione della lingua.

Portachiavi:	Fornitore di chiavi principali:
<a href="#">AWS KMSportachiavi</a>	<a href="#">KMS MasterKey (Java)</a> <a href="#">KMS MasterKeyProvider (Java)</a> <a href="#">KMS MasterKey (Python)</a> <a href="#">KMS MasterKeyProvider (Python)</a>
	<div style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; background-color: #E6F2FF;"> <p> <b>Note</b></p> <p>I SDK di crittografia AWS per Python e SDK di crittografia AWS per Java non includono una chiave master o un provider di chiavi master equivalente al portachiavi <a href="#">Discovery AWS KMS regionale</a>.</p> </div>

Portachiavi:	Fornitore di chiavi principali:
<a href="#">AWS KMSPortachiavi gerarchico</a>	Disponibile solo con la versione 4. x di AWS Encryption SDK per .NET e versione 3. x delSDK di crittografia AWS per Java.
<a href="#">Keyring non elaborato AES</a>	Se utilizzati con chiavi di crittografia simmetriche: <a href="#">JceMasterKey</a> (Java)  <a href="#">RawMasterKey</a> (Python)
<a href="#">Keyring non elaborato RSA</a>	Se utilizzati con chiavi di crittografia asimmetriche: <a href="#">JceMasterKey</a> (Java)  <a href="#">RawMasterKey</a> (Python)

 **Note**

Il portachiavi Raw RSA non supporta chiavi KMS asimmetriche. Se desideri utilizzare chiavi RSA KMS asimmetriche, versione 4. x of the AWS Encryption SDK for .NET supporta i AWS KMS portachiavi che utilizzano la crittografia simmetrica () o l'RSA asimmetrica. SYMMETRIC\_DEFAULT AWS KMS keys

## Scegliere un portachiavi

Il portachiavi determina le chiavi di avvolgimento che proteggono i dati, le chiavi e, in ultima analisi, i dati. Utilizzate le chiavi di avvolgimento più sicure e pratiche per il vostro compito. Se possibile, utilizzate chiavi di wrapping protette da un modulo di sicurezza hardware o da un'infrastruttura di gestione delle chiavi, come le chiavi KMS in [AWS Key Management Service](#)(AWS KMS) o le chiavi di crittografia. [AWS CloudHSM](#)

AWS Encryption SDKFornisce diversi portachiavi e configurazioni di portachiavi in più linguaggi di programmazione ed è possibile creare portachiavi personalizzati. È inoltre possibile creare un [portachiavi multiplo](#) che includa uno o più portachiavi dello stesso tipo o di un tipo diverso.

### Argomenti

- [Portachiavi AWS KMS](#)
- [AWS KMS Portachiavi gerarchici](#)
- [Keyring non elaborati AES](#)
- [Keyring non elaborato RSA](#)
- [Keyring multipli](#)

## Portachiavi AWS KMS

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

[È possibile utilizzare una chiave AWS KMS multiregionale in un portachiavi o in un provider di AWS KMS chiavi master a partire dalla versione 2.3. x della AWS Encryption SDK e versione 3.0. x della CLI di AWS crittografia.](#) Per dettagli ed esempi sull'utilizzo del nuovo simbolo multiregionale, vedere. [Utilizzo di più regioni AWS KMS keys](#) Per informazioni sulle chiavi multiregionali, consulta [Uso delle chiavi multiregionali nella Guida](#) per gli sviluppatori. AWS Key Management Service

### Note

Versione 4. x di AWS Encryption SDK per .NET e versione 3. x di queste SDK di crittografia AWS per Java sono le uniche implementazioni del linguaggio di programmazione che supportano AWS KMS portachiavi che utilizzano RSA asimmetrico. AWS KMS keys Se si tenta di includere una chiave KMS asimmetrica in un portachiavi di crittografia in qualsiasi altra implementazione linguistica, la chiamata encrypt ha esito negativo. Se la includi in un portachiavi di decrittografia, viene ignorata. Tutte le menzioni relative ai portachiavi KMS nel riquadro si riferiscono ai portachiavi. AWS Encryption SDK AWS KMS

AWS KMSi portachiavi possono includere due tipi di chiavi avvolgenti:

- Chiave generatrice: genera una chiave di dati in testo semplice e la crittografa. Un portachiavi che crittografa i dati deve avere una chiave generatrice.
- Chiavi aggiuntive: crittografa la chiave di dati in testo semplice generata dalla chiave del generatore. AWS KMSi portachiavi possono avere zero o più chiavi aggiuntive.

Durante la crittografia, il AWS KMS portachiavi utilizzato deve avere una chiave generatrice. Durante la decrittografia, la chiave del generatore è facoltativa e la distinzione tra le chiavi dei generatori e le chiavi aggiuntive viene ignorata.

Quando un portachiavi di AWS KMS crittografia ha una sola AWS KMS chiave, tale chiave viene utilizzata per generare e crittografare la chiave dati.

Come tutti i portachiavi, i AWS KMS portachiavi possono essere utilizzati indipendentemente o in un [portachiavi multiplo con altri portachiavi](#) dello stesso tipo o di un tipo diverso.

## Argomenti

- [AWS KMS Autorizzazioni richieste per i portachiavi](#)
- [Identificazione di AWS KMS keys in un keyring AWS KMS](#)
- [Creazione di un portachiavi per la crittografia AWS KMS](#)
- [Creazione di un AWS KMS portachiavi per la decrittografia](#)
- [Utilizzo di un keyring di individuazione AWS KMS](#)
- [Utilizzo di un keyring di individuazione regionale AWS KMS](#)

## AWS KMS Autorizzazioni richieste per i portachiavi

AWS Encryption SDK Non richiede un Account AWS e non dipende da nessuno. Servizio AWS Tuttavia, per utilizzare un AWS KMS portachiavi, sono necessarie le seguenti autorizzazioni Account AWS minime sul AWS KMS keys portachiavi.

- Per crittografare con un AWS KMS portachiavi, è necessaria l'autorizzazione [kms:GenerateDataKey](#) sulla chiave del generatore. È necessaria l'autorizzazione [KMS:Encrypt](#) su tutte le chiavi aggiuntive nel portachiavi. AWS KMS
- Per decriptare con un AWS KMS portachiavi, è necessaria l'autorizzazione [KMS:Decrypt](#) su almeno una chiave del portachiavi. AWS KMS
- [Per crittografare con un portachiavi multiplo composto da portachiavi, è necessaria l'autorizzazione kms: sulla AWS KMS chiave del generatore nel portachiavi del generatore. GenerateDataKey](#) È necessaria l'autorizzazione [KMS:Encrypt](#) su tutte le altre chiavi in tutti gli altri portachiavi. AWS KMS

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

## Identificazione di AWS KMS keys in un keyring AWS KMS

Un AWS KMS portachiavi può includerne uno o più. AWS KMS keys Per specificare un elemento AWS KMS key in un AWS KMS portachiavi, utilizzate un identificatore di AWS KMS chiave supportato. Gli identificatori di chiave che è possibile utilizzare per identificare un elemento AWS KMS key in un portachiavi variano a seconda dell'operazione e dell'implementazione del linguaggio. Per informazioni dettagliate sugli identificatori chiave di un AWS KMS key, consulta [Key Identifiers](#) nella Developer Guide. AWS Key Management Service

Come procedura consigliata, utilizzate l'identificatore di chiave più specifico e pratico per la vostra attività.

- In un portachiavi di crittografia per SDK di crittografia AWS per C, è possibile utilizzare una [chiave ARN o un alias ARN](#) per identificare le chiavi KMS. In tutte le altre implementazioni linguistiche, puoi utilizzare un [ID chiave](#), un [ARN di chiave](#), un nome alias o un [alias ARN](#) per crittografare i [dati](#).
- In un keyring di decrittazione devi utilizzare un ARN di chiave per identificare le AWS KMS keys. Questo requisito si applica a tutte le implementazioni di linguaggio di AWS Encryption SDK. Per informazioni dettagliate, consultare [Selezione dei tasti di avvolgimento](#).
- In un keyring utilizzato per la crittografia e la decrittazione devi utilizzare un ARN di chiave per identificare le AWS KMS keys. Questo requisito si applica a tutte le implementazioni di linguaggio di AWS Encryption SDK.

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

## Creazione di un portachiavi per la crittografia AWS KMS

È possibile configurare ogni AWS KMS portachiavi con uno AWS KMS key o più portachiavi nello stesso Account AWS. Regioni AWS AWS KMS keys Devono essere chiavi di crittografia simmetriche (SYMMETRIC\_DEFAULT). [È inoltre possibile utilizzare una chiave KMS multiregionale con crittografia simmetrica. Come con tutti i portachiavi, puoi utilizzare uno o più AWS KMS portachiavi in un portachiavi multiplo.](#)

Quando si crea un AWS KMS portachiavi per crittografare i dati, è necessario specificare una chiave generatrice, AWS KMS key che viene utilizzata per generare una chiave di dati in testo semplice e crittografarla. La chiave dati non è matematicamente correlata alla chiave KMS. Facoltativamente,

puoi anche specificare ulteriori AWS KMS keys per crittografare la stessa chiave di dati di testo normale.

Per decrittografare il messaggio crittografato protetto da questo portachiavi, il portachiavi utilizzato deve includere almeno uno dei messaggi AWS KMS keys definiti nel portachiavi, altrimenti no. AWS KMS keys ([Un AWS KMS portachiavi con no AWS KMS keys è noto come portachiavi Discovery.](#)) [AWS KMS](#)

Nelle implementazioni AWS Encryption SDK linguistiche diverse dalla SDK di crittografia AWS per C, tutte le chiavi di inserimento in un portachiavi di crittografia o in un portachiavi multiplo devono essere in grado di crittografare la chiave dati. Se una chiave di wrapping non riesce a crittografare, il metodo `encrypt` fallisce. Di conseguenza, il chiamante deve disporre delle [autorizzazioni necessarie](#) per tutte le chiavi del portachiavi. Se si utilizza un portachiavi Discovery per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia non riesce. L'eccezione è la SDK di crittografia AWS per C, in cui l'operazione di crittografia ignora un portachiavi di rilevamento standard, ma ha esito negativo se si specifica un portachiavi di rilevamento multiarea, da solo o in un portachiavi multiregionale.

Gli esempi seguenti creano un AWS KMS portachiavi con una chiave generatrice e una chiave aggiuntiva. Questi esempi utilizzano gli [ARN chiave](#) per identificare le chiavi KMS. Si tratta di una procedura ottimale per i AWS KMS portachiavi utilizzati per la crittografia e un requisito per i AWS KMS portachiavi utilizzati per la decrittografia. Per informazioni dettagliate, consultare [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

## C

Per identificare un elemento AWS KMS key in un portachiavi di crittografia in SDK di crittografia AWS per C, specificare una [chiave ARN](#) o un [alias ARN](#). In un keyring di decrittografia devi utilizzare un ARN di chiave. Per informazioni dettagliate, consultare [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

Per un esempio completo, vedi [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

Per creare un AWS KMS portachiavi con una o più AWS KMS chiavi in formato .NET, create un portachiavi multiplo. AWS Encryption SDK Il portachiavi AWS Encryption SDK per .NET include un portachiavi multiplo solo per le chiavi. AWS KMS

[Quando si specifica un AWS KMS key per un portachiavi di crittografia in per.NET, è possibile utilizzare qualsiasi identificatore di chiave valido: un ID chiave, un ARN di chiave, un nome alias o un alias ARN. AWS Encryption SDK](#) Per informazioni su come identificarlo in un portachiavi, consulta. AWS KMS keys AWS KMS [Identificazione di AWS KMS keys in un keyring AWS KMS](#)

L'esempio seguente utilizza la versione 4. x di the per.NET AWS Encryption SDK per creare un AWS KMS portachiavi con una chiave generatrice e chiavi aggiuntive. Per un esempio completo, vedi [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

[Quando si specifica un AWS KMS key per un portachiavi di crittografia in SDK di crittografia AWS per JavaScript, è possibile utilizzare qualsiasi identificatore di chiave valido: un ID di chiave, un ARN di chiave, un nome alias o un alias ARN.](#) Per informazioni sull'AWS KMS keys identificazione di un portachiavi, vedere. AWS KMS [Identificazione di AWS KMS keys in un keyring AWS KMS](#)

Per un esempio completo, consulta [kms\\_simple.ts](#) nel repository in. SDK di crittografia AWS per 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

[Quando si specifica un AWS KMS key per un portachiavi di crittografia in SDK di crittografia AWS per JavaScript, è possibile utilizzare qualsiasi identificatore di chiave valido: un ID di chiave, un ARN di chiave, un nome alias o un alias ARN.](#) Per informazioni sull'AWS KMS keys identificazione di un portachiavi, vedere. AWS KMS [Identificazione di AWS KMS keys in un keyring AWS KMS](#)

Per un esempio completo, consulta [kms\\_simple.ts](#) nel repository in. SDK di crittografia AWS per 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

Per creare un AWS KMS portachiavi con una o più chiavi in, crea un AWS KMS portachiavi multiplo. SDK di crittografia AWS per Java SDK di crittografia AWS per JavaInclude un portachiavi multiplo solo per le chiavi. AWS KMS

[Quando si specifica un AWS KMS key per un portachiavi di crittografia in SDK di crittografia AWS per Java, è possibile utilizzare qualsiasi identificatore di chiave valido: un ID di chiave, un ARN di](#)

[chiave, un nome alias o un alias ARN](#). Per informazioni sull'AWS KMS keys identificazione di un portachiavi, vedere [AWS KMS Identificazione di AWS KMS keys in un keyring AWS KMS](#)

Per un esempio completo, consulta [BasicEncryptionKeyringExample.java](#) nel SDK di crittografia AWS per Java repository in. 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);
```

## Creazione di un AWS KMS portachiavi per la decrittografia

[È inoltre necessario specificare un AWS KMS portachiavi quando si decrittografa il messaggio crittografato che restituisce](#). AWS Encryption SDK Se il portachiavi di decrittografia lo specifica AWS KMS keys, AWS Encryption SDK utilizzerà solo quelle chiavi di avvolgimento per decrittografare le chiavi di dati crittografate nel messaggio crittografato. Puoi anche utilizzare un [keyring di individuazione AWS KMS](#) che non specifica alcuna AWS KMS keys.

Durante la decrittografia, AWS Encryption SDK cerca nel portachiavi una persona in grado di decrittografare una delle AWS KMS chiavi di dati crittografate. AWS KMS key In particolare, AWS Encryption SDK utilizza il seguente modello per ogni chiave di dati crittografata in un messaggio crittografato.

- AWS Encryption SDK Ottiene la chiave ARN di chi AWS KMS key ha crittografato la chiave dati dai metadati del messaggio crittografato.

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

AWS Encryption SDK non prova mai a decrittare una chiave di dati crittografata, a meno che l'ARN della AWS KMS key che l'ha crittografata non sia inclusa nel keyring di decrittazione. Se il keyring di decrittazione non include l'ARN delle AWS KMS keys che hanno crittografato le chiavi di dati, AWS Encryption SDK non riesce a decrittare la chiamata senza coinvolgere AWS KMS.

[A partire dalla versione 1.7. x, quando si decrittografa una chiave di dati crittografata, passa AWS Encryption SDK sempre la chiave ARN del AWS KMS key al KeyId parametro dell'operazione Decrypt. AWS KMS](#) Identificare il AWS KMS key momento della decrittografia è una procedura AWS KMS consigliata che garantisce la decrittografia della chiave di dati crittografata con la chiave di wrapping che si intende utilizzare.

Una chiamata di decrittografia con un AWS KMS portachiavi ha esito positivo quando almeno uno dei componenti del portachiavi di decrittografia è AWS KMS key in grado di decrittografare una delle chiavi di dati crittografate nel messaggio crittografato. Inoltre, l'intermediario deve avere l'autorizzazione `kms:Decrypt` per la AWS KMS key. Questo comportamento consente di crittografare i dati relativi a più account Regioni AWS e AWS KMS keys in diversi account, ma fornisce un portachiavi di decrittografia più limitato e personalizzato per un particolare account, regione, utente, gruppo o ruolo.

Quando si specifica una chiave AWS KMS key in un portachiavi di decrittografia, è necessario utilizzare la relativa chiave ARN. In caso contrario, la AWS KMS key non viene riconosciuta. Per informazioni su come trovare l'ARN chiave, consulta [Finding the Key ID and ARN](#) nella Developer Guide. AWS Key Management Service

#### Note

Se un keyring di crittografia viene riutilizzato per la decrittazione assicurati che le AWS KMS keys nel keyring siano identificate dai relativi ARN di chiave.

Ad esempio, il seguente AWS KMS portachiavi include solo la chiave aggiuntiva utilizzata nel portachiavi di crittografia. Tuttavia, invece di fare riferimento alla chiave aggiuntiva tramite il relativo `alias/alias/exampleAlias`, l'esempio utilizza l'ARN della chiave aggiuntiva come richiesto dalle chiamate di decrittografia.

Puoi utilizzare questo keyring per decrittare un messaggio crittografato sia con la chiave generatore che con la chiave aggiuntiva, a condizione che hai l'autorizzazione per utilizzare la chiave aggiuntiva per decrittare i dati.

C

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

C# / .NET

Poiché questo portachiavi di decrittografia include solo una AWS KMS chiave, nell'esempio viene utilizzato il `CreateAwsKmsKeyring()` metodo con un'istanza del relativo oggetto. `CreateAwsKmsKeyringInput` Per creare un AWS KMS portachiavi con una sola AWS KMS chiave, è possibile utilizzare un portachiavi a chiave singola o multipla. Per informazioni dettagliate, consultare [Crittografia dei dati in per.NET AWS Encryption SDK](#). L'esempio seguente utilizza la versione 4. x di a per.NET AWS Encryption SDK per creare un AWS KMS portachiavi per la decrittografia.

```
// 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

Poiché questo portachiavi di decrittografia include solo una AWS KMS chiave, nell'esempio viene utilizzato il `CreateAwsKmsKeyring()` metodo con un'istanza del relativo oggetto. `CreateAwsKmsKeyringInput` Per creare un AWS KMS portachiavi con una sola AWS KMS chiave, è possibile utilizzare un portachiavi a chiave singola o multipla.

```
// 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);
```

È inoltre possibile utilizzare un AWS KMS portachiavi che specifica una chiave generatrice per la decrittografia, come la seguente. Quando esegui la decrittazione, l'AWS Encryption SDK ignora la differenza tra le chiavi del generatore e le chiavi aggiuntive. Può utilizzare una qualsiasi delle AWS KMS keys specificate per decrittare una chiave di dati crittografata. La chiamata a AWS KMS riesce solo quando l'intermediario ha l'autorizzazione per utilizzare la AWS KMS key per decrittare i dati.

C

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

C# / .NET

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .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);
```

A differenza di un keyring di crittografia che utilizza tutte le AWS KMS keys specificate, un messaggio crittografato può essere decrittato utilizzando un keyring con AWS KMS keys non correlate al messaggio crittografato e AWS KMS keys per le quali l'intermediario non dispone di autorizzazioni. Se una chiamata di decrittazione a AWS KMS non riesce, ad esempio quando l'intermediario non ha le autorizzazioni necessarie, AWS Encryption SDK passa semplicemente alla chiave di dati crittografata successiva.

## Utilizzo di un keyring di individuazione AWS KMS

[Durante la decrittografia, è consigliabile specificare le chiavi di wrapping che possono utilizzare.](#)

AWS Encryption SDK Per seguire questa procedura ottimale, utilizzate un portachiavi di AWS KMS decrittografia che limiti le chiavi di AWS KMS wrapping a quelle specificate. Tuttavia, puoi anche creare un portachiavi AWS KMS Discovery, ovvero un AWS KMS portachiavi che non specifichi alcuna chiave di avvolgimento.

AWS Encryption SDK Fornisce un portachiavi AWS KMS Discovery standard e un portachiavi Discovery per chiavi multiregionali. AWS KMS Per informazioni sull'utilizzo delle chiavi multiregionali con, vedere. AWS Encryption SDK [Utilizzo di più regioni AWS KMS keys](#)

Poiché non specifica alcuna chiave di wrapping, un portachiavi Discovery non può crittografare i dati. Se si utilizza un portachiavi Discovery per crittografare i dati, da solo o in un portachiavi multiplo, l'operazione di crittografia non riesce. L'eccezione è la SDK di crittografia AWS per C, in cui l'operazione di crittografia ignora un portachiavi di rilevamento standard, ma ha esito negativo se si specifica un portachiavi di rilevamento multiarea, da solo o in un portachiavi multiregionale.

Durante la decrittografia, un portachiavi Discovery consente di chiedere AWS KMS di AWS Encryption SDK decrittografare qualsiasi chiave di dati crittografata utilizzando la chiave che l'ha crittografata, indipendentemente da chi la possiede o ha accesso a AWS KMS key tale chiave. AWS KMS key La chiamata ha esito positivo solo quando il chiamante dispone dell'autorizzazione per. `kms:Decrypt AWS KMS key`

 Important

Se includi un portachiavi AWS KMS Discovery in un portachiavi [multiplo di decrittografia](#), [il portachiavi](#) Discovery ha la precedenza su tutte le restrizioni relative alle chiavi KMS specificate dagli altri portachiavi del portachiavi multiplo. Il portachiavi multiplo si comporta come il portachiavi meno restrittivo. Un portachiavi AWS KMS Discovery non ha alcun effetto sulla crittografia se utilizzato da solo o in un portachiavi multiplo.

Per praticità, AWS Encryption SDK fornisce un keyring di individuazione AWS KMS, ma, se possibile, consigliamo di utilizzare un keyring di portata più limitata per i motivi seguenti.

- Autenticità: un portachiavi AWS KMS Discovery può utilizzare qualsiasi chiave utilizzata per crittografare una chiave dati nel messaggio crittografato, solo in modo AWS KMS key che il chiamante abbia il permesso di utilizzarla per decrittografarla. AWS KMS key anche se potrebbe non essere la AWS KMS key che intende impiegare. Ad esempio, una delle chiavi di dati potrebbe essere stata crittografata con una AWS KMS key meno sicura rispetto allo standard richiesto.
- Latenza e prestazioni: un portachiavi AWS KMS Discovery potrebbe essere sensibilmente più lento rispetto ad altri portachiavi perché AWS Encryption SDK tenta di decrittografare tutte le chiavi dati crittografate, comprese quelle crittografate da AWS KMS keys altre regioni Account AWS e AWS KMS keys che il chiamante non è autorizzato a utilizzare per la decrittografia.

Se utilizzi un portachiavi di rilevamento, ti consigliamo di utilizzare un filtro di rilevamento per limitare le chiavi KMS che possono essere utilizzate a quelle contenute in partizioni e specifiche. Account AWS I filtri Discovery sono supportati nelle versioni 1.7. x e versioni successive di AWS Encryption SDK. Per informazioni su come trovare l'ID e la partizione dell'account, consulta I tuoi Account AWS identificatori e il formato ARN in. Riferimenti generali di AWS

Il codice seguente crea un'istanza di un portachiavi di AWS KMS rilevamento con un filtro di rilevamento che limita le chiavi KMS AWS Encryption SDK utilizzabili a quelle presenti nella partizione e nell'account di esempio 111122223333. aws

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

C

Per un esempio completo, vedi: [kms\\_discovery.cpp](#).

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

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

C# / .NET

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .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

In JavaScript, è necessario specificare in modo esplicito la proprietà `discovery`.

```

const clientProvider = getClient(KMS, { credentials })

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

```

## JavaScript Node.js

In JavaScript, è necessario specificare in modo esplicito la proprietà `discovery`.

```

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);
```

## Utilizzo di un keyring di individuazione regionale AWS KMS

Un portachiavi AWS KMS Regional Discovery è un portachiavi che non specifica gli ARN delle chiavi KMS. Invece, consente la decrittografia utilizzando solo AWS Encryption SDK le chiavi KMS in particolare. Regioni AWS

Quando si decrittografa con un portachiavi AWS KMS Regional Discovery, AWS Encryption SDK decripta qualsiasi chiave di dati crittografata che è stata crittografata con un codice specificato. AWS KMS key Regione AWS Per avere successo, il chiamante deve disporre dell'`kms : Decrypt` autorizzazione su almeno una delle chiavi di dati specificate Regione AWS che hanno AWS KMS keys crittografato una chiave dati.

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

### Important

Se includi un portachiavi di rilevamento AWS KMS regionale in un portachiavi multiplo di decrittografia, il portachiavi di rilevamento regionale ha la precedenza su tutte le restrizioni relative alle [chiavi KMS specificate dagli altri portachiavi del portachiavi multiplo](#). Il portachiavi multiplo si comporta come il portachiavi meno restrittivo. Un portachiavi AWS KMS Discovery non ha alcun effetto sulla crittografia se utilizzato da solo o in un portachiavi multiplo.

Il portachiavi Regional Discovery SDK di crittografia AWS per C tenta di decriptare solo con chiavi KMS nella regione specificata. Quando si utilizza un portachiavi di rilevamento in SDK di crittografia AWS per JavaScript e AWS Encryption SDK per .NET, si configura la regione sul client. AWS KMS Queste AWS Encryption SDK implementazioni non filtrano le chiavi KMS per regione, ma AWS KMS falliranno una richiesta di decrittografia delle chiavi KMS al di fuori della regione specificata.

Se utilizzi un portachiavi di rilevamento, ti consigliamo di utilizzare un filtro di rilevamento per limitare le chiavi KMS utilizzate nella decrittografia a quelle nelle partizioni e nelle partizioni specificate. Account AWS I filtri Discovery sono supportati nelle versioni 1.7. x e versioni successive di AWS Encryption SDK.

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

## C

Per esaminare un esempio che mostra questo keyring e il metodo `create_kms_client`, consulta [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

Il portachiavi AWS Encryption SDK per .NET non dispone di un portachiavi dedicato alla scoperta regionale. Tuttavia, è possibile utilizzare diverse tecniche per limitare le chiavi KMS utilizzate durante la decrittografia a una particolare regione.

Il modo più efficiente per limitare le regioni in un portachiavi di rilevamento consiste nell'utilizzare un portachiavi di rilevamento che supporta più regioni, anche se i dati sono stati crittografati utilizzando solo chiavi a regione singola. Quando incontra chiavi a regione singola, il portachiavi che supporta più regioni non utilizza alcuna funzionalità multiregionale.

Il portachiavi restituito dal metodo `filtra` le chiavi KMS per regione prima della chiamata `CreateAwsKmsMrkDiscoveryKeyring()`. AWS KMS Invia una richiesta di decrittografia AWS KMS solo quando la chiave di dati crittografata è stata crittografata da una chiave KMS nella regione specificata dal parametro nell'`Region` oggetto. `CreateAwsKmsMrkDiscoveryKeyringInput`

Gli esempi seguenti utilizzano la versione 4. x del AWS Encryption SDK per .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);
```

Puoi anche limitare le chiavi KMS a uno specifico Regione AWS specificando una regione nell'istanza del AWS KMS client () [AmazonKeyManagementServiceClient](#). Tuttavia, questa configurazione è meno efficiente e potenzialmente più costosa rispetto all'utilizzo di un portachiavi di rilevamento con riconoscimento multiregionale. Invece di filtrare le chiavi KMS AWS Encryption SDK per regione prima della chiamata AWS KMS, fo.NET chiama ogni chiave di dati crittografata (finché non ne decripta una) e si affida AWS KMS per limitare le chiavi KMS utilizzate AWS KMS alla regione specificata.

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .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

Per visualizzare questo keyring e la funzione `limitRegions` in un esempio di lavoro, vedi [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);

```

SDK di crittografia AWS per JavaScript esporta anche una funzione `excludeRegions` per Node.js e il browser. Questa funzione crea un keyring di individuazione regionale AWS KMS che omette le AWS KMS keys in determinate regioni. L'esempio seguente crea un portachiavi AWS KMS Regional Discovery che può essere utilizzato AWS KMS keys nell'account 111122223333 in tutti Regione AWS gli Stati Uniti orientali (Virginia settentrionale) (us-east-1).

Non SDK di crittografia AWS per C dispone di un metodo analogo, ma è possibile implementarne uno creandone uno personalizzato. [ClientSupplier](#)

Questo esempio mostra il codice per Node.js.

```

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

```

## AWS KMSPortachiavi gerarchici

### Important

Il portachiavi AWS KMS Hierarchical è supportato solo dalla versione 4. x di AWS Encryption SDK per .NET e versione 3. x del SDK di crittografia AWS per Java.

Con il portachiavi AWS KMS Hierarchical, puoi proteggere i tuoi materiali crittografici con una chiave KMS a crittografia simmetrica senza chiamare AWS KMS ogni volta che crittografi o decrittografi i

dati. È una buona scelta per le applicazioni che devono ridurre al minimo le chiamate e le applicazioni che possono riutilizzare alcuni materiali crittografici AWS KMS senza violare i requisiti di sicurezza.

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

Il portachiavi Hierarchical utilizza in genere ogni versione della chiave branch per soddisfare più richieste. Tuttavia, puoi controllare la misura in cui le chiavi di ramo attive vengono riutilizzate e determinare la frequenza con cui la chiave di ramo attiva viene ruotata. La versione attiva della chiave di ramo rimane attiva finché non viene [ruotata](#). Le versioni precedenti della chiave branch attiva non verranno utilizzate per eseguire operazioni di crittografia, ma potranno comunque essere interrogate e utilizzate nelle operazioni di decrittografia.

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

La cache locale è composta da due partizioni, una per le operazioni di crittografia e una seconda per le operazioni di decrittografia. La partizione di crittografia memorizza i materiali delle chiavi di branch assemblati dalla chiave branch attiva e li riutilizza per tutte le operazioni di crittografia

fino alla scadenza del limite della cache. La partizione di decrittografia memorizza i materiali delle chiavi di filiale assemblati per altre versioni di chiavi di filiale identificate nelle operazioni di decrittografia. La partizione di decrittografia può memorizzare più versioni di materiali chiave di filiale attivi contemporaneamente. Se è configurata per utilizzare un fornitore di ID di chiavi di filiale per un ambiente multitenant, la partizione encrypt può anche archiviare più versioni di materiali relativi alle chiavi di filiale contemporaneamente. Per ulteriori informazioni, consulta [Utilizzo del portachiavi Hierarchical in ambienti multitenant](#).

#### Note

Tutte le menzioni del portachiavi gerarchico si riferiscono al portachiavi gerarchico. AWS Encryption SDK AWS KMS

## Argomenti

- [Come funziona](#)
- [Prerequisiti](#)
- [Crea un portachiavi gerarchico](#)
- [Ruota la chiave branch attiva](#)
- [Utilizzo del portachiavi Hierarchical in ambienti multitenant](#)

## Come funziona

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

## Crittografia e firma

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

1. Il metodo di crittografia richiede al portachiavi Hierarchical i materiali di crittografia. Il portachiavi genera una chiave di dati in testo semplice, quindi verifica se nella cache locale sono presenti

- materiali branch validi per generare la chiave di wrapping. Se sono presenti materiali validi per le chiavi di filiale, il portachiavi passa alla Fase 5.
2. Se non ci sono materiali validi per le chiavi di filiale, il portachiavi Hierarchical interroga l'archivio delle chiavi di filiale per trovare la chiave di filiale attiva.
    - a. L'archivio delle chiavi di filiale chiama AWS KMS per decrittografare la chiave di ramo attiva e restituisce la chiave di ramo attiva in testo semplice. I dati che identificano la chiave di ramo attiva vengono serializzati per fornire dati autenticati aggiuntivi (AAD) nella chiamata di decrittografia a. AWS KMS
    - b. L'archivio delle chiavi di filiale restituisce la chiave di ramo in testo semplice e i dati che la identificano, ad esempio la versione della chiave di filiale.
  3. Il portachiavi Hierarchical assembla i materiali chiave del ramo (la chiave di ramo in testo semplice e la versione della chiave di ramo) e ne archivia una copia nella cache locale.
  4. Il portachiavi Hierarchical ricava una chiave di avvolgimento unica dalla chiave branch in testo semplice e un sale casuale a 16 byte. Utilizza la chiave di wrapping derivata per crittografare una copia della chiave di dati in testo non crittografato.

Il metodo di crittografia utilizza i materiali di crittografia per crittografare i dati. Per ulteriori informazioni, consulta [Come AWS Encryption SDK crittografia i dati](#).

## Decrittografa e verifica

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

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

Per ulteriori informazioni, consulta [AWS KMS Dettagli tecnici del portachiavi gerarchico](#).

3. Il portachiavi Hierarchical verifica se nella cache locale sono presenti materiali chiave di filiale validi che corrispondono alla versione della chiave di filiale identificata nel passaggio 2. Se sono presenti materiali validi per le chiavi di filiale, il portachiavi passa alla Fase 6.

4. Se non ci sono materiali validi per le chiavi di filiale, il portachiavi Hierarchical interroga l'archivio delle chiavi di filiale per trovare la chiave di filiale che corrisponde alla versione della chiave di filiale identificata nello Step 2.
  - a. L'archivio delle chiavi di filiale chiama AWS KMS per decrittografare la chiave di ramo e restituisce la chiave di ramo attiva in testo semplice. I dati che identificano la chiave di ramo attiva vengono serializzati per fornire dati autenticati aggiuntivi (AAD) nella chiamata di decrittografia a. AWS KMS
  - b. L'archivio delle chiavi di filiale restituisce la chiave di ramo in testo semplice e i dati che la identificano, ad esempio la versione della chiave di filiale.
5. Il portachiavi Hierarchical assembla i materiali chiave del ramo (la chiave di ramo in testo semplice e la versione della chiave di ramo) e ne archivia una copia nella cache locale.
6. Il portachiavi Hierarchical utilizza i materiali delle chiavi branch assemblate e il sale da 16 byte identificato nella fase 2 per riprodurre la chiave di avvolgimento univoca che crittografava la chiave dati.
7. Il portachiavi Hierarchical utilizza la chiave di wrapping riprodotta per decrittografare la chiave dati e restituisce la chiave dati in testo semplice.

Il metodo di decrittografia utilizza i materiali di decrittografia e la chiave di dati in testo semplice per decrittografare il messaggio crittografato. [Per ulteriori informazioni, consulta Come decipta un messaggio crittografato. AWS Encryption SDK](#)

## Prerequisiti

AWS Encryption SDK Non richiede un file Account AWS e non dipende da nessuno. Servizio AWS Tuttavia, il portachiavi gerarchico dipende da Amazon AWS KMS DynamoDB.

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

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

## Fase 1: Configurare un nuovo servizio di archiviazione delle chiavi

Il servizio di archiviazione delle chiavi fornisce diverse operazioni API, come `CreateKeyStore` e `CreateKey`, per aiutarti a assemblare i prerequisiti del portachiavi gerarchico e a gestire l'archivio delle chiavi della filiale.

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

Il nome dell'archivio di chiavi logiche è associato crittograficamente a tutti i dati memorizzati nella tabella per semplificare le operazioni di ripristino di DynamoDB. Il nome dell'archivio di chiavi logiche può essere lo stesso del nome della tabella DynamoDB, ma non è necessario. Si consiglia di specificare il nome della tabella DynamoDB come nome della tabella logica quando si configura per la prima volta il servizio di archiviazione delle chiavi. È necessario specificare sempre lo stesso nome di tabella logica. Nel caso in cui il nome dell'archivio chiavi della filiale cambi dopo il [ripristino della tabella DynamoDB da un backup](#), il nome dell'archivio di chiavi logico viene mappato al nome della tabella DynamoDB specificato per garantire che il portachiavi Hierarchical possa ancora accedere all'archivio delle chiavi della filiale.

### Note

Il nome dell'archivio logico delle chiavi è incluso nel contesto di crittografia di tutte le operazioni API del servizio di archiviazione chiavi che effettuano la chiamata. AWS KMS Il contesto di crittografia non è segreto, i suoi valori, incluso il nome dell'archivio di chiavi logiche, vengono visualizzati in testo semplice nei log. AWS CloudTrail

## C# / .NET

```
var kmsConfig = new KMSConfiguration { KmsKeyArn = kmsKeyArn };
var keystoreConfig = new KeyStoreConfig
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsConfiguration = kmsConfig,
    DdbTableName = keyStoreName,
    DdbClient = new AmazonDynamoDBClient(),
    LogicalKeyName = logicalKeyName
};
```

```
var keystore = new KeyStore(keystoreConfig);
```

## 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();
```

## Fase 2: Chiama per creare un archivio di chiavi in filiale **CreateKeyStore**

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

### C# / .NET

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

## Java

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

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

	Chiave di partizione	Chiave di ordinamento
Tabella di base	branch-key-id	type

### Note

È possibile creare manualmente la tabella DynamoDB che funge da archivio delle chiavi della filiale anziché utilizzare l'operazione. CreateKeyStore Se scegli di creare

manualmente l'archivio delle chiavi di filiale, devi specificare i seguenti valori di stringa per le chiavi di partizione e ordinamento:

- Chiave di partizione: `branch-key-id`
- Chiave di ordinamento: `type`

### Passaggio 3: Chiama **CreateKey** per creare una nuova chiave di ramo attiva

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

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

- **Identificatore chiave di ramo:** definisce una chiave personalizzata `branch-key-id`.

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

- [Contesto di crittografia: definisce un set opzionale di coppie chiave-valore non segrete che forniscono dati autenticati aggiuntivi \(AAD\) nel contesto di crittografia incluso nella chiamata `kms: GenerateDataKeyWithoutPlaintext`](#)

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

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();
```

Innanzitutto, l'CreateKeyoperazione genera i seguenti valori.

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

Quindi, l'CreateKeyoperazione chiama [kms: GenerateDataKeyWithoutPlaintext](#) utilizzando la seguente richiesta.

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

Successivamente, l'CreateKeyoperazione chiama [kms: ReEncrypt](#) per creare un record attivo per la chiave branch aggiornando il contesto di crittografia.

Infine, l'CreateKeyoperazione chiama [ddb: TransactWriteItems](#) per scrivere un nuovo elemento che mantenga la chiave di ramo nella tabella creata nel passaggio 2. L'elemento ha i seguenti attributi.

```
{
  "branch-key-id" : branch-key-id,
  "type" : "branch:ACTIVE",
  "enc" : the branch key returned by the GenerateDataKeyWithoutPlaintext call,
```

```

    "version": "branch:version:the branch key version UUID",
    "create-time" : "timestamp",
    "kms-arn" : "the KMS key ARN you specified in Step 1",
    "hierarchy-version" : "1",
    "aws-crypto-ec:contextKey": "contextValue"
  }

```

## Crea un portachiavi gerarchico

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

- Il nome di un archivio di chiavi della filiale

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

- 

Un limite di durata della cache (TTL)

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

- Un identificatore di chiave di ramo

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

### Note

Per inizializzare il portachiavi Hierarchical per l'uso multitenant, è necessario specificare un fornitore di ID di chiavi di filiale anziché un `branch-key-id`. Per ulteriori informazioni, consulta [Utilizzo del portachiavi Hierarchical in ambienti multitenant](#).

- (Facoltativo) Una cache

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

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

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

### Default (Recommended)

Per la maggior parte degli utenti, la cache predefinita soddisfa i requisiti di threading. La cache predefinita è progettata per supportare ambienti con molti multithread. Quando scade l'immissione di materiali relativi a una chiave di filiale, la cache predefinita impedisce la chiamata a più thread AWS KMS e Amazon DynamoDB notificando a un thread che l'immissione dei materiali chiave del ramo sta per scadere con 10 secondi di anticipo. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache. AWS KMS

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

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

### C# /.NET

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

### Java

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

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

### MultiThreaded

La MultiThreaded cache è sicura da usare in ambienti multithread, ma non fornisce alcuna funzionalità per ridurre al minimo AWS KMS le chiamate Amazon DynamoDB. Di conseguenza,

quando scade l'immissione di materiali chiave in una filiale, tutti i thread verranno avvisati contemporaneamente. Ciò può comportare più AWS KMS chiamate per aggiornare la cache.

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

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

### 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

La StormTracking cache è progettata per supportare ambienti con molti multithread. Quando una voce relativa ai materiali delle chiavi di filiale scade, la StormTracking cache impedisce a più thread di richiamare AWS KMS Amazon DynamoDB notificando in anticipo a un thread che l'immissione dei materiali chiave del ramo sta per scadere. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache. AWS KMS

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

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

Valore predefinito: 1 voce

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

Valore predefinito: 10 secondi

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

Valore predefinito: 1 secondo

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

Valore predefinito: 20 tentativi

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

Valore predefinito: 20 secondi

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

Valore predefinito: 20 millisecondi

## 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())

```

- (Facoltativo) Un elenco di token di concessione

Se controlli l'accesso alla chiave KMS nel tuo portachiavi gerarchico con le [concessioni, devi fornire tutti i](#) token di concessione necessari quando iniziizzi il portachiavi.

L'esempio seguente inizializza un portachiavi Hierarchical con un TLL limite di cache di 600 secondi e una capacità di ingresso di 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);
```

## Ruota la chiave branch attiva

Può esserci una sola versione attiva per ogni chiave di ramo alla volta. Il portachiavi Hierarchical utilizza in genere ogni versione di chiave branch attiva per soddisfare più richieste. Ma sei tu a controllare la misura in cui le chiavi branch attive vengono riutilizzate e a determinare la frequenza con cui la chiave branch attiva viene ruotata.

Le chiavi branch non vengono utilizzate per crittografare le chiavi di dati in testo semplice. Vengono utilizzate per derivare le chiavi di wrapping univoche che crittografano le chiavi di dati in testo non crittografato. Il [processo di derivazione della chiave di wrapping](#) produce una chiave di wrapping unica da 32 byte con 28 byte di casualità. Ciò significa che una chiave branch può derivare più di 79 ottillioni, o 2<sup>96</sup>,

chiavi di wrapping uniche prima che si verifichi l'usura crittografica. Nonostante questo rischio di esaurimento molto basso, potrebbe essere necessario ruotare le chiavi di filiale attive a causa di norme aziendali o contrattuali o normative governative.

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

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

C# / .NET

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

Java

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

## Utilizzo del portachiavi Hierarchical in ambienti multitenant

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

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

Prima di poter inizializzare il portachiavi Hierarchical per l'uso multitenant, è necessario creare una chiave di filiale per ogni tenant e creare un fornitore di ID di chiavi di filiale. Utilizza il fornitore di ID della chiave di filiale per creare un nome descrittivo per facilitare il riconoscimento di `branch-key-ids` quello corretto per un tenant. `branch-key-id` Ad esempio, il nome descrittivo consente di fare riferimento a una chiave di filiale come `tenant1` invece `dib3f61619-4d35-48ad-a275-050f87e15122`.

Per le operazioni di decrittografia, è possibile configurare staticamente un singolo portachiavi gerarchico per limitare la decrittografia a un singolo tenant, oppure è possibile utilizzare il fornitore

di ID della chiave di filiale per identificare quale tenant è responsabile della decrittografia di un messaggio.

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

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

Chiama `CreateKey` per ogni inquilino.

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

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());
```

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

L'esempio seguente crea un fornitore di ID chiave di filiale.

C# / .NET

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

Java

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

### Passaggio 3: inizializza il tuo portachiavi gerarchico con il fornitore dell'ID della chiave di filiale

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

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

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

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

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

Default (Recommended)

Per la maggior parte degli utenti, la cache predefinita soddisfa i requisiti di threading. La cache predefinita è progettata per supportare ambienti con molti multithread. Quando scade l'immissione di materiali relativi a una chiave di filiale, la cache predefinita impedisce la chiamata a più thread AWS KMS e Amazon DynamoDB notificando a un thread che l'immissione dei materiali chiave del ramo sta per scadere con 10 secondi di anticipo. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache. AWS KMS

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

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

C#/.NET

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

## Java

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

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

### StormTracking

### MultiThreaded

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

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

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

### 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

La StormTracking cache è progettata per supportare ambienti con molti multithread. Quando l'immissione di materiali chiave di filiale scade, la StormTracking cache impedisce a più thread di richiamare AWS KMS Amazon DynamoDB notificando in anticipo a un thread che l'immissione dei materiali chiave del ramo sta per scadere. Ciò garantisce che solo un thread invii una richiesta di aggiornamento della cache. AWS KMS

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

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

Valore predefinito: 1 voce

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

Valore predefinito: 10 secondi

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

Valore predefinito: 1 secondo

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

Valore predefinito: 20 tentativi

- In flight time to live (TTL): definisce il numero di secondi che mancano al timeout di un tentativo di aggiornamento dei materiali chiave della filiale. Ogni volta che la cache

ritorna `NoSuchEntry` in risposta a `unaGetCacheEntry`, quella chiave di ramo viene considerata in esecuzione finché la stessa chiave non viene scritta con una `PutCache` voce.

Valore predefinito: 20 secondi

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

Valore predefinito: 20 millisecondi

## 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())
```

- (Facoltativo) Un elenco di token di concessione

Se controlli l'accesso alla chiave KMS nel tuo portachiavi gerarchico con le [concessioni](#), [devi fornire tutti i](#) token di concessione necessari quando iniziizzi il portachiavi.

L'esempio seguente inizializza un portachiavi Hierarchical con il branch key ID supplier creato nel passaggio 2, un TTL con limite di cache di 600 secondi e una capacità di ingresso di 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(branchKeyStoreName)
        .branchKeyIdSupplier(branchKeyIdSupplier)
        .ttlSeconds(600)
        .cache(CacheType.builder() //OPTIONAL
            .Default(DefaultCache.builder()
                .entryCapacity(100)
                .build())
            .build());
final IKeyring hierarchicalKeyring =
    matProv.CreateAwsKmsHierarchicalKeyring(keyringInput);
```

#### Passaggio 4: Creare nomi descrittivi per ogni chiave di filiale

L'esempio seguente crea nomi descrittivi per le due chiavi di ramo create nel passaggio 1. AWS Encryption SDK utilizza contesti di crittografia per mappare il nome descrittivo definito dall'utente a quello associato a `branch-key-id`.

## C# / .NET

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

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

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

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

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

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

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

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

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

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

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

```
    }  
}
```

## Java

```
// Create encryption context for branchKeyTenantA  
Map<String, String> encryptionContextA = new HashMap<>();  
encryptionContextA.put("tenant", "TenantA");  
encryptionContextA.put("encryption", "context");  
encryptionContextA.put("is not", "secret");  
encryptionContextA.put("but adds", "useful metadata");  
encryptionContextA.put("that can help you", "be confident that");  
encryptionContextA.put("the data you are handling", "is what you think it is");  
  
// Create encryption context for branchKeyTenantB  
Map<String, String> encryptionContextB = new HashMap<>();  
encryptionContextB.put("tenant", "TenantB");  
encryptionContextB.put("encryption", "context");  
encryptionContextB.put("is not", "secret");  
encryptionContextB.put("but adds", "useful metadata");  
encryptionContextB.put("that can help you", "be confident that");  
encryptionContextB.put("the data you are handling", "is what you think it is");  
  
// Instantiate the AWS Encryption SDK  
final AwsCrypto crypto = AwsCrypto.builder().build();  
  
final CryptoResult<byte[], ?> encryptResultA = crypto.encryptData(keyring,  
    plaintext, encryptionContextA);  
  
final CryptoResult<byte[], ?> encryptResultB = crypto.encryptData(keyring,  
    plaintext, encryptionContextB);  
  
// Use the encryption contexts to define friendly names for each branch key  
public class ExampleBranchKeyIdSupplier implements IBranchKeyIdSupplier {  
    private static String branchKeyIdForTenantA;  
    private static String branchKeyIdForTenantB;  
  
    public ExampleBranchKeyIdSupplier(String tenant1Id, String tenant2Id) {  
        this.branchKeyIdForTenantA = tenant1Id;  
        this.branchKeyIdForTenantB = tenant2Id;  
    }  
}
```

```
@Override
public GetBranchKeyIdOutput GetBranchKeyId(GetBranchKeyIdInput input) {

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

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

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

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

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

## Keyring non elaborati AES

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

Il portachiavi Raw AES crittografa i dati utilizzando l'algoritmo AES-GCM e una chiave di wrapping specificata come array di byte. [È possibile specificare solo una chiave di avvolgimento in ogni portachiavi Raw AES, ma è possibile includere più portachiavi Raw AES, da soli o con altri portachiavi, in un portachiavi multiplo.](#)

Il portachiavi Raw AES è equivalente e interagisce con la [JceMasterKey](#) classe in e con la classe in SDK di crittografia AWS per Python quando viene SDK di crittografia AWS per Java utilizzato con una chiave di crittografia AES. [RawMasterKey](#) È possibile crittografare e decrittare i dati con implementazioni diverse, ma utilizzando la stessa chiave di wrapping. Per informazioni dettagliate, consultare [Compatibilità dei keyring](#).

## Namespace e nomi chiave

Per identificare la chiave AES in un portachiavi, il portachiavi Raw AES utilizza uno spazio dei nomi e un nome chiave forniti dall'utente. Questi valori non sono segreti. Vengono visualizzati in testo semplice nell'intestazione del [messaggio crittografato restituito dall'operazione](#) di crittografia. Si consiglia di utilizzare uno spazio dei nomi delle chiavi (HSM o sistema di gestione delle chiavi) e un nome di chiave che identifichi la chiave AES in quel sistema.

### Note

Lo spazio dei nomi e il nome della chiave sono equivalenti ai campi Provider ID (o Provider) e Key ID presenti nel e. `JceMasterKey` `RawMasterKey` SDK di crittografia AWS per Cand AWS Encryption SDK for .NET riserva il valore dello spazio dei nomi `aws-kms` chiave per le chiavi KMS. Non utilizzare questo valore dello spazio dei nomi in un portachiavi Raw AES o Raw RSA con queste librerie.

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

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

I seguenti esempi mostrano come creare un portachiavi Raw AES. La `AESWrappingKey` variabile rappresenta il materiale chiave fornito.

## C

Per creare un'istanza di un portachiavi Raw AES in, usa. SDK di crittografia AWS per C `aws_cryptosdk_raw_aes_keyring_new()` [Per un esempio completo, vedete `raw\_aes\_keyring.c`.](#)

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

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

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

## C# / .NET

Per creare un portachiavi Raw AES in.NET, usa il metodo. AWS Encryption SDK `materialProviders.CreateRawAesKeyring()` Per un esempio completo, consulta [KeyringExampleRawAES.cs](#).

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .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

SDK di crittografia AWS per JavaScriptNel browser ottiene le sue primitive crittografiche dall'API. [WebCrypto](#) Prima di costruire il portachiavi, è necessario `RawAesKeyringWebCrypto.importCryptoKey()` importare il materiale grezzo della chiave nel backend. WebCrypto Ciò garantisce che il portachiavi sia completo anche se tutte le chiamate a sono asincrone. WebCrypto

Quindi, per creare un'istanza di un portachiavi Raw AES, usa il metodo.

`RawAesKeyringWebCrypto()` È necessario specificare l'algoritmo di wrapping AES («wrapping suite») in base alla lunghezza del materiale chiave. Per un esempio completo, vedete [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

Per creare un'istanza di un portachiavi Raw AES in SDK di crittografia AWS per JavaScript for Node.js, create un'istanza della classe. `RawAesKeyringNode` È necessario specificare l'algoritmo di wrapping AES («wrapping suite») in base alla lunghezza del materiale chiave. Per un esempio completo, vedete [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

Per creare un'istanza di un portachiavi Raw AES in, usa. SDK di crittografia AWS per Java `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);
```

## Keyring non elaborato RSA

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

Un keyring non elaborato RSA che esegue crittografia e decrittazione deve includere una coppia di chiavi pubblica e privata asimmetriche. Tuttavia, è possibile crittografare i dati con un portachiavi

Raw RSA che ha solo una chiave pubblica e decrittografare i dati con un portachiavi Raw RSA che ha solo una chiave privata. [Puoi includere qualsiasi portachiavi Raw RSA in un portachiavi multiplo.](#) Se configuri un portachiavi Raw RSA con una chiave pubblica e una privata, assicurati che facciano parte della stessa coppia di chiavi. Alcune implementazioni linguistiche di non AWS Encryption SDK costruiranno un portachiavi Raw RSA con chiavi di coppie diverse. Altri si affidano a te per verificare che le tue chiavi appartengano alla stessa coppia di chiavi.

Il portachiavi Raw RSA è equivalente e interagisce con l'[JceMasterKey](#) in SDK di crittografia AWS per Java e l'[RawMasterKey](#) in the SDK di crittografia AWS per Python quando viene utilizzato con chiavi di crittografia asimmetriche RSA. È possibile crittografare e decrittare i dati con implementazioni diverse, ma utilizzando la stessa chiave di wrapping. Per informazioni dettagliate, consultare [Compatibilità dei keyring](#).

#### Note

Il portachiavi Raw RSA non supporta le chiavi KMS asimmetriche. Se desideri utilizzare chiavi RSA KMS asimmetriche, versione 4. x di AWS Encryption SDK per .NET e versione 3. x dei AWS KMS portachiavi di SDK di crittografia AWS per Java supporto che utilizzano la crittografia simmetrica (SYMMETRIC\_DEFAULT) o l'RSA asimmetrica. AWS KMS keys Se si crittografano i dati con un portachiavi Raw RSA che include la chiave pubblica di una chiave RSA KMS, né l'utente né lo possono decriptare. AWS Encryption SDK AWS KMS Non è possibile esportare la chiave privata di una chiave KMS AWS KMS asimmetrica in un portachiavi RSA Raw. [L'operazione Decrypt non può AWS KMS decrittografare il messaggio crittografato che restituisce.](#) AWS Encryption SDK

Quando crei un keyring non elaborato RSA in SDK di crittografia AWS per C, assicurati di fornire i contenuti del file PEM che include ciascuna chiave sotto forma di stringa C con terminazione null, non come percorso o nome di file. [Quando crei un portachiavi Raw RSA JavaScript, tieni presente la potenziale incompatibilità con altre implementazioni linguistiche.](#)

#### Namespace e nomi

Per identificare il materiale chiave RSA in un portachiavi, il portachiavi Raw RSA utilizza uno spazio dei nomi e un nome chiave forniti dall'utente. Questi valori non sono segreti. Vengono visualizzati in testo semplice nell'intestazione del messaggio [crittografato](#) restituito dall'operazione di crittografia. Ti consigliamo di utilizzare lo spazio dei nomi e il nome della chiave che identificano la coppia di chiavi RSA (o la relativa chiave privata) nel tuo HSM o sistema di gestione delle chiavi.

**Note**

Lo spazio dei nomi della chiave e il nome della chiave sono equivalenti ai campi Provider ID (o Provider) e Key ID presenti nel `JceMasterKey RawMasterKey`.  
The SDK di crittografia AWS per C riserva il valore dello spazio dei nomi `aws-kms` chiave per le chiavi KMS. Non utilizzarlo in un portachiavi Raw AES o Raw RSA con. SDK di crittografia AWS per C

Se si creano portachiavi diversi per crittografare e decrittografare un determinato messaggio, lo spazio dei nomi e i valori del nome sono fondamentali. Se lo spazio dei nomi della chiave e il nome della chiave nel portachiavi di decrittografia non corrispondono esattamente, con distinzione tra maiuscole e minuscole, per lo spazio dei nomi della chiave e il nome della chiave nel portachiavi di crittografia, il portachiavi di decrittografia non viene utilizzato, anche se le chiavi appartengono alla stessa coppia di chiavi.

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

**Modalità di imbottitura**

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

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

- OAEP con SHA-1 e MGF1 con imbottitura SHA-1
- OAEP con SHA-256 e MGF1 con imbottitura SHA-256
- OAEP con SHA-384 e MGF1 con imbottitura SHA-384
- OAEP con SHA-512 e MGF1 con imbottitura SHA-512

- Imbottitura PKCS1 v1.5

Gli esempi seguenti mostrano come creare un portachiavi Raw RSA con la chiave pubblica e privata di una coppia di chiavi RSA e l'OAEP con SHA-256 e MGF1 con modalità padding SHA-256. `RSAPublicKey` e `RSAPrivateKey` le variabili and rappresentano il materiale chiave fornito.

## C

Per creare un portachiavi Raw RSA in SDK di crittografia AWS per C, usa.

```
aws_cryptosdk_raw_rsa_keyring_new
```

Quando crei un keyring non elaborato RSA in SDK di crittografia AWS per C, assicurati di fornire i contenuti del file PEM che include ciascuna chiave sotto forma di stringa C con terminazione null, non come percorso o nome di file. [Per un esempio completo, vedi `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

Per creare un'istanza di un portachiavi Raw RSA in formato .NET, usa il metodo `AWS Encryption SDK materialProviders.CreateRawRsaKeyring()` [Per un esempio completo, vedi `RawRSA.cs.KeyringExample`](#)

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .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

SDK di crittografia AWS per JavaScriptNel browser ottiene le sue primitive crittografiche dalla libreria. [WebCrypto](#) Prima di costruire il portachiavi, è necessario utilizzare `importPublicKey()` e/o `importPrivateKey()` importare il materiale chiave grezzo nel backend. WebCrypto Ciò garantisce che il portachiavi sia completo anche se tutte le chiamate a sono asincrone. WebCrypto L'oggetto utilizzato dai metodi di importazione include l'algoritmo di wrapping e la relativa modalità di riempimento.

Dopo aver importato il materiale chiave, utilizzate il `RawRsaKeyringWebCrypto()` metodo per creare un'istanza del portachiavi. [Quando crei un portachiavi Raw RSA in JavaScript, tieni presente la potenziale incompatibilità con altre implementazioni linguistiche.](#)

[Per un esempio completo, vedete `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

Per creare un'istanza di un portachiavi Raw RSA in Node.js, create una nuova istanza della SDK di crittografia AWS per JavaScript classe. `RawRsaKeyringNode` Il `wrapKey` parametro contiene la chiave pubblica. Il `unwrapKey` parametro contiene la chiave privata. Il `RawRsaKeyringNode` costruttore calcola automaticamente una modalità di riempimento predefinita, sebbene sia possibile specificare una modalità di riempimento preferita.

[Quando crei un portachiavi RSA non elaborato JavaScript, tieni presente la potenziale incompatibilità con altre implementazioni linguistiche.](#)

[Per un esempio completo, vedete `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);
```

## Keyring multipli

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

Quando crei un keyring multiplo per crittografare i dati, uno dei keyring viene designato come keyring generatore, tutti gli altri keyring sono i keyring figlio. che si occupa di generare e crittografare la chiave di dati di testo normale. Quindi, tutte le chiavi di wrapping in tutti i keyring figlio crittografano la stessa chiave di dati di testo normale. Il keyring multiplo restituisce la chiave di dati di testo normale e una chiave di dati crittografata per ciascuna chiave di wrapping nel keyring multiplo. Se crei un keyring multiplo senza keyring generatore, puoi utilizzarlo per decrittare i dati, ma non per crittografarli. Se il portachiavi del generatore è un portachiavi [KMS, la chiave del generatore nel portachiavi](#) genera e crittografa la chiave in testo semplice. AWS KMS Quindi, tutte le chiavi aggiuntive AWS KMS keys presenti nel portachiavi e tutte le AWS KMS chiavi inserite in tutti i portachiavi secondari del portachiavi multiplo crittografano la stessa chiave in chiaro.

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

[A partire dalla versione 1.7. x](#), quando una chiave dati crittografata viene crittografata con un portachiavi AWS Key Management Service (AWS KMS) (o provider di chiavi master), passa AWS Encryption SDK sempre l'ARN della chiave al parametro AWS KMS key KeyId AWS KMS [dell'](#)operazione Decrypt. Si tratta di una procedura AWS KMS consigliata che garantisce la decrittografia della chiave dati crittografata con la chiave di wrapping che si intende utilizzare.

Per un esempio di utilizzo di un keyring multiplo consulta:

- C: [multi\\_keyring.cpp](#)
- C# /.NET: [.cs MultiKeyringExample](#)
- JavaScript [Node.js: multi\\_keyring.ts](#)
- JavaScript Browser: [multi\\_keyring.ts](#)

- [MultiKeyringExampleGiava: .java](#)

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

## 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);
```

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

## C

Nel costruttore del keyring multiplo in C, specifica solo il keyring generatore.

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

Per aggiungere un keyring figlio al tuo keyring multiplo, usa il metodo `aws_cryptosdk_multi_keyring_add_child`. Devi chiamare il metodo una volta per ogni keyring figlio che aggiungi.

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

## C# / .NET

Il `CreateMultiKeyringInput` costruttore .NET consente di definire un portachiavi del generatore e dei portachiavi secondari. L'`CreateMultiKeyringInput` oggetto risultante è immutabile.

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

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

## JavaScript Browser

JavaScript i portachiavi multipli sono immutabili. Il JavaScript costruttore multi-portachiavi consente di specificare il portachiavi del generatore e più portachiavi per bambini.

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

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

## JavaScript Node.js

JavaScript i portachiavi multipli sono immutabili. Il JavaScript costruttore multi-portachiavi consente di specificare il portachiavi del generatore e più portachiavi per bambini.

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

## Java

Il `CreateMultiKeyringInput` costruttore Java consente di definire un portachiavi del generatore e dei portachiavi secondari. L'`createMultiKeyringInput` oggetto risultante è immutabile.

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

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

# Linguaggi di programmazione AWS Encryption SDK

Il AWS Encryption SDK è disponibile per i seguenti linguaggi di programmazione. Tutte le implementazioni linguistiche sono interoperabili. È possibile crittografare con un'implementazione di una lingua e decrittografare con un'altra. L'interoperabilità potrebbe essere soggetta a vincoli linguistici. In tal caso, questi vincoli sono descritti nell'argomento relativo all'implementazione della lingua. Inoltre, durante la crittografia e la decrittografia, è necessario utilizzare keyring compatibili o chiavi master e provider di chiavi master. Per informazioni dettagliate, consulta [the section called “Compatibilità dei keyring”](#).

## Argomenti

- [SDK di crittografia AWS per C](#)
- [AWS Encryption SDK per .NET](#)
- [SDK di crittografia AWS per Java](#)
- [SDK di crittografia AWS per JavaScript](#)
- [SDK di crittografia AWS per Python](#)
- [Interfaccia a riga di comando AWS Encryption SDK](#)

## SDK di crittografia AWS per C

SDK di crittografia AWS per C fornisce una libreria di crittografia lato client agli sviluppatori che scrivono applicazioni in C e funge da base per le implementazioni di AWS Encryption SDK nei linguaggi di programmazione di livello superiore.

Come per tutte le implementazioni di AWS Encryption SDK, SDK di crittografia AWS per C offre le caratteristiche di protezione dei dati avanzate. Queste includono la [crittografia envelope](#), dati autenticati aggiuntivi (AAD) e [suite di algoritmi](#) di chiavi simmetriche, autenticate e sicure, come, ad esempio, AES-GCM a 256 bit con derivazione della chiave e firma.

Tutte le implementazioni specifiche per il linguaggio dell'AWS Encryption SDK sono completamente interoperabili. Ad esempio, puoi crittografare i dati con SDK di crittografia AWS per C e decrittografalo con [qualsiasi implementazione linguistica supportata](#), compreso il kit [AWS CLI di crittografia](#).

La SDK di crittografia AWS per C richiede il kit AWS SDK for C++ per interagire con AWS Key Management Service (AWS KMS). È necessario utilizzarlo solo se si sta utilizzando l'opzione

opzionale [AWS KMS Keyring](#). L'AWS Encryption SDK non richiede comunque AWS KMS, né altri servizi AWS.

### Ulteriori informazioni

- Per informazioni dettagliate SDK di crittografia AWS per C, consulta [Esempi di di C](#), [ilesempinellaaws-encryption-sdk-c repository](#) sul GitHub, e il [SDK di crittografia AWS per C Documentazione API](#).
- Per una discussione su come utilizzare il SDK di crittografia AWS per C per crittografare i dati in modo da poterli decrittografare in più Regioni AWS, consulta [Come decrittografare i testi cifrati in più regioni con il AWS Encryption SDK in C](#) nella AWS Blog di sicurezza.

### Argomenti

- [Installazione di SDK di crittografia AWS per C](#)
- [Utilizzo di SDK di crittografia AWS per C](#)
- [SDK di crittografia AWS per C Esempi di](#)

## Installazione di SDK di crittografia AWS per C

Installazione della versione più recente di SDK di crittografia AWS per C.

### Note

Tutte le versioni di SDK di crittografia AWS per C le versioni precedenti alla versione 2.0.0 sono incluse in [end-of-support fase](#).

Puoi eseguire l'aggiornamento in tutta sicurezza dalla versione 2.0.xe successivamente alla versione più recente di SDK di crittografia AWS per C senza modifiche al codice o ai dati.

Tuttavia, [nuove funzioni di sicurezza](#) introdotto nella versione 2.0.x non sono compatibili con le versioni precedenti. Per eseguire l'aggiornamento da versioni precedenti alla 1.7.x alla versione 2.0.xe successivamente, è necessario eseguire l'aggiornamento alla versione più recente di 1.x versione del SDK di crittografia AWS per C. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Puoi trovare istruzioni dettagliate per l'installazione e la costruzione del SDK di crittografia AWS per C nel [file README](#) del [aws-encryption-sdk-c repository](#). Include istruzioni per la creazione su piattaforme Amazon Linux, Ubuntu, macOS e Windows.

Prima di iniziare, decidi se utilizzarlo [AWS KMSportachiavi](#) nell'AWS Encryption SDK. Se utilizzi un plugin [AWS KMSportachiavi](#), è necessario installare [AWS SDK for C++](#). Il [AWS SDK](#) è necessario per interagire con [AWS Key Management Service](#) (AWS KMS). Quando si utilizza [AWS KMSportachiavi](#), il [AWS Encryption SDK](#) utilizza [AWS KMS](#) per generare e proteggere le chiavi di crittografia che proteggono i dati.

Non è necessario installare il plugin [AWS SDK for C++](#) se utilizzi un altro tipo di portachiavi, ad esempio un portachiavi AES non elaborato, un portachiavi RSA non elaborato o un portachiavi multiplo che non include [AWS KMSportachiavi](#). Tuttavia, quando si utilizza un tipo di portachiavi non elaborato, è necessario generare e proteggere le proprie chiavi raw wrapping.

Per informazioni sulla scelta dei tipi di portachiavi da utilizzare, consulta [the section called “Scegliere un portachiavi”](#).

In caso di problemi con l'installazione, [segnalare un problema](#) nel [aws-encryption-sdk-c](#) repository o utilizza uno dei link di feedback in questa pagina.

## Utilizzo di SDK di crittografia AWS per C

Questo argomento illustra alcune delle caratteristiche del SDK di crittografia AWS per C che non sono supportate in altre implementazioni del linguaggio di programmazione.

Gli esempi in questa sezione mostrano come utilizzare [versione 2.0.xe](#) successivamente del SDK di crittografia AWS per C. Per esempi che utilizzano versioni precedenti, trova la versione nella sezione [Versioni di](#) elenco di [repository aws-encryption-sdk-c](#) repository di in GitHub.

Per informazioni dettagliate sulla programmazione con SDK di crittografia AWS per C, vedere gli [esempi C](#), gli [esempi](#) in [aws-encryption-sdk-c repository](#) su GitHub e la [documentazione dell'API SDK di crittografia AWS per C](#).

Consulta anche: [Utilizzo dei keyring](#)

### Argomenti

- [Modelli per la crittografia e la decrittazione dei dati](#)
- [Conteggio dei riferimenti](#)

## Modelli per la crittografia e la decrittazione dei dati

Quando utilizzi il SDK di crittografia AWS per C, segui un modello simile al seguente: crei un [keyring](#), un [CMM](#) che utilizza il keyring, una sessione che utilizza il CMM (e il keyring) e infine elabori la sessione.

### 1. Carica stringhe di errore.

Chiama il `aws_cryptosdk_load_error_strings()` metodo nel codice C o C++. Carica informazioni di errore molto utili per il debug.

Devi chiamarlo solo una volta, come nel tuo `main` metodo.

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

### 2. Crea un keyring.

Configura il [keyring](#) con le chiavi di wrapping da utilizzare per crittografare le chiavi di dati. Questo esempio utilizza un [AWS KMSKeyring](#) con una AWS KMS key ma puoi scegliere qualsiasi tipo di keyring al suo posto.

Per identificare un AWS KMS key in un portachiavi di crittografia nel SDK di crittografia AWS per C, specificare un [ARN della chiave](#) o [ARN dell'alias](#). In un keyring di decrittografia devi utilizzare un ARN di chiave. Per dettagli, consultare [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

```
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. Crea una sessione.

Nel SDK di crittografia AWS per C, per crittografare un singolo messaggio di testo normale o decrittare un singolo messaggio di testo cifrato, indipendentemente dalle dimensioni, viene utilizzata una sessione. La sessione mantiene lo stato del messaggio durante tutta l'elaborazione.

Configura la sessione con un allocatore, un CMM e una modalità: `AWS_CRYPTOSDK_ENCRYPT` o `AWS_CRYPTOSDK_DECRYPT`. Per modificare la modalità della sessione, utilizza il metodo `aws_cryptosdk_session_reset`.

Quando crei una sessione con un keyring, il SDK di crittografia AWS per C crea automaticamente un gestore di materiali crittografici (CMM) predefinito. Non è necessario creare, mantenere o eliminare in modo permanente questo oggetto.

Ad esempio, la sessione seguente utilizza l'allocatore e il keyring definito nella fase 1. Quando esegui la crittografia dei dati, la modalità è `AWS_CRYPTOSDK_ENCRYPT`.

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

#### 4. Esegui la crittografia o la decrittazione dei dati.

Per elaborare i dati della sessione, utilizza il metodo `aws_cryptosdk_session_process`. Se il buffer di input è abbastanza grande da contenere l'intero testo normale e il buffer di output è abbastanza grande da contenere l'intero testo cifrato, è possibile chiamare `aws_cryptosdk_session_process_full`. Tuttavia, se devi gestire i dati di streaming, puoi chiamare `aws_cryptosdk_session_process` in un loop. Per un esempio, consulta [file\\_streaming.cpp](#). La `aws_cryptosdk_session_process_full` è introdotto in AWS Encryption SDK versioni 1.9.x e 2.2.x.

Quando la sessione è configurata per crittografare i dati, i campi di testo normale descrivono l'input, mentre quelli di testo cifrato l'output. Il campo `plaintext` contiene il messaggio da crittografare, mentre il campo `ciphertext` riceve il [messaggio crittografato](#) restituito dal metodo di crittografia.

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

Quando la sessione è configurata per decrittare i dati, i campi di testo cifrato descrivono l'input, mentre quelli di testo normale l'output. Il campo `ciphertext` contiene il [messaggio crittografato](#) restituito dal metodo di crittografia, mentre il campo `plaintext` riceve il messaggio di testo normale restituito dal metodo di decrittazione.

Per decrittare i dati, chiama il metodo `aws_cryptosdk_session_process_full`.

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

## Conteggio dei riferimenti

Per evitare perdite di memoria, assicurati di rilasciare i riferimenti a tutti gli oggetti che crei quando finisci di utilizzarli. In caso contrario, potrebbero verificarsi perdite di memoria. L'SDK offre dei metodi per semplificare questa operazione.

Ogni volta che crei un oggetto padre con uno dei seguenti oggetti figlio, l'oggetto padre ottiene e mantiene un riferimento all'oggetto figlio, come indicato di seguito:

- Un [keyring](#), ad esempio la creazione di una sessione con un keyring
- Un predefinito [responsabile dei materiali crittografici](#) (CMM), ad esempio la creazione di una sessione o di un CMM personalizzato con un CMM predefinito
- Una [cache della chiave di dati](#), ad esempio un CMM di caching con un keyring e una cache

A meno che non sia necessario un riferimento indipendente all'oggetto figlio, puoi rilasciare il riferimento all'oggetto figlio non appena crei l'oggetto padre. Il rimanente riferimento all'oggetto figlio viene rilasciato quando l'oggetto padre viene eliminato in modo permanente. Questo modello consente di mantenere il riferimento a ogni oggetto solo per il tempo necessario ed evita le perdite di memoria causate da riferimenti non rilasciati.

Sarai responsabile solo del rilascio dei riferimenti agli oggetti figlio creati in modo esplicito. Non sei responsabile della gestione dei riferimenti a qualsiasi oggetto creato dall'SDK. Se l'SDK crea un oggetto, ad esempio il CMM predefinito che `ilaws_cryptosdk_caching_cmm_new_from_keyring`Il metodo aggiunge a una sessione, l'SDK gestisce la creazione e la distruzione dell'oggetto e dei relativi riferimenti.

Nell'esempio seguente, quando crei una sessione con un [keyring](#), la sessione ottiene un riferimento al keyring e mantiene tale riferimento fino a quando la sessione non viene eliminata in modo permanente. Se non è necessario mantenere un riferimento aggiuntivo al keyring, puoi utilizzare il metodo `aws_cryptosdk_keyring_release` per rilasciare l'oggetto keyring non appena viene creata la sessione. Questo metodo riduce il conteggio dei riferimenti per il keyring. Il riferimento della sessione al keyring viene rilasciato quando chiami `aws_cryptosdk_session_destroy` per eliminare in modo permanente la sessione.

```
// 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);
```

Per attività più complesse, ad esempio riutilizzare un keyring per più sessioni o specificare una suite di algoritmi in un CMM, potrebbe essere necessario mantenere un riferimento indipendente all'oggetto. In tal caso, non chiamare immediatamente i metodi di rilascio. Al contrario, rilascia i riferimenti quando non utilizzi più gli oggetti, oltre ad eliminare in modo permanente la sessione.

Questa tecnica di conteggio dei riferimenti funziona anche quando si utilizzano CMM alternativi, ad esempio il di caching CMM per [caching della chiave dei dati](#). Quando crei un CMM di caching da una cache e da un keyring, il CMM di caching ottiene un riferimento a entrambi gli oggetti. A meno che non siano necessari per un'altra attività, è possibile rilasciare i riferimenti indipendenti alla cache e al keyring non appena viene creato il CMM di caching. Quindi, quando crei una sessione con il CMM di caching, puoi rilasciare il riferimento al CMM di caching.

Sei responsabile solo del rilascio dei riferimenti agli oggetti creati in modo esplicito. Gli oggetti creati dai metodi creati per l'utente, ad esempio il CMM predefinito alla base del di caching CMM, vengono gestiti dal metodo.

```
/ 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);
```

## SDK di crittografia AWS per C Esempi di

Gli esempi seguenti illustrano come utilizzare SDK di crittografia AWS per C per crittografare e decrittare i dati.

Gli esempi in questa sezione mostrano come utilizzare le versioni 2.0.xe più tardi del SDK di crittografia AWS per C. Per esempi che utilizzano versioni precedenti, trova la tua versione nella [Versioni di](#) elenco dei [aws-encryption-sdk-c repository](#) repository GitHub.

Quando installi e crei l'SDK di crittografia AWS per C, il codice sorgente per questi e altri esempi è incluso nella sottodirectory `examples` e sono compilati e integrati nella directory `build`. È possibile anche trovarle nel [esempi](#) sottodirectory del [aws-encryption-sdk-c repository](#) GitHub.

### Argomenti

- [Crittografia e decrittazione di stringhe](#)

## Crittografia e decrittazione di stringhe

L'esempio seguente illustra come utilizzare SDK di crittografia AWS per C per crittografare e decrittare una stringa.

Questo esempio presenta il [AWS KMS Keyring](#), un tipo di portachiavi che utilizza un AWS KMS key nel [AWS Key Management Service \(AWS KMS\)](#) per generare e crittografare le chiavi dati. L'esempio include il codice scritto in C++. La SDK di crittografia AWS per C richiede il AWS SDK for C++ per chiamare AWS KMS quando si utilizza AWS KMS keyring. Se usi un portachiavi che non interagisce con AWS KMS, come un portachiavi AES grezzo, un portachiavi RSA grezzo o un portachiavi multiplo che non include AWS KMS Keyring, il AWS SDK for C++ non è obbligatorio.

Per aiuto nella creazione di unAWS KMS key, consulta [Creazione di chiavi](#) nellaAWS Key Management ServiceGuida per gli sviluppatori. Per aiuto nell'identificazione delAWS KMS keyin unAWS KMSkeyring, consulta [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

Vedi l'esempio di codice completo: [string.cpp](#)

## Argomenti

- [Crittografare una stringa](#)
- [Decrittare una stringa](#)

## Crittografare una stringa

La prima parte di questo esempio utilizza unaAWS KMSKeyring con unaAWS KMS keyper crittografare una stringa di testo normale.

Fase 1. Carica le stringhe di errore.

Chiama ilaws\_cryptosdk\_load\_error\_strings()nel codice C o C++. Carica informazioni di errore molto utili per il debug.

Devi chiamarlo solo una volta, come nel tuomainmetodo.

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

Fase 2: Costruisci il keyring.

Creazione di unAWS KMSkeyring per crittografare. Il portachiavi in questo esempio è configurato con unoAWS KMS key, ma è possibile configurare unAWS KMSKeyring con piùAWS KMS keys, inclusoAWS KMS keyin diversiRegioni AWS e conti diversi.

Per identificare unAWS KMS keyin un portachiavi di crittografia nelSDK di crittografia AWS per C, specificare un[ARN della chiave](#)o[ARN di alias](#). In un keyring di decrittografia devi utilizzare un ARN di chiave. Per dettagli, consultare [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

## [Identificazione di AWS KMS keys in un keyring AWS KMS](#)

Quando crei un portachiavi con piùAWS KMS keys, si specifica ilAWS KMS keyutilizzato per generare e crittografare la chiave di dati di testo normale e una serie opzionale diAWS KMS keysche eseguono la crittografia della stessa chiave. In questo caso, è necessario specificare solo il generatoreAWS KMS key.

Prima di eseguire questo codice, sostituisci l'ARN della chiave di esempio con uno valido.

```
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);
```

### Fase 3: Crea una sessione.

Crea una sessione utilizzando l'allocatore, un enumeratore di modalità e il keyring.

Ogni sessione richiede una modalità: `AWS_CRYPTOSDK_ENCRYPT` per la crittografia e `AWS_CRYPTOSDK_DECRYPT` per la decrittazione. Per modificare la modalità di una sessione esistente, utilizza il metodo `aws_cryptosdk_session_reset`.

Dopo aver creato una sessione con il keyring, puoi rilasciare il riferimento al keyring con il metodo fornito dall'SDK. La sessione mantiene un riferimento all'oggetto keyring durante la sua durata. I riferimenti al keyring e agli oggetti sessione vengono rilasciati quando si distrugge la sessione. Questa tecnica di [conteggio dei riferimenti](#) aiuta a prevenire perdite di memoria e a impedire che gli oggetti vengano rilasciati mentre sono in 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);
```

### Fase 4: Imposta il contesto di crittografia.

Un [contesto di crittografia](#) è rappresentato da tipi di dati autenticati aggiuntivi arbitrari e non segreti. Quando fornisci un contesto durante la crittografia, l'AWS Encryption SDK lo vincola al testo cifrato a livello crittografico, per cui dovrai utilizzare lo stesso contesto di crittografia per decrittare i dati. L'utilizzo di un contesto di crittografia è facoltativo, ma viene consigliato come best practice.

Crea innanzitutto una tabella hash che includa le stringhe del contesto di crittografia.

```
/* 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)
```

Ottieni un puntatore modificabile per il contesto di crittografia nella sessione. Quindi, utilizza la funzione `aws_cryptosdk_enc_ctx_clone` per copiare il contesto di crittografia nella sessione. Una copia viene salvata in `my_enc_ctx` per la convalida del valore dopo la decrittazione dei dati.

Il contesto di crittografia fa parte della sessione, non è un parametro assegnato alla funzione di elaborazione della sessione. Questo garantisce che lo stesso contesto di crittografia venga utilizzato per ogni segmento di un messaggio, anche se la funzione di elaborazione della sessione viene chiamata più volte per crittografare l'intero messaggio.

```
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)
```

## Fase 5: Crittografia della stringa.

Per crittografare la stringa di testo normale, utilizza il metodo `aws_cryptosdk_session_process_full` con la sessione in modalità di crittografia. Questo metodo, introdotto in AWS Encryption SDK versioni 1.9.x e 2.2.x, è progettato per la crittografia e la decrittografia non in streaming. Per gestire i dati in streaming, chiamare `aws_cryptosdk_session_process` in un loop.

Durante la crittografia, i campi in testo normale sono destinati all'input, mentre quelli cifrati sono i campi di output. Al termine dell'elaborazione, il campo `ciphertext_output` contiene il [messaggio crittografato](#), inclusi il testo cifrato effettivo, le chiavi di dati crittografate e il contesto di crittografia. Puoi decrittare questo messaggio crittografato utilizzando AWS Encryption SDK per qualsiasi linguaggio di programmazione supportato.

```
/* 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;
}
```

## Fase 6: Pulire la sessione.

Il passaggio finale distrugge la sessione, includendo i riferimenti al CMM e al keyring.

Se preferisci, invece di distruggere la sessione, puoi riutilizzare la sessione con lo stesso keyring e CMM per decrittare la stringa o per crittografare o decrittare altri messaggi. Per utilizzare la sessione per la decrittazione, scegli il metodo `aws_cryptosdk_session_reset` per modificare la modalità in `AWS_CRYPTOSDK_DECRYPT`.

## Decrittare una stringa

La seconda parte di questo esempio spiega come decrittare un messaggio crittografato che contiene il testo cifrato della stringa originale.

### Fase 1: Carica le stringhe di errore.

Chiama `aws_cryptosdk_load_error_strings()` nel codice C o C++. Carica informazioni di errore molto utili per il debug.

Devi chiamarlo solo una volta, come nel tuo `main` metodo.

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

### Fase 2: Costruisci il keyring.

Quando esegui la decrittazione dei dati in AWS KMS, passi il [messaggio crittografato](#) restituito dall'API di crittografia. La [API di decrittografia](#) non prende AWS KMS key come input. Invece AWS KMS utilizza la stessa AWS KMS key per decrittare il testo cifrato utilizzato per crittografarlo. Tuttavia, il AWS Encryption SDK consente di specificare un AWS KMS Keyring con AWS KMS keys per crittografare e decrittare.

Durante la decrittazione, puoi configurare un keyring solo con le AWS KMS keys da utilizzare per decrittare il messaggio crittografato. Ad esempio, puoi creare un keyring solo con la AWS KMS key utilizzata da un determinato ruolo nella tua azienda. La AWS Encryption SDK non utilizzerà mai un AWS KMS key a meno che non sia presente nel keyring di decrittografia. Se l'SDK non riesce a decrittare le chiavi di dati crittografate con le AWS KMS keys nel keyring fornito, perché nessuna AWS KMS keys nel keyring è stata utilizzata per crittografare le chiavi di dati o perché l'intermediario non ha le autorizzazioni per utilizzare le AWS KMS keys nel keyring per la decrittazione, la chiamata di decrittazione non riesce.

Quando si specifica una AWS KMS key per un keyring di decrittografia è necessario utilizzarne [ARN della chiave](#). [ARN di alias](#) sono consentiti solo nei keyring di crittografia. Per aiuto nell'identificazione del AWS KMS key in un AWS KMS keyring, consulta [Identificazione di AWS KMS keys in un keyring AWS KMS](#).

In questo esempio è stato specificato un keyring configurato con la stessa AWS KMS key utilizzato per crittografare la stringa. Prima di eseguire questo codice, sostituisci l'ARN della chiave di esempio con uno valido.

```
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);
```

### Fase 3: Crea una sessione.

Crea una sessione utilizzando l'allocatore e il keyring. Per configurare la sessione per la decrittazione, configura la sessione con la modalità `AWS_CRYPTOSDK_DECRYPT`.

Dopo aver creato una sessione con un keyring, puoi rilasciare il riferimento al keyring con il metodo fornito dall'SDK. La sessione mantiene un riferimento all'oggetto keyring durante la sua durata e sia la sessione che il keyring vengono rilasciati quando si distrugge la sessione. Questa tecnica di conteggio dei riferimenti aiuta a prevenire perdite di memoria e a impedire che gli oggetti vengano rilasciati mentre sono in 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);
```

#### Fase 4: Decrittografia della stringa.

Per decrittare la stringa, utilizza il metodo `aws_cryptosdk_session_process_full` con la sessione configurata per la decrittazione. Questo metodo, introdotto in AWS Encryption SDK versioni 1.9.xe 2.2.x, è progettato per la crittografia e la decrittografia non in streaming. Per gestire i dati in streaming, chiamare il `aws_cryptosdk_session_process` in un loop.

Durante la decrittazione, i campi in testo cifrato sono destinati all'input, mentre quelli in testo normale sono i campi di output. Il campo `ciphertext_input` contiene il [messaggio crittografato](#) restituito dal metodo di crittografia. Al termine dell'elaborazione, il campo `plaintext_output` contiene la stringa di testo normale (decrittato).

```
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;  
}
```

#### Fase 5: Verifica il contesto di crittografia.

Assicurarsi che il contesto di crittografia effettivo, quello utilizzato per decrittare il messaggio, contenga il contesto di crittografia fornito durante la crittografia del messaggio. Il contesto di crittografia effettivo potrebbe includere coppie supplementari, perché il [responsabile dei materiali crittografici](#) (CMM) può aggiungere delle coppie a tale contesto prima della crittografia del messaggio.

In SDK di crittografia AWS per C, non è necessario fornire un contesto di crittografia durante la decrittazione perché è incluso nel messaggio crittografato restituito dall'SDK. Tuttavia, prima di restituire il messaggio di testo normale, la funzione di decrittazione deve verificare che tutte le coppie nel contesto di crittografia fornito siano anche presenti nel contesto di crittografia utilizzato per decrittare il messaggio.

Ottieni innanzitutto un puntatore di sola lettura per la tabella hash nella sessione. Questa tabella hash contiene il contesto di crittografia utilizzato per decrittare il messaggio.

```
const struct aws_hash_table *session_enc_ctx =  
    aws_cryptosdk_session_get_enc_ctx_ptr(session);
```

Quindi, esamina il contesto di crittografia nella tabella hash `my_enc_ctx` copiata durante la crittografia. Verifica che ogni coppia nella tabella hash `my_enc_ctx` utilizzata per la crittografia sia presente nella tabella hash `session_enc_ctx` utilizzata per la decrittazione. Se una chiave è mancante o ha un valore differente, interrompi l'elaborazione e scrivi un messaggio di errore.

```
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();  
    }  
}
```

Fase 6: Pulire la sessione.

Dopo aver verificato il contesto di crittografia, puoi eliminare la sessione o riutilizzarla. Se devi riconfigurare la sessione, utilizza `aws_cryptosdk_session_reset` metodo.

```
aws_cryptosdk_session_destroy(session);
```

## AWS Encryption SDK per .NET

The AWS Encryption SDK for .NET è una libreria di crittografia lato client per sviluppatori che scrivono applicazioni in C# e altri linguaggi di programmazione .NET. ed è supportata su Windows, macOS e Linux.

Tutte le implementazioni del [linguaggio di programmazione](#) di sono completamente interoperabili AWS Encryption SDK. Tuttavia, se si crittografano i dati utilizzando il [contesto di crittografia richiesto, CMM](#) nella versione 4. x di AWS Encryption SDK per .NET, è possibile decrittografarlo solo con la versione 4. x del AWS Encryption SDK per .NET o versione 3. x del SDK di crittografia AWS per Java.

### Note

La versione 4.0.0 di per.NET si discosta dalla specifica del AWS Encryption SDK messaggio. AWS Encryption SDK Di conseguenza, i messaggi crittografati dalla versione 4.0.0 possono essere decrittografati solo dalla versione 4.0.0 o successiva di per.NET. AWS Encryption SDK Non possono essere decrittografati da nessun'altra implementazione del linguaggio di programmazione.

La versione 4.0.1 di AWS Encryption SDK for .NET scrive messaggi in base alla specifica del AWS Encryption SDK messaggio ed è interoperabile con altre implementazioni del linguaggio di programmazione. Per impostazione predefinita, la versione 4.0.1 è in grado di leggere i messaggi crittografati dalla versione 4.0.0. Tuttavia, se non si desidera decrittografare i messaggi crittografati dalla versione 4.0.0, è possibile specificare la [NetV4\\_0\\_0\\_RetryPolicy](#) proprietà per impedire al client di leggere questi messaggi. Per ulteriori informazioni, consulta le note di [rilascio della versione 4.0.1](#) nel repository su [aws-encryption-sdk-dafny GitHub](#)

The AWS Encryption SDK for .NET si differenzia da alcune delle altre implementazioni del linguaggio di programmazione per i AWS Encryption SDK seguenti motivi:

- Nessun supporto per la memorizzazione nella cache delle chiavi di [dati](#)

### Note

Versione 4. x of the AWS Encryption SDK for .NET supporta il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

- Nessun supporto per lo streaming di dati
- [Nessuna registrazione o traccia dello stack da per.NET](#) AWS Encryption SDK
- [Richiede AWS SDK for .NET](#)

The AWS Encryption SDK for .NET include tutte le funzionalità di sicurezza introdotte nelle versioni 2.0. x e versioni successive di altre implementazioni linguistiche di AWS Encryption SDK. Tuttavia, se si utilizza AWS Encryption SDK for .NET per decrittografare dati crittografati con una versione precedente alla 2.0. [versione x, un'altra implementazione linguistica di AWS Encryption SDK, potrebbe essere necessario modificare la politica di impegno.](#) Per informazioni dettagliate, vedi [Come impostare la tua politica di impegno.](#)

The AWS Encryption SDK for .NET è un prodotto di AWS Encryption SDK in [Dafny](#), un linguaggio di verifica formale in cui si scrivono le specifiche, il codice per implementarle e le bozze per testarle. Il risultato è una libreria che implementa le funzionalità di AWS Encryption SDK in un framework che garantisce la correttezza funzionale.

### Ulteriori informazioni

- Per esempi che mostrano come configurare le opzioni in AWS Encryption SDK, ad esempio la specificazione di una suite di algoritmi alternativa, la limitazione delle chiavi di dati crittografate e l'utilizzo di chiavi multiregionali, vedi. AWS KMS [Configurazione della AWS Encryption SDK](#)
- Per informazioni dettagliate sulla programmazione con AWS Encryption SDK for .NET, consulta la [aws-encryption-sdk-net](#) directory del repository on. aws-encryption-sdk-dafny GitHub

### Argomenti

- [Installazione del file AWS Encryption SDK per.NET](#)
- [Esecuzione del debug di per.NET AWS Encryption SDK](#)
- [AWS KMSportachiavi in AWS Encryption SDK per.NET](#)
- [Contesti di crittografia richiesti nella versione 4.x](#)
- [AWS Encryption SDKper esempi di.NET](#)

## Installazione del file AWS Encryption SDK per.NET

Il AWS Encryption SDK for.NET è disponibile come

[AWS.Cryptography.EncryptionSDK](#) pacchetto in NuGet. Per informazioni dettagliate sull'installazione e la AWS Encryption SDK creazione di per.NET, consulta il file [README.md](#) nel repository. `aws-encryption-sdk-net`

## Versione 3.x

Versione 3. x of the AWS Encryption SDK for .NET supporta .NET Framework 4.5.2 — 4.8 solo su Windows. Supporta .NET Core 3.0+ e .NET 5.0 e versioni successive su tutti i sistemi operativi supportati.

## Versione 4.x

Versione 4. x of the AWS Encryption SDK for .NET supporta .NET 6.0 e .NET Framework net48 e versioni successive.

Il AWS Encryption SDK per.NET richiede le AWS SDK for .NET chiavi anche se non si utilizzano AWS Key Management Service (AWS KMS). Viene installato con il NuGet pacchetto. Tuttavia, a meno che non si utilizzino AWS KMS chiavi, AWS Encryption SDK per.NET non richiede AWS credenziali o interazioni con alcun AWS servizio. Account AWS Per informazioni sulla configurazione di un AWS account, se necessario, consulta [Utilizzo dell'AWS Encryption SDK con AWS KMS](#).

## Esecuzione del debug di per.NET AWS Encryption SDK

Il file AWS Encryption SDK for .NET non genera alcun registro. Le eccezioni in per.NET generano un messaggio di eccezione, ma nessuna traccia dello stack. AWS Encryption SDK

Per aiutarti a eseguire il debug, assicurati di abilitare l'accesso a. AWS SDK for .NET I log e i messaggi di errore di AWS SDK for .NET possono aiutarti a distinguere gli errori derivanti AWS SDK for .NET da quelli presenti in .NET. AWS Encryption SDK Per informazioni sulla AWS SDK for .NET registrazione, consulta la Guida per [AWSLogging](#) gli AWS SDK for .NETsviluppatori. (Per vedere l'argomento, espandi la sezione Apri per visualizzare il contenuto di.NET Framework).

## AWS KMSportachiavi in AWS Encryption SDK per.NET

I AWS KMS portachiavi di base presenti per.NET richiedono solo una chiave KMS. AWS Encryption SDK Richiedono anche un AWS KMS client, che ti dà l'opportunità di configurare il client per la chiave Regione AWS KMS.

Per creare un AWS KMS portachiavi con una o più chiavi avvolgenti, usa un portachiavi multiplo. AWS Encryption SDKFor.NET ha uno speciale portachiavi multiplo che accetta una o più AWS KMS chiavi e un portachiavi multiplo standard che accetta uno o più portachiavi di qualsiasi tipo supportato. Alcuni programmatori preferiscono utilizzare un metodo con portachiavi multipli per creare tutti i propri portachiavi, e for .NET supporta questa strategia. AWS Encryption SDK

## [The AWS Encryption SDK for .NET fornisce portachiavi di base a chiave singola e portachiavi multipli per tutti i casi d'uso tipici, incluse le chiavi multiregionali. AWS KMS](#)

Ad esempio, per creare un AWS KMS portachiavi con una sola AWS KMS chiave, puoi utilizzare il metodo `CreateAwsKmsKeyring()`

### Version 3.x

L'esempio seguente utilizza la versione 3. x di a per.NET AWS Encryption SDK per creare un AWS KMS client predefinito per la regione che contiene la chiave specificata.

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

    AwsCryptographicMaterialProvidersFactory.CreateDefaultAwsCryptographicMaterialProviders();

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

// Instantiate the keyring input object
var kmsKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = keyArn
};

// Create the keyring
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

### Version 4.x

L'esempio seguente utilizza la versione 4. x di a per.NET AWS Encryption SDK per creare un AWS KMS client per la regione che contiene la chiave specificata.

```
// 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);
```

Per creare un portachiavi con una o più AWS KMS chiavi, usa il `CreateAwsKmsMultiKeyring()` metodo. Questo esempio utilizza due AWS KMS chiavi. Per specificare una chiave KMS, utilizzate solo il `Generator` parametro. Il `KmsKeyIds` parametro che specifica le chiavi KMS aggiuntive è facoltativo.

L'input per questo portachiavi non richiede un client. AWS KMS AWS Encryption SDK utilizza invece il AWS KMS client predefinito per ogni regione rappresentato da una chiave KMS nel portachiavi. Ad esempio, se la chiave KMS identificata dal valore del `Generator` parametro si trova nella regione degli Stati Uniti occidentali (Oregon) (`us-west-2`), AWS Encryption SDK crea un AWS KMS client predefinito per la regione. `us-west-2` Se è necessario personalizzare il AWS KMS client, utilizzare il `CreateAwsKmsKeyring()` metodo.

L'esempio seguente utilizza la versione 4. x del AWS Encryption SDK per .NET e il `CreateAwsKmsKeyring()` metodo per personalizzare il AWS KMS client.

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

string generatorKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
List<string> additionalKeys = new List<string> { "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321" };

// Instantiate the keyring input object
var createEncryptKeyringInput = new CreateAwsKmsMultiKeyringInput
{
    Generator = generatorKey,
    KmsKeyIds = additionalKeys
};
```

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

Versione 4. x of the AWS Encryption SDK for .NET supporta i AWS KMS portachiavi che utilizzano la crittografia simmetrica (SYMMETRIC\_DEFAULT) o le chiavi RSA KMS asimmetriche. AWS KMSi portachiavi creati con chiavi RSA KMS asimmetriche possono contenere solo una coppia di chiavi.

Per crittografare con un AWS KMS portachiavi RSA asimmetrico, non è necessario [kms:GenerateDataKey](#) o [KMS:Encrypt](#) perché è necessario specificare il materiale della chiave pubblica che si desidera utilizzare per la crittografia quando si crea il portachiavi. Non viene effettuata alcuna chiamata durante la crittografia con questo portachiavi. AWS KMS [Per decrittografare con un portachiavi AWS KMS RSA asimmetrico, è necessaria l'autorizzazione KMS:Decrypt](#).

Per creare un AWS KMS portachiavi RSA asimmetrico, devi fornire la chiave pubblica e la chiave privata ARN della tua chiave RSA KMS asimmetrica. La chiave pubblica deve essere codificata in PEM. L'esempio seguente crea un AWS KMS portachiavi con una coppia di chiavi RSA asimmetrica.

```
// 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);
```

## Contesti di crittografia richiesti nella versione 4.x

Con la versione 4. x of the AWS Encryption SDK per .NET, è possibile utilizzare il contesto di crittografia richiesto CMM per richiedere [contesti di crittografia](#) nelle operazioni crittografiche. Un

contesto di crittografia è un insieme di coppie chiave-valore non segrete. Il contesto di crittografia è associato crittograficamente ai dati crittografati in modo che sia necessario lo stesso contesto di crittografia per decrittografare il campo. Quando si utilizza il contesto di crittografia richiesto CMM, è possibile specificare una o più chiavi di contesto di crittografia richieste (chiavi obbligatorie) che devono essere incluse in tutte le chiamate di crittografia e decrittografia.

#### Note

Il contesto di crittografia richiesto CMM è interoperabile solo con la versione 3. x di SDK di crittografia AWS per Java. Non è interoperabile con nessun'altra implementazione del linguaggio di programmazione. Se si crittografano i dati utilizzando il contesto di crittografia richiesto CMM, è possibile decrittografarli solo con la versione 3. x della versione o 4. SDK di crittografia AWS per Java o del AWS Encryption SDK per.NET.

In `encrypt`, AWS Encryption SDK verifica che tutte le chiavi del contesto di crittografia richieste siano incluse nel contesto di crittografia specificato. Segna AWS Encryption SDK i contesti di crittografia specificati. Solo le coppie chiave-valore che non sono chiavi richieste vengono serializzate e archiviate in testo semplice nell'intestazione del messaggio crittografato restituito dall'operazione di crittografia.

In `decrypt`, è necessario fornire un contesto di crittografia che contenga tutte le coppie chiave-valore che rappresentano le chiavi richieste. AWS Encryption SDK utilizza questo contesto di crittografia e le coppie chiave-valore memorizzate nell'intestazione del messaggio crittografato per ricostruire il contesto di crittografia originale specificato nell'operazione di crittografia. Se AWS Encryption SDK non è possibile ricostruire il contesto di crittografia originale, l'operazione di decrittografia ha esito negativo. Se si fornisce una coppia chiave-valore che contiene la chiave richiesta con un valore errato, il messaggio crittografato non può essere decrittografato. È necessario fornire la stessa coppia chiave-valore specificata in `encrypt`.

#### Important

Valuta attentamente i valori che scegli per le chiavi richieste nel tuo contesto di crittografia. Devi essere in grado di fornire nuovamente le stesse chiavi e i valori corrispondenti al momento di `decrypt`. Se non riesci a riprodurre le chiavi richieste, il messaggio crittografato non può essere decrittografato.

L'esempio seguente inizializza un AWS KMS portachiavi con il contesto di crittografia richiesto CMM.

```
var encryptionContext = new Dictionary<string, string>()
{
    {"encryption", "context"},
    {"is not", "secret"},
    {"but adds", "useful metadata"},
    {"that can help you", "be confident that"},
    {"the data you are handling", "is what you think it is"}
};

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

// Instantiate the keyring input object
var createKeyringInput = new CreateAwsKmsKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    KmsKeyId = kmsKey
};

// Create the keyring
var kmsKeyring = mpl.CreateAwsKmsKeyring(createKeyringInput);

var createCMMInput = new CreateRequiredEncryptionContextCMMInput
{
    UnderlyingCMM = mpl.CreateDefaultCryptographicMaterialsManager(new
    CreateDefaultCryptographicMaterialsManagerInput{Keyring = kmsKeyring}),
    // If you pass in a keyring but no underlying cmm, it will result in a failure
    because only cmm is supported.
    RequiredEncryptionContextKeys = new List<string>(encryptionContext.Keys)
};

// Create the required encryption context CMM
var requiredEcCMM = mpl.CreateRequiredEncryptionContextCMM(createCMMInput);
```

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

## AWS Encryption SDK per esempi di .NET

Negli esempi seguenti vengono illustrati i modelli di codifica di base utilizzati AWS Encryption SDK per la programmazione con .NET. In particolare, si crea un'istanza della libreria AWS Encryption SDK e della libreria dei fornitori di materiali. Quindi, prima di chiamare ogni metodo, create un'istanza di un oggetto che definisce l'input per il metodo. È molto simile al modello di codifica utilizzato in AWS SDK for .NET

Per esempi che mostrano come configurare le opzioni in AWS Encryption SDK, ad esempio la specificazione di una suite di algoritmi alternativa, la limitazione delle chiavi dati crittografate e l'utilizzo di chiavi AWS KMS multiregionali, vedi. [Configurazione della AWS Encryption SDK](#)

Per altri esempi di programmazione con AWS Encryption SDK for .NET, consulta [gli esempi](#) nella aws-encryption-sdk-net directory del repository su. aws-encryption-sdk-dafny GitHub

### Crittografia dei dati in per .NET AWS Encryption SDK

Questo esempio mostra lo schema di base per la crittografia dei dati. Crittografa un file di piccole dimensioni con chiavi di dati protette da una chiave di AWS KMS wrapping.

Fase 1: Crea un'istanza della libreria AWS Encryption SDK e della libreria dei fornitori di materiali.

Inizia creando un'istanza della libreria AWS Encryption SDK e dei fornitori di materiali. Utilizzerai i metodi descritti in per crittografare e AWS Encryption SDK decrittografare i dati. Utilizzerai i metodi della libreria dei provider di materiali per creare i portachiavi che specificano quali chiavi proteggono i tuoi dati.

Il modo in cui create un'istanza della libreria AWS Encryption SDK e della libreria dei fornitori di materiali differisce tra le versioni 3. x e 4. x del AWS Encryption SDK per .NET. Tutti i passaggi seguenti sono gli stessi per entrambe le versioni 3. x e 4. x del AWS Encryption SDK per .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());
```

## Fase 2: Creare un oggetto di input per il portachiavi.

Ogni metodo che crea un portachiavi ha una classe di oggetti di input corrispondente. Ad esempio, per creare l'oggetto di input per il `CreateAwsKmsKeyring()` metodo, create un'istanza della `CreateAwsKmsKeyringInput` classe.

Anche se l'input per questo portachiavi non specifica una chiave del [generatore, la singola chiave KMS](#) specificata dal `KmsKeyId` parametro è la chiave del generatore. Genera e crittografa la chiave dati che crittografa i dati.

Questo oggetto di input richiede un AWS KMS client per la chiave Regione AWS KMS. Per creare un AWS KMS client, crea un'istanza della `AmazonKeyManagementServiceClient` classe in AWS SDK for .NET. La chiamata al `AmazonKeyManagementServiceClient()` costruttore senza parametri crea un client con i valori predefiniti.

In un AWS KMS portachiavi utilizzato per la crittografia con AWS Encryption SDK for .NET, è possibile [identificare le chiavi KMS utilizzando l'ID](#) della chiave, l'ARN della chiave, il nome alias o l'alias ARN. In un AWS KMS portachiavi utilizzato per la decrittografia, è necessario utilizzare una chiave ARN per identificare ogni chiave KMS. Se prevedi di riutilizzare il tuo portachiavi di crittografia per la decrittografia, utilizza un identificatore ARN di chiave per tutte le chiavi 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
};
```

## Fase 3: Crea il portachiavi.

Per creare il portachiavi, chiamate il metodo `keyring` con l'oggetto di input del portachiavi. Questo esempio utilizza il `CreateAwsKmsKeyring()` metodo, che richiede solo una chiave KMS.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

## Fase 4: Definire un contesto di crittografia.

Un [contesto di crittografia](#) è un elemento facoltativo, ma fortemente consigliato, delle operazioni crittografiche in AWS Encryption SDK. È possibile definire una o più coppie chiave-valore non segrete.

### Note

Con la versione 4. x of the AWS Encryption SDK per .NET, è possibile richiedere un contesto di crittografia in tutte le richieste di crittografia con il [contesto di crittografia richiesto CMM](#).

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

## Passaggio 5: Creare l'oggetto di input per la crittografia.

Prima di chiamare il `Encrypt()` metodo, create un'istanza della `EncryptInput` classe.

```
string plaintext = File.ReadAllText("C:\\Documents\\CryptoTest\\TestFile.txt");

// Define the encrypt input
var encryptInput = new EncryptInput
{
    Plaintext = plaintext,
    Keyring = keyring,
    EncryptionContext = encryptionContext
};
```

## Fase 6: Crittografia il testo in chiaro.

Usa il `Encrypt()` metodo di AWS Encryption SDK per crittografare il testo in chiaro usando il portachiavi che hai definito.

Il `Encrypt()` metodo `EncryptOutput` restituisce i metodi per ottenere il messaggio crittografato (`Ciphertext`), il contesto di crittografia e la suite di algoritmi.

```
var encryptOutput = encryptionSdk.Encrypt(encryptInput);
```

Passaggio 7: ottieni il messaggio crittografato.

Il `Decrypt()` metodo in per.NET accetta il `Ciphertext` membro dell'`EncryptOutput` istanza. AWS Encryption SDK

Il `Ciphertext` membro dell'`EncryptOutput` oggetto è il [messaggio crittografato](#), un oggetto portatile che include i dati crittografati, le chiavi di dati crittografati e i metadati, incluso il contesto di crittografia. È possibile archiviare in modo sicuro il messaggio crittografato per un periodo di tempo prolungato o inviarlo al `Decrypt()` metodo per recuperare il testo non crittografato.

```
var encryptedMessage = encryptOutput.Ciphertext;
```

## Decrittografia in modalità rigorosa per.NET AWS Encryption SDK

Le migliori pratiche consigliano di specificare le chiavi da utilizzare per decrittografare i dati, un'opzione nota come modalità rigorosa. AWS Encryption SDK utilizza solo le chiavi KMS specificate nel portachiavi per decrittografare il testo cifrato. Le chiavi del portachiavi di decrittografia devono includere almeno una delle chiavi che hanno crittografato i dati.

Questo esempio mostra lo schema di base per la decrittografia in modalità rigorosa con for.NET. AWS Encryption SDK

Fase 1: Creare un'istanza della libreria AWS Encryption SDK e dei fornitori di materiali.

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

Passaggio 2: crea l'oggetto di input per il tuo portachiavi.

Per specificare i parametri per il metodo `keyring`, create un oggetto di input. Ogni metodo `keyring` in per.NET ha un oggetto di input corrispondente. AWS Encryption SDK Poiché questo esempio utilizza il `CreateAwsKmsKeyring()` metodo per creare il portachiavi, crea un'istanza della `CreateAwsKmsKeyringInput` classe per l'input.

In un portachiavi di decrittografia, è necessario utilizzare una chiave ARN per identificare le chiavi 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  
};
```

### Fase 3: Creare il portachiavi.

Per creare il portachiavi di decrittografia, questo esempio utilizza il `CreateAwsKmsKeyring()` metodo e l'oggetto di input del portachiavi.

```
var keyring = materialProviders.CreateAwsKmsKeyring(kmsKeyringInput);
```

### Fase 4: Creare l'oggetto di input per la decrittografia.

Per creare l'oggetto di input per il `Decrypt()` metodo, create un'istanza della classe `DecryptInput`

Il `Ciphertext` parametro del `DecryptInput()` costruttore prende il `Ciphertext` membro dell'`EncryptOutput` oggetto restituito dal metodo. `Encrypt()` La `Ciphertext` proprietà rappresenta il [messaggio crittografato](#), che include i dati crittografati, le chiavi di dati crittografate e i metadati AWS Encryption SDK necessari per decrittografare il messaggio.

Con la versione 4. x di AWS Encryption SDK per .NET, è possibile utilizzare il `EncryptionContext` parametro opzionale per specificare il contesto di crittografia nel `Decrypt()` metodo.

Utilizza il `EncryptionContext` parametro per verificare che il contesto di crittografia utilizzato per crittografare sia incluso nel contesto di crittografia utilizzato per decrittografare il testo cifrato. AWS Encryption SDK aggiunge coppie al contesto di crittografia, inclusa la firma digitale se si utilizza una suite di algoritmi con firma, come la suite di algoritmi predefinita.

```
var encryptedMessage = encryptOutput.Ciphertext;  
  
var decryptInput = new DecryptInput  
{
```

```

    Ciphertext = encryptedMessage,
    Keyring = keyring,
    EncryptionContext = encryptionContext // OPTIONAL
};

```

Passaggio 5: decriptare il testo cifrato.

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

Fase 6: Verificare il contesto di crittografia — Versione 3. x

Il `Decrypt()` metodo della versione 3. x of the AWS Encryption SDK for .NET non utilizza un contesto di crittografia. Ottiene i valori del contesto di crittografia dai metadati del messaggio crittografato. Tuttavia, prima di restituire o utilizzare il testo non crittografato, è consigliabile verificare che il contesto di crittografia utilizzato per decrittografare il testo cifrato includa il contesto di crittografia fornito durante la crittografia.

Verifica che il contesto di crittografia utilizzato per crittografare sia incluso nel contesto di crittografia utilizzato per decrittografare il testo cifrato. AWS Encryption SDK aggiunge coppie al contesto di crittografia, inclusa la firma digitale se utilizzi una suite di algoritmi con firma, come la suite di algoritmi predefinita.

```

// 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");
}

```

## Decrittografia con un portachiavi Discovery in formato.NET AWS Encryption SDK

Invece di specificare le chiavi KMS per la decrittografia, puoi fornire un portachiavi AWS KMS Discovery, ovvero un portachiavi che non specifica alcuna chiave KMS. Un portachiavi di rilevamento consente di AWS Encryption SDK decrittografare i dati utilizzando la chiave KMS che li ha crittografati, a condizione che il chiamante disponga dell'autorizzazione di decrittografia sulla chiave.

Per le migliori pratiche, aggiungi un filtro di rilevamento che limiti le chiavi KMS che possono essere utilizzate a quelle, in particolare, di una partizione specificata. Account AWS

Il portachiavi AWS Encryption SDK per .NET fornisce un portachiavi di Discovery di base che richiede un AWS KMS client e un portachiavi Discovery multiplo che richiede di specificarne uno o più. Regioni AWS Sia il client che le regioni limitano le chiavi KMS che possono essere utilizzate per decrittografare il messaggio crittografato. Gli oggetti di input per entrambi i portachiavi utilizzano il filtro di scoperta consigliato.

L'esempio seguente mostra lo schema per la decrittografia dei dati con un portachiavi AWS KMS Discovery e un Discovery Filter.

Fase 1: Crea un'istanza della libreria AWS Encryption SDK e della libreria dei fornitori di materiali.

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

Passaggio 2: creare l'oggetto di input per il portachiavi.

Per specificare i parametri per il metodo keyring, create un oggetto di input. Ogni metodo keyring in per.NET ha un oggetto di input corrispondente. AWS Encryption SDK Poiché questo esempio utilizza il `CreateAwsKmsDiscoveryKeyring()` metodo per creare il portachiavi, crea un'istanza della `CreateAwsKmsDiscoveryKeyringInput` classe per l'input.

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

var discoveryKeyringInput = new CreateAwsKmsDiscoveryKeyringInput
{
    KmsClient = new AmazonKeyManagementServiceClient(),
    DiscoveryFilter = new DiscoveryFilter()
    {
        AccountIds = accounts,
        Partition = "aws"
    }
};
```

Fase 3: Creare il portachiavi.

Per creare il portachiavi di decrittografia, questo esempio utilizza il `CreateAwsKmsDiscoveryKeyring()` metodo e l'oggetto di input del portachiavi.

```
var discoveryKeyring =  
    materialProviders.CreateAwsKmsDiscoveryKeyring(discoveryKeyringInput);
```

#### Fase 4: Creare l'oggetto di input per la decrittografia.

Per creare l'oggetto di input per il `Decrypt()` metodo, create un'istanza della classe `DecryptInput`. Il valore del `Ciphertext` parametro è il `Ciphertext` membro dell'`EncryptOutput` oggetto restituito dal `Encrypt()` metodo.

Con la versione 4. x di AWS Encryption SDK per .NET, è possibile utilizzare il `EncryptionContext` parametro opzionale per specificare il contesto di crittografia nel `Decrypt()` metodo.

Utilizza il `EncryptionContext` parametro per verificare che il contesto di crittografia utilizzato per crittografare sia incluso nel contesto di crittografia utilizzato per decrittografare il testo cifrato. AWS Encryption SDK aggiunge coppie al contesto di crittografia, inclusa la firma digitale se si utilizza una suite di algoritmi con firma, come la suite di algoritmi predefinita.

```
var ciphertext = encryptOutput.Ciphertext;  
  
var decryptInput = new DecryptInput  
{  
    Ciphertext = ciphertext,  
    Keyring = discoveryKeyring,  
    EncryptionContext = encryptionContext // OPTIONAL  
};  
var decryptOutput = encryptionSdk.Decrypt(decryptInput);
```

#### Fase 5: Verifica del contesto di crittografia — Versione 3. x

Il `Decrypt()` metodo della versione 3. x of the AWS Encryption SDK for .NET non utilizza un contesto di crittografia `Decrypt()`. Ottiene i valori del contesto di crittografia dai metadati del messaggio crittografato. Tuttavia, prima di restituire o utilizzare il testo non crittografato, è consigliabile verificare che il contesto di crittografia utilizzato per decrittografare il testo cifrato includa il contesto di crittografia fornito durante la crittografia.

Verifica che il contesto di crittografia utilizzato per crittografare sia incluso nel contesto di crittografia utilizzato per decrittografare il testo cifrato. AWS Encryption SDK aggiunge coppie al contesto di crittografia, inclusa la firma digitale se utilizzi una suite di algoritmi con firma, come la suite di algoritmi predefinita.

```
// 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");
}
```

## SDK di crittografia AWS per Java

In questo argomento viene descritto come installare e utilizzare SDK di crittografia AWS per Java. Per dettagli sulla programmazione con SDK di crittografia AWS per Java, consulta il [aws-encryption-sdk-java](#) repository su GitHub. Per la documentazione sulle API, consulta [Javadoc](#) per il SDK di crittografia AWS per Java.

### Argomenti

- [Prerequisiti](#)
- [Installazione](#)
- [AWS KMS sporta chiavi in SDK di crittografia AWS per Java](#)
- [Contesti di crittografia richiesti nella versione 3.x](#)
- [Esempi di SDK di crittografia AWS per Java](#)

## Prerequisiti

Prima di installare SDK di crittografia AWS per Java, verifica che siano soddisfatti i prerequisiti indicati di seguito.

### Un ambiente di sviluppo Java

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

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

## Bouncy Castle

Il SDK di crittografia AWS per Java richiede [Bouncy Castle](#).

- SDK di crittografia AWS per Java versioni 1.6.1 e successive utilizzano Bouncy Castle per serializzare e deserializzare gli oggetti crittografici. Puoi usare Bouncy Castle o [Bouncy Castle FIPS](#) per soddisfare questo requisito. Per informazioni sull'installazione e la configurazione di Bouncy Castle FIPS, consulta la [documentazione di BC FIPS](#), in particolare le Guide per l'utente e i PDF della policy di sicurezza.
- Le versioni precedenti del SDK di crittografia AWS per Java utilizzano dell'API di crittografia di Bouncy Castle per Java. Questo requisito è soddisfatto solo da Bouncy Castle non FIPS.

Se non hai Bouncy Castle, vai alle [Ultime release di Bouncy Castle](#) per scaricare il file del provider che corrisponde al tuo JDK. [Puoi anche usare Apache Maven per ottenere l'artefatto per il provider standard di Bouncy Castle \(bcprov-ext-jdk15on\) o l'artefatto per Bouncy Castle FIPS \(bc-fips\)](#).

## AWS SDK for Java

Versione 3. x of the SDK di crittografia AWS per Java richiede AWS SDK for Java 2.x, anche se non si utilizzano AWS KMS portachiavi.

Versione 2. x o precedente di SDK di crittografia AWS per Java non richiede AWS SDK for Java. Tuttavia, AWS SDK for Java è necessario utilizzare [AWS Key Management Service](#) (AWS KMS) come fornitore di chiavi principali. A partire dalla SDK di crittografia AWS per Java versione 2.4.0, SDK di crittografia AWS per Java supporta sia la versione 1.x che la 2.x di AWS SDK for Java. AWS Encryption SDK il codice per AWS SDK for Java 1.x e 2.x è interoperabile. Ad esempio, è possibile crittografare i dati con AWS Encryption SDK codice che supporta AWS SDK for Java 1.x e decrittografarli utilizzando codice che supporta (o viceversa) AWS SDK for Java 2.x. Le versioni precedenti alla 2.4.0 supportano SDK di crittografia AWS per Java solo la versione 1.x. AWS SDK for Java Per informazioni sull'aggiornamento della versione di in uso AWS Encryption SDK, vedere [Migrazione della tua AWS Encryption SDK](#).

Quando aggiorni il SDK di crittografia AWS per Java codice dalla versione AWS SDK for Java 1.x alla versione 2.x AWS SDK for Java, sostituisci i riferimenti all'[AWS KMS interfaccia](#) in AWS SDK for Java 1.x con riferimenti all'[KmsClient interfaccia](#) in AWS SDK for Java 2.x. [Non SDK di crittografia AWS per Java supporta l'interfaccia. KmsAsyncClient](#) Inoltre, aggiorna il codice per utilizzare gli oggetti AWS KMS correlati allo spazio dei nomi kmsdkv2 nel namespace, anziché nel namespace. kms

Per installare AWS SDK for Java utilizza Apache Maven.

- Per [importare l'intero AWS SDK for Java](#) come dipendenza, dichiaralo nel file `pom.xml`.
- Per creare una dipendenza solo per il AWS KMS modulo in AWS SDK for Java 1.x, segui le istruzioni per [specificare](#) moduli particolari e imposta il valore su `artifactId aws-java-sdk-kms`
- [Per creare una dipendenza solo per il AWS KMS modulo in AWS SDK for Java 2.x, segui le istruzioni per specificare moduli particolari.](#) Imposta il `groupId` to `software.amazon.awssdk` e il `artifactId` `kms`

Per ulteriori modifiche, consulta [Cosa c'è di diverso tra AWS SDK for Java 1.x e 2.x](#) nella AWS SDK for Java 2.x Developer Guide.

Gli esempi di Java nella AWS Encryption SDK Developer Guide utilizzano il AWS SDK for Java 2.x

## Installazione

Installa la versione più recente di SDK di crittografia AWS per Java.

### Note

[Tutte le versioni SDK di crittografia AWS per Java precedenti alla 2.0.0 sono in fase di sviluppo. end-of-support](#)

È possibile eseguire l'aggiornamento in tutta sicurezza dalla versione 2.0. x e versioni successive alla versione più recente di SDK di crittografia AWS per Java senza modifiche al codice o ai dati. Tuttavia, nella versione 2.0 sono state introdotte [nuove funzionalità di sicurezza](#). x non sono retrocompatibili. Per eseguire l'aggiornamento da versioni precedenti alla 1.7. x alla versione 2.0. x e versioni successive, è necessario prima eseguire l'aggiornamento alla versione più recente 1. versione x di AWS Encryption SDK. Per informazioni dettagliate, consultare [Migrazione della tua AWS Encryption SDK](#).

Puoi installare il SDK di crittografia AWS per Java in uno dei seguenti modi:

### Manualmente

Per installare SDK di crittografia AWS per Java, clona o scarica il [aws-encryption-sdk-java](#) GitHub repository.

## Utilizzo di Apache Maven

Il SDK di crittografia AWS per Java è disponibile tramite [Apache Maven](#) con la seguente definizione di dipendenza.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-encryption-sdk-java</artifactId>
  <version>3.0.0</version>
</dependency>
```

[Dopo aver installato l'SDK, inizia a guardare il codice Java di esempio in questa guida e Javadoc attivo. GitHub](#)

## AWS KMSportachiavi in SDK di crittografia AWS per Java

Versione 3. x of the SDK di crittografia AWS per Java utilizza i [portachiavi per eseguire la crittografia delle buste](#). I AWS KMS portachiavi di base del KMS SDK di crittografia AWS per Java accettano solo una chiave KMS. Richiedono anche un AWS KMS client, che ti dà l'opportunità di configurare il client per la chiave Regione AWS KMS.

Per creare un AWS KMS portachiavi con una o più chiavi avvolgenti, usa un portachiavi multiplo. SDK di crittografia AWS per JavaHa uno speciale portachiavi multiplo che accetta una o più AWS KMS chiavi e un portachiavi multiplo standard che accetta uno o più portachiavi di qualsiasi tipo supportato. Alcuni programmatori preferiscono utilizzare un metodo con più portachiavi per creare tutti i loro portachiavi e supportano questa strategia. SDK di crittografia AWS per Java

SDK di crittografia AWS per Java[Fornisce portachiavi di base a chiave singola e portachiavi multipla per tutti i casi d'uso tipici, comprese le chiavi multiregionali. AWS KMS](#)

Ad esempio, per creare un AWS KMS portachiavi con una sola AWS KMS chiave, è possibile utilizzare il metodo]. 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);
```

Per creare un portachiavi con una o più AWS KMS chiavi, utilizzate il `CreateAwsKmsMultiKeyring()` metodo. Questo esempio utilizza due chiavi KMS. Per specificare una chiave KMS, usa solo il `generator` parametro. Il `msKeyIds` parametro che specifica le chiavi KMS aggiuntive è facoltativo.

L'input per questo portachiavi non richiede un client. AWS KMS AWS Encryption SDK utilizza invece il AWS KMS client predefinito per ogni regione rappresentato da una chiave KMS nel portachiavi. Ad esempio, se la chiave KMS identificata dal valore del `Generator` parametro si trova nella regione degli Stati Uniti occidentali (Oregon) (`us-west-2`), AWS Encryption SDK crea un AWS KMS client predefinito per la regione. `us-west-2` Se è necessario personalizzare il AWS KMS client, utilizzare il `CreateAwsKmsKeyring()` metodo.

```
// 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);
```

SDK di crittografia AWS per Java supporta AWS KMS portachiavi che utilizzano la crittografia simmetrica (`SYMMETRIC_DEFAULT`) o le chiavi RSA KMS asimmetriche. AWS KMSi portachiavi creati con chiavi RSA KMS asimmetriche possono contenere solo una coppia di chiavi.

Per crittografare con un AWS KMS portachiavi RSA asimmetrico, non hai bisogno di [kms:GenerateDataKey](#) o [KMS:Encrypt](#) perché devi specificare il materiale della chiave pubblica che desideri utilizzare per la crittografia quando crei il portachiavi. Non vengono effettuate chiamate durante la crittografia con questo portachiavi. AWS KMS [Per decrittografare con un portachiavi AWS KMS RSA asimmetrico](#), è necessaria l'autorizzazione [KMS:Decrypt](#).

Per creare un AWS KMS portachiavi RSA asimmetrico, devi fornire la chiave pubblica e la chiave privata ARN della tua chiave RSA KMS asimmetrica. La chiave pubblica deve essere codificata in PEM. L'esempio seguente crea un AWS KMS portachiavi con una coppia di chiavi RSA asimmetrica.

```
// 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);
```

## Contesti di crittografia richiesti nella versione 3.x

Con la versione 3. x di SDK di crittografia AWS per Java, è possibile utilizzare il contesto di crittografia richiesto CMM per richiedere [contesti di crittografia nelle operazioni](#) crittografiche. Un contesto di crittografia è un insieme di coppie chiave-valore non segrete. Il contesto di crittografia è associato crittograficamente ai dati crittografati in modo che sia necessario lo stesso contesto di crittografia per decrittografare il campo. Quando si utilizza il contesto di crittografia richiesto CMM, è possibile specificare una o più chiavi di contesto di crittografia richieste (chiavi obbligatorie) che devono essere incluse in tutte le chiamate di crittografia e decrittografia.

### Note

Il contesto di crittografia richiesto CMM è interoperabile solo con la versione 4. x del per .NET AWS Encryption SDK. Non è interoperabile con nessun'altra implementazione del linguaggio di programmazione. Se si crittografano i dati utilizzando il contesto di crittografia richiesto CMM, è possibile decrittografarli solo con la versione 3. x della versione o 4. SDK di crittografia AWS per Java x del AWS Encryption SDK per.NET.

In encrypt, AWS Encryption SDK verifica che tutte le chiavi del contesto di crittografia richieste siano incluse nel contesto di crittografia specificato. Segna AWS Encryption SDK i contesti di crittografia specificati. Solo le coppie chiave-valore che non sono chiavi richieste vengono serializzate e archiviate in testo semplice nell'intestazione del messaggio crittografato restituito dall'operazione di crittografia.

In decrypt, è necessario fornire un contesto di crittografia che contenga tutte le coppie chiave-valore che rappresentano le chiavi richieste. AWS Encryption SDK utilizza questo contesto di crittografia e le coppie chiave-valore memorizzate nell'intestazione del messaggio crittografato per ricostruire il contesto di crittografia originale specificato nell'operazione di crittografia. Se AWS Encryption SDK non è possibile ricostruire il contesto di crittografia originale, l'operazione di decrittografia ha esito negativo. Se si fornisce una coppia chiave-valore che contiene la chiave richiesta con un valore errato, il messaggio crittografato non può essere decrittografato. È necessario fornire la stessa coppia chiave-valore specificata in encrypt.

**⚠ Important**

Valuta attentamente i valori che scegli per le chiavi richieste nel tuo contesto di crittografia. Devi essere in grado di fornire nuovamente le stesse chiavi e i valori corrispondenti al momento di decrypt. Se non riesci a riprodurre le chiavi richieste, il messaggio crittografato non può essere decrittografato.

L'esempio seguente inizializza un AWS KMS portachiavi con il contesto di crittografia richiesto CMM.

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

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

// Create a list of required encryption contexts
final List<String> requiredEncryptionContextKeys = Arrays.asList("encryption",
    "context");

// Create the keyring
final MaterialProviders materialProviders = MaterialProviders.builder()
    .MaterialProvidersConfig(MaterialProvidersConfig.builder().build())
    .build();
final CreateAwsKmsKeyringInput keyringInput = CreateAwsKmsKeyringInput.builder()
    .kmsKeyId(keyArn)
    .kmsClient(KmsClient.create())
    .build();
IKeyring kmsKeyring = materialProviders.CreateAwsKmsKeyring(keyringInput);

// Create the required encryption context CMM
ICryptographicMaterialsManager cmm =
    materialProviders.CreateDefaultCryptographicMaterialsManager(
        CreateDefaultCryptographicMaterialsManagerInput.builder()
            .keyring(kmsKeyring)
```

```
        .build()
    );
    ICryptographicMaterialsManager requiredCMM =
        materialProviders.CreateRequiredEncryptionContextCMM(
            CreateRequiredEncryptionContextCMMInput.builder()
                .requiredEncryptionContextKeys(requiredEncryptionContextKeys)
                .underlyingCMM(cmm)
                .build()
        );
```

## Esempi di SDK di crittografia AWS per Java

Gli esempi seguenti illustrano come utilizzare SDK di crittografia AWS per Java per crittografare e decrittare i dati. Questi esempi mostrano come utilizzare la versione 3. x e versioni successive di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java sostituisce i [fornitori di chiavi principali](#) con [portachiavi](#). Per gli esempi che utilizzano versioni precedenti, trova la tua versione nell'elenco delle [versioni](#) del [aws-encryption-sdk-java](#) repository su. GitHub

### Argomenti

- [Crittografia e decrittazione di stringhe](#)
- [Crittografia e decrittazione di flussi di byte](#)
- [Crittografia e decrittografia di flussi di byte con un portachiavi multiplo](#)

## Crittografia e decrittazione di stringhe

L'esempio seguente mostra come utilizzare la versione 3. x delle stringhe SDK di crittografia AWS per Java per crittografare e decrittografare. Prima di utilizzare la stringa, convertirla in un matrice di byte.

[Questo esempio utilizza un portachiavi. AWS KMS](#) Quando si esegue la crittografia con un AWS KMS portachiavi, è possibile utilizzare un ID chiave, un ARN della chiave, un nome alias o un alias ARN per identificare le chiavi KMS. Durante la decrittografia, è necessario utilizzare una chiave ARN per identificare le chiavi KMS.

Quando chiami il metodo `encryptData()` viene restituito un [messaggio crittografato](#) (`CryptoResult`) che include il testo cifrato, le chiavi dati crittografate e il contesto di crittografia. Quando chiama `getResult` sull'oggetto `CryptoResult`, viene restituita una versione di stringa codificata in base 64 del [messaggio crittografato](#) che puoi passare al metodo `decryptData()`.

Allo stesso modo, quando si chiama `decryptData()`, l'`CryptoResult` oggetto restituito contiene il messaggio in chiaro e un ID. AWS KMS key Prima che l'applicazione restituisce il testo normale, verifica che l'ID della AWS KMS key e il contesto di crittografia nel messaggio crittografato siano quelli previsti.

```
// 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:</p>
 *
 * <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);
}
}

```

## Crittografia e decrittazione di flussi di byte

L'esempio seguente illustra come utilizzare AWS Encryption SDK per crittografare e decrittare i flussi di byte.

Questo esempio utilizza un portachiavi [Raw AES](#).

Durante la crittografia, questo esempio utilizza il `AwsCrypto.builder().withEncryptionAlgorithm()` metodo per specificare una suite di algoritmi senza firme [digitali](#). Durante la decrittografia, per garantire che il testo cifrato non sia firmato, questo esempio utilizza il metodo `createUnsignedMessageDecryptingStream()`. Il `createUnsignedMessageDecryptingStream()` metodo fallisce se incontra un testo cifrato con una firma digitale.

Se state eseguendo la crittografia con la suite di algoritmi predefinita, che include le firme digitali, utilizzate invece il `createDecryptingStream()` metodo, come illustrato nell'esempio seguente.

```

// 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("Example
{
        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");
}
}

```

## Crittografia e decrittografia di flussi di byte con un portachiavi multiplo

[L'esempio seguente mostra come utilizzare il AWS Encryption SDK con un portachiavi multiplo.](#)

Quando utilizzi un keyring multiplo per crittografare i dati, questi possono essere decrittati con le chiavi di wrapping contenute in qualsiasi keyring. Questo esempio utilizza un [AWS KMSportachiavi e un portachiavi Raw RSA come portachiavi secondari](#).

[Questo esempio esegue la crittografia con la suite di algoritmi predefinita, che include una firma digitale.](#) Durante lo streaming, AWS Encryption SDK rilascia il testo non crittografato dopo i controlli di integrità, ma prima di aver verificato la firma digitale. Per evitare di utilizzare il testo non crittografato fino alla verifica della firma, questo esempio memorizza nel buffer il testo semplice e lo scrive su disco solo quando la decrittografia e la verifica sono complete.

```
// 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());
    }
}
```

## SDK di crittografia AWS per JavaScript

La SDK di crittografia AWS per JavaScript è progettato per fornire una libreria di crittografia lato client per gli sviluppatori che scrivono applicazioni browser WebJavaScripto applicazioni server Web in Node.js.

Come per tutte le implementazioni di AWS Encryption SDK, SDK di crittografia AWS per JavaScript offre le caratteristiche di protezione dei dati avanzate. Queste includono la [crittografia envelope](#), [dati autenticati aggiuntivi](#) (AAD) e [suite di algoritmi](#) di chiavi simmetriche, autenticate e sicure, come, ad esempio, AES-GCM a 256 bit con derivazione della chiave e firma.

Tutte le implementazioni specifiche del linguaggio di AWS Encryption SDK sono progettate per essere interoperabili, fatte salve le limitazioni del linguaggio stesso. Per ulteriori informazioni sui vincoli di linguaggio per JavaScript, vedere [the section called “Compatibilità”](#).

Ulteriori informazioni

- Per informazioni dettagliate SDK di crittografia AWS per JavaScript, consulta [aws-encryption-sdk-javascript](#) repository GitHub.
- Per esempi di programmazione, vedere [the section called “Esempi”](#) e [example-browser](#) [example-node](#) moduli in [aws-encryption-sdk-javascript](#) repository.
- Per un esempio reale di utilizzo SDK di crittografia AWS per JavaScript per crittografare i dati in un'applicazione Web, vedere [Come abilitare la crittografia in un browser con SDK di crittografia AWS per JavaScript Node.js](#) nella AWS Blog sulla sicurezza.

## Argomenti

- [Compatibilità del SDK di crittografia AWS per JavaScript](#)
- [Installazione di SDK di crittografia AWS per JavaScript](#)
- [Moduli in SDK di crittografia AWS per JavaScript](#)
- [SDK di crittografia AWS per JavaScript Esempi di](#)

## Compatibilità del SDK di crittografia AWS per JavaScript

SDK di crittografia AWS per JavaScript è progettato per essere interoperabile con altre implementazioni di linguaggio di AWS Encryption SDK. Nella maggior parte dei casi, è possibile crittografare i dati con SDK di crittografia AWS per JavaScript e decrittarli con qualsiasi altra implementazione di linguaggio, inclusa l'[interfaccia a riga di comando AWS Encryption SDK](#). È inoltre possibile utilizzare SDK di crittografia AWS per JavaScript per decrittare [i messaggi crittografati](#) prodotti da altre implementazioni linguistiche di AWS Encryption SDK.

Tuttavia, quando si utilizza SDK di crittografia AWS per JavaScript, è necessario essere consapevoli di alcuni problemi di compatibilità nell'implementazione del linguaggio JavaScript e nei browser Web.

Inoltre, quando si utilizzano implementazioni linguistiche diverse, assicurarsi di configurare provider di chiavi master compatibili, le chiavi master e i keyring compatibili. Per dettagli, consultare [Compatibilità dei keyring](#).

## Compatibilità SDK di crittografia AWS per JavaScript

L'implementazione JavaScript di AWS Encryption SDK differisce da altre implementazioni di linguaggio nei seguenti modi:

- L'operazione di crittografia di SDK di crittografia AWS per JavaScript non restituisce testo cifrato non framed. Tuttavia, SDK di crittografia AWS per JavaScript decrittterà il testo cifrato framed e non framed restituito da altre implementazioni del linguaggio di AWS Encryption SDK.
- A partire da Node.js versione 12.9.0, Node.js supporta le seguenti opzioni di wrapping chiave RSA:
  - OAEP con SHA1, SHA256, SHA384 o SHA512
  - OAEP con SHA1 e MGF1 con SHA1
  - PKCS1v15
- Prima della versione 12.9.0, Node.js supporta solo le seguenti opzioni di wrapping della chiave RSA:

- OAEP con SHA1 e MGF1 con SHA1
- PKCS1v15

## Compatibilità browser

Alcuni browser Web non supportano le operazioni di crittografia di base richieste da SDK di crittografia AWS per JavaScript. È possibile compensare alcune delle operazioni mancanti configurando un fallback per WebCryptoAPI implementata dal browser.

### Limitazioni del browser Web

Le seguenti limitazioni sono comuni a tutti i browser Web:

- La WebCryptoAPI non supporta il wrapping delle chiavi PKCS1v15.
- I browser non supportano chiavi a 192 bit.

### Operazioni crittografiche richieste

SDK di crittografia AWS per JavaScript richiede le seguenti operazioni nei browser Web. Se un browser non supporta queste operazioni, non è compatibile con SDK di crittografia AWS per JavaScript.

- Il browser deve includere `crypto.getRandomValues()`, che è un metodo per generare valori crittograficamente casuali. Per informazioni sulle versioni del browser Web che supportano `crypto.getRandomValues()`, consulta [Posso utilizzare la crittografia.getRandomValues\(\)?](#)

### Fallback richiesto

SDK di crittografia AWS per JavaScript richiede le seguenti librerie e operazioni nei browser Web. Se si supporta un browser Web che non soddisfa questi requisiti, è necessario configurare un fallback. Altrimenti, i tentativi di utilizzare il SDK di crittografia AWS per JavaScript con il browser falliranno.

- La WebCryptoAPI, che esegue operazioni di crittografia di base nelle applicazioni Web, non è disponibile per tutti i browser. Per informazioni sulle versioni del browser Web che supportano la crittografia Web, consulta [Posso utilizzare la crittografia Web?](#)
- Le versioni moderne del browser Web Safari non supportano la crittografia AES-GCM di zero byte, che AWS Encryption SDK richiede. Se il browser implementa il WebCryptoAPI, ma non può

utilizzare AES-GCM per crittografare zero byte, SDK di crittografia AWS per JavaScript utilizza la libreria di fallback solo per la crittografia a byte zero. Utilizza il WebCryptoAPI per tutte le altre operazioni.

Per configurare un fallback per entrambe le limitazioni, aggiungere le istruzioni seguenti al codice. Nella funzione [configureFallback](#) specificare una libreria che supporti le funzionalità mancanti. Nell'esempio seguente viene utilizzata la ricerca Microsoft.JavaScriptLibreria di crittografia (`msCrypto`), ma è possibile sostituirla con una libreria compatibile. Per un esempio completo, consulta [fallback.ts](#).

```
import { configureFallback } from '@aws-crypto/client-browser'  
configureFallback(msCrypto)
```

## Installazione di SDK di crittografia AWS per JavaScript

SDK di crittografia AWS per JavaScript consiste in una raccolta di moduli interdipendenti. Molti dei moduli sono solo raccolte di moduli progettati per lavorare insieme. Alcuni moduli sono progettati per funzionare in modo indipendente. Alcuni moduli sono necessari per tutte le implementazioni; alcuni altri sono necessari solo per casi speciali. Per informazioni sui moduli del AWS Encryption SDK per JavaScript, consulta [Moduli in SDK di crittografia AWS per JavaScript](#) e il `README.md` file in ciascuno dei moduli del [aws-encryption-sdk-javascript](#) archivio su GitHub.

### Note

Tutte le versioni di SDK di crittografia AWS per JavaScript versioni precedenti alla versione 2.0.0 sono nella [end-of-support fase](#).

È possibile eseguire l'aggiornamento dalla versione 2.0.xe successivamente alla versione più recente di SDK di crittografia AWS per JavaScript senza modifiche al codice o ai dati.

Tuttavia, [nuove funzioni di sicurezza](#) introdotto nella versione 2.0.x non sono compatibili con le versioni precedenti. Per eseguire l'aggiornamento da versioni precedenti alla 1.7.x alla versione 2.0.xe successivamente, è necessario eseguire l'aggiornamento alla versione più recente della versione 1.x versione del SDK di crittografia AWS per JavaScript. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Per installare i moduli, utilizzare il [gestore di pacchetti npm](#).

Ad esempio, per installare il modulo `client-node`, che include tutti i moduli che è necessario programmare con SDK di crittografia AWS per JavaScript in Node.js, utilizzare il comando seguente.

```
npm install @aws-crypto/client-node
```

Per installare il modulo `client-browser`, che include tutti i moduli che è necessario programmare con SDK di crittografia AWS per JavaScript nel browser, utilizzare il seguente comando.

```
npm install @aws-crypto/client-browser
```

Per esempi funzionanti di utilizzo del SDK di crittografia AWS per JavaScript, guarda gli esempi nella `example-node` e `example-browser` moduli nel [aws-encryption-sdk-javascript](#) archivio su GitHub.

## Moduli in SDK di crittografia AWS per JavaScript

I moduli in SDK di crittografia AWS per JavaScript semplificano l'installazione del codice di cui hai bisogno per i tuoi progetti.

### Moduli per JavaScript Node.js

#### [client-node](#)

Include tutti i moduli necessari per programmare con SDK di crittografia AWS per JavaScript in Node.js.

#### [caching-materials-manager-Node](#)

Esporta funzioni che supportano la funzionalità di [memorizzazione nella cache della chiave dati](#) in SDK di crittografia AWS per JavaScript in Node.js.

#### [decrypt-node](#)

Esporta le funzioni che decrittano e verificano i messaggi crittografati che rappresentano dati e flussi di dati. Includi nel modulo `client-node`.

#### [encrypt-node](#)

Esporta funzioni che crittografano e firmano diversi tipi di dati. Includi nel modulo `client-node`.

#### [example-node](#)

Esporta esempi di lavoro di programmazione con SDK di crittografia AWS per JavaScript in Node.js. Include esempio di diversi tipi di keyring e diversi tipi di dati.

## [hkdf-node](#)

Esporta una [funzione di derivazione chiave basata su HMAC](#) (HKDF) che SDK di crittografia AWS per JavaScript in Node.js utilizza in particolari suite di algoritmi. La SDK di crittografia AWS per JavaScript nel browser, utilizza la funzione HKDF nativa nel WebCryptoAPI.

## [integration-node](#)

Definisce i test che verificano che SDK di crittografia AWS per JavaScript in Node.js sia compatibile con altre implementazioni di linguaggio di AWS Encryption SDK.

## [kms-keyring-node](#)

Esporta funzioni che supportano AWS KMSkeyring in Node.js.

## [raw-aes-keyring-Node](#)

Esporta funzioni che supportano i [keyring AES Raw](#) in Node.js.

## [raw-rsa-keyring-Node](#)

Esporta funzioni che supportano i [keyring RSA Raw](#) in Node.js.

## Moduli per JavaScript Browser

### [client-browser](#)

Include tutti i moduli che è necessario programmare con SDK di crittografia AWS per JavaScript nel browser.

### [caching-materials-managerBrowser](#)

Esporta le funzioni che supportano la funzione di [memorizzazione nella cache della chiave dati](#) per JavaScript nel browser.

### [decrypt-browser](#)

Esporta le funzioni che decrittano e verificano i messaggi crittografati che rappresentano dati e flussi di dati.

### [encrypt-browser](#)

Esporta funzioni che crittografano e firmano diversi tipi di dati.

## [example-browser](#)

Esempi di lavoro di programmazione con SDK di crittografia AWS per JavaScript nel browser. Include esempi di diversi tipi di keyring e diversi tipi di dati.

## [integration-browser](#)

Definisce i test che verificano che lo script SDK di crittografia AWS per Java nel browser sia compatibile con altre implementazioni di linguaggio di AWS Encryption SDK.

## [kms-keyring-browser](#)

Esporta funzioni che supportano [AWS KMS Prtachiavi](#) nel browser.

## [raw-aes-keyringBrowser](#)

Esporta le funzioni che supportano i [keyring AES Raw](#) nel browser.

## [raw-rsa-keyringBrowser](#)

Esporta le funzioni che supportano i [keyring RSA Raw](#) nel browser.

## Moduli per tutte le implementazioni

### [cache-material](#)

Supporta la funzione di [memorizzazione nella cache della chiave dati](#). Fornisce il codice per l'assemblaggio dei materiali crittografici memorizzati nella cache con ogni chiave di dati.

### [kms-keyring](#)

Esporta le funzioni che supportano i [keyring KMS](#).

### [material-management](#)

Implementa il [gestore dei materiali crittografici](#) (CMM).

### [raw-keyring](#)

Esporta le funzioni necessarie per i keyring AES e RSA non elaborati.

### [serialize](#)

Esporta le funzioni che l'SDK utilizza per serializzare il suo output.

### [web-crypto-backend](#)

Esporta le funzioni che utilizzano WebCryptoAPI nel SDK di crittografia AWS per JavaScript nel browser.

## SDK di crittografia AWS per JavaScript Esempi di

Gli esempi seguenti illustrano come utilizzare SDK di crittografia AWS per JavaScript per crittografare e decrittare i dati.

Puoi trovare ulteriori esempi di utilizzo SDK di crittografia AWS per JavaScript nella [example-node](#) e [example-browser](#) moduli nella [aws-encryption-sdkJavaScript](#) repository su GitHub. Questi moduli di esempio non vengono installati quando installi i moduli `client-browser` o `client-node`.

Consulta gli esempi di codice completi: Nodo: [kms\\_simple.ts](#), Browser: [kms\\_simple.ts](#)

### Argomenti

- [Crittografia dei dati con unAWS KMSKeyring](#)
- [Decrittazione dei dati con unAWS KMSKeyring](#)

### Crittografia dei dati con unAWS KMSKeyring

Nell'esempio seguente viene illustrato come utilizzare SDK di crittografia AWS per JavaScript per crittografare e decrittare una stringa breve o una matrice di byte.

Questo esempio presenta un [AWS KMSKeyring](#), un tipo di portachiavi che utilizza unAWS KMS key per generare e crittografare le chiavi dati. Per aiuto nella creazione di unAWS KMS key, consulta [Creazione di chiavi](#) nella AWS Key Management Service Guida per gli sviluppatori. Per aiuto nell'identificazione delAWS KMS keys in unAWS KMSKeyring, vedi [Identificazione di AWS KMS keys in un keyring AWS KMS](#)

Fase 1: Costruisci il keyring.

Creazione di unAWS KMSkeyring per la crittografia.

Quando si crittografa con unAWS KMSportachiavi, è necessario specificare un generatore di chiavi, cioè unAWS KMS key che viene utilizzato per generare la chiave di dati di testo normale e crittografarla. Inoltre puoi specificare zero o più chiavi aggiuntive per crittografare la stessa chiave di dati di testo normale. Il keyring restituisce la chiave di dati di testo normale e una copia crittografata di tale chiave di dati per ciascuna AWS KMS key nel keyring, inclusa la chiave generatore. Per decrittare i dati, è necessario decrittare una qualsiasi delle chiavi di dati crittografate.

Per specificare le AWS KMS keys per un keyring di crittografia in SDK di crittografia AWS per JavaScript, puoi utilizzare [qualsiasi identificatore di chiave AWS KMS supportato](#). In questo

esempio viene utilizzata una chiave generatore, identificata dal relativo [ARN di alias](#) e una chiave aggiuntiva, identificata da un [ARN di chiave](#).

#### Note

Se hai intenzione di riutilizzare il tuo AWS KMS Keyring per la decrittazione, devi utilizzare i codici ARN di chiave per identificare AWS KMS keys nel keyring.

Prima di eseguire questo codice, sostituisci gli identificatori di AWS KMS key di esempio con identificatori validi. Devi disporre delle [autorizzazioni necessarie per utilizzare le AWS KMS keys](#) nel keyring.

#### JavaScript Browser

Inizia fornendo le credenziali nel browser. La SDK di crittografia AWS per JavaScript esempi di utilizzano [webpack.DefinePlugin](#) che sostituisce le costanti delle credenziali con le credenziali effettive. Tuttavia puoi utilizzare qualsiasi metodo per fornire le credenziali. Quindi, utilizza le credenziali per creare un client AWS KMS.

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

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

Successivamente, specificare il AWS KMS keys per la chiave generatore e la chiave aggiuntiva. Quindi, crea un AWS KMS Keyring che utilizza AWS KMS client e il 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 })
```

### Fase 2: Impostazione del contesto di crittografia.

Un [contesto di crittografia](#) è rappresentato da tipi di dati autenticati aggiuntivi arbitrari e non segreti. Quando fornisci un contesto durante la crittografia, l'AWS Encryption SDK lo vincola al testo cifrato a livello crittografico, per cui dovrai utilizzare lo stesso contesto di crittografia per decrittare i dati. L'utilizzo di un contesto di crittografia è facoltativo, ma viene consigliato come best practice.

Crea un oggetto semplice che includa le coppie di contesto di crittografia. La chiave e il valore di ogni coppia devono essere una stringa.

## 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'  
}
```

### Fase 3: Crittografia i dati.

Per crittografare i dati di testo normale, chiama la funzione `encrypt`. Passa nelAWS KMSkeyring, i dati di testo normale e il contesto di crittografia.

La funzione `encrypt` restituisce un [messaggio crittografato](#) (`result`) che contiene i dati crittografati, le chiavi di dati crittografate e i metadati importanti, inclusi il contesto di crittografia e la firma.

Puoi [decriptare questo messaggio crittografato](#) utilizzando AWS Encryption SDK per qualsiasi linguaggio di programmazione supportato.

#### 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 })
```

## Decrittazione dei dati con unAWS KMSKeyring

Puoi utilizzare SDK di crittografia AWS per JavaScript per decrittare il messaggio crittografato e recuperare i dati originali.

In questo esempio, esegui la decrittazione dei dati crittografati nell'esempio [the section called “Crittografia dei dati con unAWS KMSKeyring”](#).

Fase 1: Costruisci il keyring.

Per decrittare i dati, passa il [messaggio crittografato](#) (`result`) restituito dalla funzione `encrypt`. Il messaggio crittografato include i dati crittografati, le chiavi di dati crittografate e i metadati importanti, inclusi il contesto di crittografia e la firma.

È inoltre necessario specificare un [AWS KMSKeyring](#) durante la decrittografia. Puoi utilizzare lo stesso keyring usato per crittografare i dati o un keyring diverso. Per restituire l'esito positivo, almeno una AWS KMS key nel keyring di decrittazione deve essere in grado di decrittare una delle chiavi di dati crittografate nel messaggio crittografato. Poiché non vengono generate chiavi di dati, non è necessario specificare una chiave generatore in un keyring di decrittazione. Se la specifichi, la chiave generatore e le chiavi aggiuntive vengono trattate allo stesso modo.

Per specificare unAWS KMS keyper un portachiavi di decrittografia nelSDK di crittografia AWS per JavaScript, devi utilizzare[ARN della chiave](#). In caso contrario, la AWS KMS key non viene riconosciuta. Per aiuto nell'identificazione delAWS KMS keysin unAWS KMSKeyring, vedi[Identificazione di AWS KMS keys in un keyring AWS KMS](#)

### Note

Se usi lo stesso keyring per crittografare e decrittare, usa gli ARN di chiave per identificare le AWS KMS keys nel keyring.

In questo esempio, viene creato un keyring che include solo una delle AWS KMS keys nel keyring di crittografia. Prima di eseguire questo codice, sostituisci l'ARN della chiave di esempio con uno valido. Devi disporre dell'autorizzazione `kms:Decrypt` per la AWS KMS key.

### JavaScript Browser

Inizia fornendo le credenziali nel browser. LaSDK di crittografia AWS per JavaScriptesempi di utilizzano[webpack.DefinePlugin](#)che sostituisce le costanti delle credenziali con le credenziali effettive. Tuttavia puoi utilizzare qualsiasi metodo per fornire le credenziali. Quindi, utilizza le credenziali per creare un client AWS KMS.

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

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

Quindi, crea unAWS KMSKeyring che utilizzaAWS KMScliente. In questo esempio viene utilizzata solo una delle AWS KMS keys del keyring di crittografia.

```
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 })
```

### Fase 2: Decrittografare i dati.

A questo punto, chiama la funzione `decrypt`. Passa il keyring di decrittazione appena creato (`keyring`) e il [messaggio crittografato](#) restituito dalla funzione `encrypt` (`result`). AWS Encryption SDK utilizza il keyring per decrittare una delle chiavi di dati crittografate. Quindi, utilizza la chiave di dati in testo normale per decrittare i dati.

Se la chiamata ha esito positivo, il campo `plaintext` contiene i dati di testo normale (decrittati). Il campo `messageHeader` contiene i metadati relativi al processo di decrittazione, incluso il contesto di crittografia utilizzato per decrittare i dati.

## JavaScript Browser

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

## JavaScript Node.js

```
const { plaintext, messageHeader } = await decrypt(keyring, result)
```

### Fase 3: Verifica il contesto di crittografia.

Il [contesto di crittografia](#) utilizzato per decrittare i dati è incluso nell'intestazione del messaggio (`messageHeader`) restituita dalla funzione `decrypt`. Prima che l'applicazione restituisca i dati di testo normale, verifica che il contesto di crittografia fornito durante la crittografia sia incluso nel contesto di crittografia utilizzato per la decrittazione. Una mancata corrispondenza potrebbe indicare che i dati sono stati manomessi o che non è stato decrittato il testo crittografato corretto.

Quando si verifica il contesto di crittografia, non è necessaria una corrispondenza esatta. Quando usi un algoritmo di crittografia con la firma, il [responsabile di materiali crittografici](#) (CMM) aggiunge la chiave di firma pubblica al contesto di crittografia prima di crittografare il messaggio. Tuttavia, tutte le coppie di contesto di crittografia inviate devono essere incluse nel contesto di crittografia restituito.

Innanzitutto, ottieni il contesto di crittografia dall'intestazione del messaggio. Verifica quindi che ogni coppia chiave-valore nel contesto di crittografia originale (`context`) corrisponda a una coppia chiave-valore nel contesto di crittografia restituito (`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 il controllo del contesto di crittografia ha esito positivo, puoi restituire i dati di testo normale.

## SDK di crittografia AWS per Python

In questo argomento viene descritto come installare e utilizzare SDK di crittografia AWS per Python. Per informazioni dettagliate sulla programmazione con SDK di crittografia AWS per Python, consulta [aws-encryption-sdk-python](#) archivio su GitHub. Per la documentazione sulle API, consulta [Leggi i documenti](#).

### Argomenti

- [Prerequisiti](#)
- [Installazione](#)
- [Codice di esempio per SDK di crittografia AWS per Python](#)

## Prerequisiti

Prima di installare SDK di crittografia AWS per Python, verifica che siano soddisfatti i prerequisiti indicati di seguito.

### Una versione supportata di Python

Python 3.6 o versioni successive sono richieste da SDK di crittografia AWS per Python versioni 3.1.0 e successive.

Versioni precedenti di AWS Encryption SDK supporta Python 2.7 e Python 3.4 e versioni successive, ma si consiglia di utilizzare la versione più recente di AWS Encryption SDK.

Per scaricare Python, consulta la pagina relativa ai [download di Python](#).

### Lo strumento di installazione pip per Python

pip è incluso in Python 3.6 e versioni successive, anche se potresti volerlo aggiornare. Per ulteriori informazioni sull'aggiornamento o l'installazione pip, consulta [Installazione](#) nel pip documentazione.

## Installazione

Installazione della versione più recente di SDK di crittografia AWS per Python.

### Note

Tutte le versioni di SDK di crittografia AWS per Python le versioni precedenti alla 3.0.0 sono nella [end-of-support fase](#).

Puoi eseguire l'aggiornamento in sicurezza dalla versione 2.0.xe successivamente alla versione più recente di AWS Encryption SDK senza modifiche al codice o ai dati.

Tuttavia, [nuove funzioni di sicurezza](#) introdotto nella versione 2.0.x non sono compatibili con le versioni precedenti. Per eseguire l'aggiornamento da versioni precedenti alla 1.7.x alla versione 2.0.xe successivamente, è necessario eseguire l'aggiornamento alla versione più recente 1.x versione del AWS Encryption SDK. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Utilizzo di pip per installare SDK di crittografia AWS per Python, come mostrato nei seguenti esempi.

Per installare la versione più recente

```
pip install aws-encryption-sdk
```

Per ulteriori dettagli sull'utilizzo di pip per installare e aggiornare pacchetti, consulta la sezione relativa all'[installazione dei pacchetti](#).

Il SDK di crittografia AWS per Python richiede la [libreria di crittografia](#) (pyca/cryptography) su tutte le piattaforme. Tutte le versioni di pip installa e crea automaticamente il cryptography libreria su Windows. pip 8.1 e versioni successive: installazione e creazione automatica cryptography su Linux. Se si utilizza una versione precedente di pip il tuo ambiente Linux non dispone degli strumenti necessari per creare il cryptography libreria, è necessario installare Maven: Per ulteriori informazioni, consulta la sezione relativa alla [creazione di una crittografia in Linux](#).

Versioni 1.10.0 e 2.5.0 di SDK di crittografia AWS per Python pinn [crittografia](#) dipendenza tra 2.5.0 e 3.3.2. Altre versioni di SDK di crittografia AWS per Python installare la versione più recente della crittografia. Se è necessaria una versione della crittografia successiva alla 3.3.2, consigliamo di utilizzare il rilascio principale di SDK di crittografia AWS per Python.

Per l'ultima versione di sviluppo di SDK di crittografia AWS per Python, vai alla [aws-encryption-sdk-python](#) archivio in GitHub.

Dopo aver installato il SDK di crittografia AWS per Python, inizia guardando il [Codice di esempio Python](#) in questa guida.

## Codice di esempio per SDK di crittografia AWS per Python

Gli esempi seguenti illustrano come utilizzare SDK di crittografia AWS per Python per crittografare e decrittare i dati.

Gli esempi in questa sezione mostrano come utilizzare [versione 2.0.x](#) e successivamente del SDK di crittografia AWS per Python. Per esempi che utilizzano versioni precedenti, trova la versione nella [Versioni di](#) elenco della [aws-encryption-sdk-python](#) repository su GitHub.

### Argomenti

- [Crittografia e decrittazione di stringhe](#)
- [Crittografia e decrittazione di flussi di byte](#)
- [Crittografia e decrittazione dei flussi di byte con più provider di chiavi master](#)

- [Utilizzo del caching della chiave di dati per crittografare i messaggi](#)

## Crittografia e decrittazione di stringhe

L'esempio seguente illustra come utilizzare AWS Encryption SDK per crittografare e decrittare le stringhe. Questo esempio utilizza una AWS KMS key nel [AWS Key Management Service \(AWS KMS\)](#) come chiave master.

Durante la crittografia, il `StrictAwsKmsMasterKeyProvider` il costruttore prende un ID chiave, l'ARN di chiave, il nome dell'alias o l'ARN di alias. Durante la decrittografia, è [richiede una chiave ARN](#). In questo caso, perché il `keyArn` parametro viene utilizzato per crittografare e decrittografare, il suo valore deve essere una chiave ARN. Per informazioni sugli ID per AWS KMS chiavi, vedi [Identificatori chiave](#) nella [AWS Key Management Service Guida per gli sviluppatori](#).

```
# 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 AWS KMS key.

    :param str key_arn: Amazon Resource Name (ARN) of the AWS 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()
)

```

## Crittografia e decrittazione di flussi di byte

L'esempio seguente illustra come utilizzare AWS Encryption SDK per crittografare e decrittare i flussi di byte. Questo esempio non utilizza AWS. Utilizza un provider di chiavi master statico e passeggero.

Durante la crittografia, questo esempio utilizza una suite di algoritmi alternativi senza [firme digitali](#) (AES\_256\_GCM\_HKDF\_SHA512\_COMMIT\_KEY). Questa suite di algoritmi è appropriata quando gli utenti che crittografano e decifrano i dati sono ugualmente attendibili. Quindi, durante la decrittografia, l'esempio utilizza il `decrypt-unsigned` modalità di streaming, che fallisce se incontra

testo cifrato firmato. Lademocrypt-unsignedviene introdotta la modalità streaming inAWS Encryption SDKversioni 1.9.xe 2.2.x.

```
# 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

```

## Crittografia e decrittazione dei flussi di byte con più provider di chiavi master

L'esempio seguente spiega come utilizzare AWS Encryption SDK con più di un provider di chiavi master. L'utilizzo di più di un provider di chiavi master crea ridondanza se un provider di chiavi master non è disponibile per la decrittografia. Questo esempio utilizza una AWS KMS key e una key pair RSA come chiavi master.

Questo esempio crittografa con il [suite di algoritmi predefinita](#), che include una [firma digitale](#). Durante lo streaming, il AWS Encryption SDK lascia testo chiaro dopo i controlli di integrità, ma prima di aver verificato la firma digitale. Per evitare di utilizzare il testo normale fino alla verifica della firma, questo esempio esegue il buffer del testo normale e lo scrive su disco solo al termine della decrittografia e della verifica.

```
# 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)

```

## Utilizzo del caching della chiave di dati per crittografare i messaggi

L'esempio seguente mostra come utilizzare il [caching della chiave di dati](#) in SDK di crittografia AWS per Python. È progettato per mostrarti come configurare un'istanza della [cache locale](#) (LocalCryptoMaterialsCache) con il valore di capacità richiesto e un'istanza del [responsabile della cache di materiali crittografici](#) (memorizzazione nella cache CMM) [consoglie di sicurezza cache](#).

Questo esempio molto semplice crea una funzione che crittografa una stringa fissa. Consente di specificare un AWS KMS key, le dimensioni della cache (capacità) richieste e un valore massimo per l'età. Per un esempio più complesso e realistico del caching della chiave di dati, consulta la [Codice di esempio di memorizzazione nella cache delle chiavi dati](#).

Anche se facoltativo, questo esempio utilizza anche un [contesto di crittografia](#) come dati autenticati aggiuntivi. Quando decritti i dati crittografati con un contesto di crittografia, l'applicazione deve verificare che tale contesto sia quello previsto prima di restituire i dati di testo normale

all'intermediario. Un contesto di crittografia è un elemento della best practice consigliata per qualsiasi operazione di crittografia o decrittazione, ma gioca un ruolo speciale nel caching della chiave di dati. Per informazioni dettagliate, consulta [.Contesto di crittografia: Come selezionare le voci di 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_ENCRYPT_POLICY)

# 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

```

## Interfaccia a riga di comando AWS Encryption SDK

La AWS Encryption SDK Interfaccia a riga di comando (AWS CLI) consente di utilizzare AWS Encryption SDK per crittografare e decrittografare i dati in modo interattivo nella riga di comando e negli script. Non è necessario avere competenze specifiche di crittografia o programmazione.

### Note

Le versioni della AWS CLI di crittografia precedente alla 4.0.0 si trovano in IDE [end-of-support fase](#). Puoi eseguire l'aggiornamento in sicurezza dalla versione 2.1.xe successivamente alla versione più recente di AWS Crittografia CLI senza modifiche al codice o ai dati. Tuttavia, [nuove funzioni di sicurezza](#) introdotta nella versione 2.1.x non sono compatibili

con le versioni precedenti. Per eseguire l'aggiornamento dalla versione 1.7.xo versioni precedenti, è necessario eseguire l'aggiornamento alla versione più recente 1.xversione delAWSCrittografia CLI. Per dettagli, consulta [Migrazione della tuaAWS Encryption SDK](#). Le nuove funzionalità di sicurezza sono state inizialmente rilasciate inAWSEncryption CLI versioni 1.7.xe 2.0.x. Tuttavia,AWSCrittografia CLI versione 1.8.xsostituisce la versione 1.7.xeAWSCrittografia CLI 2.1.xsostituisce 2.0.x. Per informazioni dettagliate, consulta la sezione relativa[consulenza di sicurezza](#)nel[aws-encryption-sdk-cli](#)archivio su GitHub.

Come tutte le implementazioni delAWS Encryption SDK, ilAWSEncryption CLI offre funzionalità avanzate di protezione dei dati. Eccone alcuni[crittografia envelope](#), dati autenticati aggiuntivi (AAD) e chiave simmetrica sicura, autenticata e simmetrica[suite di algoritmi](#), ad esempio AES-GCM a 256 bit con derivazione di chiave a 256 bit,[impegno chiave](#)e firma.

IIAWSLa CLI di crittografia si basa sulla crittografia CLI[SDK di crittografia AWS per Python](#)ed è supportata su Linux, macOS e Windows. È possibile eseguire comandi e script per crittografare e decrittografare i dati nella shell preferita su Linux o macOS, in una finestra del prompt dei comandi (cmd.exe) su Windows e in PowerShell console su qualsiasi sistema.

Tutte le implementazioni specifiche del linguaggioAWS Encryption SDK, tra cuiAWSEncryption CLI, sono interoperabili. Ad esempio, è possibile crittografare i dati con[SDK di crittografia AWS per Java](#)e decrittografarlo conAWSCrittografia CLI.

In questo argomento viene presentata laAWSEncryption CLI, spiega come installarlo e utilizzarlo e fornisce diversi esempi per aiutarti a iniziare. Per un avvio rapido, vedi[Come crittografare e decrittografare i dati conAWSCriptazione CLI](#)nelAWSBlog sulla sicurezza. Per informazioni più dettagliate, consulta.[Lettura di The Doc](#)se unisciti a noi nello sviluppo delAWSCrittografia CLI in[aws-encryption-sdk-cli](#)archivio su GitHub.

## Prestazioni

IIAWSLa CLI di crittografia si basa sulla crittografia CLISDK di crittografia AWS per Python. Ogni volta che si esegue l'interfaccia a riga di comando, avvia una nuova istanza del runtime Python. Per migliorare le prestazioni, quando possibile, utilizzare un singolo comando anziché una serie di comandi indipendenti. Ad esempio, eseguire un comando che elabora i file in una directory in modo ricorsivo invece di eseguire comandi separati per ogni file.

## Argomenti

- [Installazione dell'interfaccia a riga di comando di AWS Encryption SDK](#)

- [Come usare la CLI di AWS crittografia](#)
- [Esempi diAWSCLI di crittografia](#)
- [Sintassi e riferimento parametri dell'interfaccia a riga di comando AWS Encryption SDK](#)
- [Versioni delAWSCriptazione CLI](#)

## Installazione dell'interfaccia a riga di comando di AWS Encryption SDK

In questo argomento viene descritto come installareAWSCrittografia CLI. Per informazioni dettagliate, consulta[aws-encryption-sdk-cli](#)repository su GitHub e[Lettura di documenti](#).

### Argomenti

- [Installazione dei prerequisiti](#)
- [Installazione e aggiornamento delAWSCriptazione CLI](#)

## Installazione dei prerequisiti

IIAWSLa riga di crittografia è basata suSDK di crittografia AWS per Python. Per installareAWSEncryption CLI, hai bisogno di Python e*pip*, lo strumento di gestione dei pacchetti Python. Python e *pip* sono disponibili su tutte le piattaforme supportate.

Installare i seguenti prerequisiti prima di installareAWSCrittografia CLI,

### Python

Python 3.6 o versioni successive è richiesto daAWSEncryption, versioni 4.1.0 e successive.

Versioni precedenti diAWSEncryption CLI supporta Python 2.7 e 3.4 e versioni successive, ma si consiglia di utilizzare la versione più recente diAWSCrittografia CLI.

Python è incluso nella maggior parte delle installazioni Linux e macOS, ma è necessario eseguire l'aggiornamento a Python 3.6 o versione successiva. Consigliamo di utilizzare la versione più recente di Python. Su Windows, devi installare Python; non è installato di default. Per scaricare e installare Python, consul[Download Python](#).

Per stabilire se Python è installato, nella riga di comando, digita quanto segue.

```
python
```

Per verificare la versione di Python, utilizza il parametro `-V` (V maiuscola).

```
python -V
```

In Windows, dopo aver installato Python, aggiungi il percorso al file `python.exe` al valore di `Path` di ambiente.

Per impostazione predefinita, Python è installato nella directory di tutti gli utenti o in una directory profilo utente (`$home` o `%userprofile%`) nella sottodirectory `AppData\Local\Programs\Python`. Per trovare la posizione del file `python.exe` nel sistema, verifica una delle seguenti chiavi di registro. È possibile PowerShell per eseguire una ricerca nel registro.

```
PS C:\> dir HKLM:\Software\Python\PythonCore\version\InstallPath
# -or-
PS C:\> dir HKCU:\Software\Python\PythonCore\version\InstallPath
```

## pip

`pip` è il programma di gestione del pacchetto Python. Per installare `AWSCryptography CLI` e relative dipendenze, di cui hai bisogno `pip 8.1` o versioni successive. Per semplificare l'installazione o l'aggiornamento di `pip`, consulta [Installazione](#) nella documentazione di `pip`.

Nelle installazioni Linux, versioni di `pip` precedenti alla 8.1 non possono creare il pacchetto `AWSCryptography CLI` che `AWSCryptography CLI` richiede. Se scegli di non aggiornare il tuo `pip`, è possibile installare gli strumenti di compilazione separatamente. Per ulteriori informazioni, consulta la sezione relativa alla [creazione di una crittografia in Linux](#).

## AWS Command Line Interface

Il `AWS Command Line Interface (AWS CLI)` è obbligatorio solo se si utilizza `AWS KMS keys` nel `AWS Key Management Service (AWS KMS)` con `AWSCryptography CLI`. Se si sta usando un altro [provider di chiavi master](#), `AWS CLI` non è richiesto.

Da utilizzare `AWS KMS keys` con `AWSCryptography CLI`, è necessario [installare e configurare](#) lo `AWS CLI`. La configurazione crea le credenziali utilizzate per l'autenticazione a `AWS KMS` a disposizione `AWSCryptography CLI`.

## Installazione e aggiornamento dell'AWS Crittografia CLI

Installare la versione più recente di AWS Crittografia CLI. Quando si utilizza `pip` per installare AWS Encryption CLI, installa automaticamente le librerie di cui la CLI ha bisogno, tra cui [SDK di crittografia AWS per Python](#), [Python libreria di crittografia](#) e [AWS SDK for Python \(Boto3\)](#).

### Note

Versioni dell'AWS CLI a riga di crittografia precedente alla 4.0.0 si trova in [end-of-support fase](#).

È possibile eseguire l'aggiornamento in sicurezza dalla versione 2.1.xe successivamente alla versione più recente di AWS Crittografia CLI senza modifiche al codice o ai dati.

Tuttavia [nuove funzioni di sicurezza](#) introdotto nella versione 2.1.x non sono compatibili con le versioni precedenti. Per eseguire l'aggiornamento dalla versione 1.7.x o versioni precedenti, è necessario eseguire l'aggiornamento alla versione più recente 1.x versione di AWS Crittografia CLI. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Le nuove funzionalità di sicurezza sono state inizialmente rilasciate in AWS Encryption, versioni 1.7.xe 2.0.x. Tuttavia AWS Crittografia CLI versione 1.8.x sostituisce la versione 1.7.xe AWS CLI-up 2.1.x sostituisce 2.0.x. Per informazioni dettagliate, consulta la pagina [consulenza sulla sicurezza nel aws-encryption-sdk-cli repository](#) su GitHub.

Per installare la versione più recente di AWS Crittografia CLI

```
pip install aws-encryption-sdk-cli
```

Per eseguire l'aggiornamento alla versione più recente di AWS Crittografia CLI

```
pip install --upgrade aws-encryption-sdk-cli
```

Per trovare i numeri di versione di AWS Crittografia CLI e AWS Encryption SDK

```
aws-encryption-cli --version
```

L'output elenca i numeri di versione di entrambe le librerie.

```
aws-encryption-sdk-cli/2.1.0 aws-encryption-sdk/2.0.0
```

Per eseguire l'aggiornamento alla versione più recente di AWS CLI

```
pip install --upgrade aws-encryption-sdk-cli
```

Installazione di AWS SDK for Python (Boto3), se non è già installato. Se Boto3 è installato, l'installatore verifica la versione di Boto3 e la aggiorna se necessario.

Per trovare la versione installata di Boto3

```
pip show boto3
```

Per eseguire l'aggiornamento alla versione più recente di Boto3

```
pip install --upgrade boto3
```

Per installare la versione di AWS CLI attualmente in fase di sviluppo, vedi [aws-encryption-sdk-cli](#) repository su GitHub.

Per ulteriori dettagli sull'utilizzo di pip per installare e aggiornare i pacchetti Python, consulta la [documentazione relativa a pip](#).

## Come usare la CLI di AWS crittografia

Questo argomento spiega come utilizzare i parametri nella CLI di AWS crittografia. Per alcuni esempi, consulta [Esempi di AWS CLI di crittografia](#). Per la documentazione completa, consulta [Leggi i documenti](#). La sintassi mostrata in questi esempi si riferisce alla versione 2.1 di AWS Encryption CLI. x e versioni successive.

### Note

[Le versioni di AWS Encryption CLI precedenti alla 4.0.0 sono in fase e. nd-of-support](#)

Puoi aggiornare in sicurezza dalla versione 2.1. x e versioni successive all'ultima versione di AWS Encryption CLI senza modifiche al codice o ai dati. Tuttavia, [nuove funzionalità di sicurezza sono](#) state introdotte nella versione 2.1. x non sono compatibili con le versioni precedenti. Per aggiornare dalla versione 1.7. x o versioni precedenti, devi prima eseguire l'aggiornamento alla versione più recente 1. versione x di AWS Encryption CLI. Per informazioni dettagliate, consultare [Migrazione della tua AWS Encryption SDK](#).

Le nuove funzionalità di sicurezza sono state originariamente rilasciate nelle versioni 1.7 di AWS Encryption CLI. x e 2.0. x. Tuttavia, AWS Encryption CLI versione 1.8. x sostituisce la versione 1.7. x e AWS Encryption CLI 2.1. x sostituisce 2.0. x. Per i dettagli, [consulta l'avviso di sicurezza pertinente nel aws-encryption-sdk-clirepository](#) su. GitHub

Per un esempio che mostra come utilizzare la funzionalità di sicurezza che limita le chiavi dati crittografate, vedere [Limitazione delle chiavi di dati crittografate](#).

Per un esempio che mostra come utilizzare le chiavi AWS KMS multiregionali, vedere [Utilizzo di più regioni AWS KMS keys](#).

## Argomenti

- [Come crittografare e decrittare i dati](#)
- [Come specificare le chiavi di avvolgimento](#)
- [Come fornire l'input](#)
- [Come specificare la posizione dell'output](#)
- [Come utilizzare un contesto di crittografia](#)
- [Come specificare una politica di impegno](#)
- [Come archiviare i parametri in un file di configurazione](#)

## Come crittografare e decrittare i dati

L'AWSEncryption CLI utilizza le funzionalità di AWS Encryption SDK per semplificare la crittografia e la decrittografia dei dati in modo sicuro.

### Note

Il `--master-keys` parametro è obsoleto nella versione 1.8. x dell'AWSEncryption CLI e rimosso nella versione 2.1. x. Utilizza invece il parametro `--wrapping-keys`. A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è richiesto per la crittografia e la decrittografia. Per informazioni dettagliate, consultare [Sintassi e riferimento parametri dell'interfaccia a riga di comando AWS Encryption SDK](#).

- Quando si crittografano i dati nella CLI di AWS crittografia, si specificano i dati in chiaro e una chiave di [avvolgimento \(o chiave master\)](#), ad esempio in (). AWS KMS key AWS Key Management Service AWS KMS Se si utilizza un provider di chiavi master personalizzate, è necessario specificare anche il provider. È inoltre possibile specificare i percorsi di output per il [messaggio crittografato](#) e per i metadati relativi all'operazione di crittografia. Un [contesto di crittografia](#) è opzionale, ma consigliato.

Nella versione 1.8. x, il `--commitment-policy` parametro è obbligatorio quando si utilizza il `--wrapping-keys` parametro; altrimenti non è valido. A partire dalla versione 2.1. x, il `--commitment-policy` parametro è facoltativo, ma consigliato.

```
aws-encryption-cli --encrypt --input myPlaintextData \  
  --wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab \  
  --output myEncryptedMessage \  
  --metadata-output ~/metadata \  
  --encryption-context purpose=test \  
  --commitment-policy require-encrypt-require-decrypt
```

L'AWSEncryption CLI crittografa i tuoi dati con una chiave dati univoca. Quindi crittografa la chiave dati sotto le chiavi di avvolgimento specificate. Restituisce un [messaggio crittografato](#) e i metadati relativi all'operazione. Il messaggio crittografato contiene i dati crittografati (testo cifrato) e una copia crittografata della chiave di dati. Non devi preoccuparti di archiviare, gestire o perdere la chiave di dati.

- Quando esegui la decrittografia dei dati, passi il tuo messaggio crittografato, il contesto di crittografia opzionale e la posizione per l'output di testo non crittografato e i metadati. È inoltre possibile specificare le chiavi di wrapping che la CLI di AWS crittografia può utilizzare per decrittografare il messaggio o indicare alla CLI di AWS crittografia che può utilizzare qualsiasi chiave di wrapping che ha crittografato il messaggio.

A partire dalla versione 1.8. x, il `--wrapping-keys` parametro è facoltativo durante la decrittografia, ma consigliato. A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è richiesto per la crittografia e la decrittografia.

Durante la decrittografia, è possibile utilizzare l'attributo chiave del `--wrapping-keys` parametro per specificare le chiavi di wrapping che decrittografano i dati. Specificare una chiave di AWS KMS avvolgimento durante la decrittografia è facoltativo, ma è una procedura [consigliata che](#) impedisce

di utilizzare una chiave che non si intendeva utilizzare. Se utilizzi un provider di chiavi master personalizzate, devi specificare il provider e la chiave di wrapping.

Se non si utilizza l'attributo `key`, è necessario impostare l'attributo [discovery](#) del `--wrapping-keys` parametro su `true`, che consente alla CLI di AWS crittografia di decodifica utilizzando qualsiasi chiave di wrapping che ha crittografato il messaggio.

È consigliabile utilizzare il `--max-encrypted-data-keys` parametro per evitare di decifrare un messaggio malformato con un numero eccessivo di chiavi dati crittografate. Specifica il numero previsto di chiavi dati crittografate (una per ogni chiave di wrapping utilizzata nella crittografia) o un numero massimo ragionevole (ad esempio 5). Per informazioni dettagliate, consultare [Limitazione delle chiavi di dati crittografate](#).

Il `--buffer` parametro restituisce testo in chiaro solo dopo l'elaborazione di tutti gli input, inclusa la verifica della firma digitale, se presente.

Il `--decrypt-unsigned` parametro decrittografa il testo cifrato e garantisce che i messaggi non siano firmati prima della decrittografia. Usa questo parametro se hai utilizzato il `--algorithm` parametro e hai selezionato una suite di algoritmi senza firma digitale per crittografare i dati. Se il testo cifrato è firmato, la decrittografia non riesce.

È possibile utilizzare `--decrypt` o `--decrypt-unsigned` per la decrittografia, ma non entrambi.

```
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
```

L'AWS Encryption CLI utilizza la chiave di wrapping per decrittografare la chiave di dati nel messaggio crittografato. Quindi usa la chiave di dati per decrittografare i dati. Restituisce i tuoi dati di testo non crittografato e i metadati relativi all'operazione.

## Come specificare le chiavi di avvolgimento

Quando si crittografano i dati nella CLI di AWS crittografia, è necessario specificare almeno una chiave di [wrapping \(o chiave master\)](#). Puoi usare AWS KMS keys in AWS Key Management Service (AWS KMS), avvolgere le chiavi di un [provider di chiavi master](#) personalizzato o entrambi. Il provider di chiavi master personalizzato può essere qualsiasi provider di chiave master compatibile con Python.

Per specificare le chiavi di wrapping nelle versioni 1.8. x e versioni successive, utilizzare il `--wrapping-keys` parametro (`-w`). Il valore di questo parametro è una raccolta di [attributi](#) con il `attribute=value` formato. Gli attributi utilizzati dipendono dal provider di chiavi master e dal comando.

- AWS KMS. Nei comandi di crittografia, è necessario specificare un `--wrapping-keys` parametro con un attributo chiave. A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è richiesto anche nei comandi di decrittografia. Durante la decrittografia, il `--wrapping-keys` parametro deve avere un attributo chiave o un attributo discovery con un valore pari a `true` (ma non entrambi). Gli altri attributi sono opzionali.
- Provider di chiavi master personalizzato. È necessario specificare un `--wrapping-keys` parametro in ogni comando. Il valore di parametro deve avere gli attributi relativi a chiave e provider.

È possibile includere [più `--wrapping-keys` parametri](#) e più attributi chiave nello stesso comando.

### Confezionamento degli attributi dei parametri chiave

Il valore del parametro `--wrapping-keys` consiste nei seguenti attributi e nei loro rispettivi valori. Un `--wrapping-keys` parametro (o `--master-keys` parametro) è richiesto in tutti i comandi di crittografia. A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è richiesto anche durante la decrittografia.

Se nel nome o nel valore di un attributo sono inclusi spazi o caratteri speciali, racchiudi il nome e il valore tra virgolette. Ad esempio, `--wrapping-keys key=12345 "provider=my cool provider"`.

### Chiave: specifica una chiave di avvolgimento

Usa l'attributo `key` per identificare una chiave di avvolgimento. Durante la crittografia, il valore può essere qualsiasi identificatore di chiave riconosciuto dal provider della chiave master.

```
--wrapping-keys key=1234abcd-12ab-34cd-56ef-1234567890ab
```

In un comando di crittografia, è necessario includere almeno un attributo e un valore chiave. Per crittografare la chiave dati con più chiavi di avvolgimento, utilizza [più](#) attributi chiave.

```
aws-encryption-cli --encrypt --wrapping-keys
key=1234abcd-12ab-34cd-56ef-1234567890ab key=1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
```

Nei comandi di crittografia utilizzati AWS KMS keys, il valore della chiave può essere l'ID della chiave, la relativa chiave ARN, un nome alias o l'alias ARN. Ad esempio, questo comando di crittografia utilizza un ARN alias nel valore dell'attributo della chiave. Per i dettagli sugli identificatori chiave di un AWS KMS key, consulta [Identificatori chiave](#) nella Guida per gli sviluppatori. AWS Key Management Service

```
aws-encryption-cli --encrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:alias/ExampleAlias
```

Nei comandi di decrittografia che utilizzano un provider di chiavi master personalizzato, sono necessari gli attributi chiave e provider.

```
\\ Custom master key provider
aws-encryption-cli --decrypt --wrapping-keys provider='myProvider' key='100101'
```

Nei comandi di decrittografia utilizzati AWS KMS, è possibile utilizzare l'attributo key per specificare l'attributo da utilizzare AWS KMS keys per la decrittografia o l'[attributo discovery](#) con un valore di true, che consente alla CLI di AWS crittografia di utilizzare tutto AWS KMS key ciò che è stato utilizzato per crittografare il messaggio. Se si specifica un AWS KMS key, deve essere una delle chiavi di wrapping utilizzate per crittografare il messaggio.

Specificare la chiave di avvolgimento è una procedura [AWS Encryption SDK consigliata](#). Assicura che utilizzi ciò che AWS KMS key intendi utilizzare.

[In un comando di decrypt, il valore dell'attributo key deve essere una chiave ARN.](#)

```
\\ AWS KMS key
aws-encryption-cli --decrypt --wrapping-keys key=arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

## Discovery: usare uno qualsiasi AWS KMS key durante la decrittografia

Se non è necessario limitare l'utilizzo durante AWS KMS keys la decrittografia, è possibile utilizzare l'attributo `discovery` con un valore di `true`. Un valore di `true` consente alla CLI di AWS crittografia di decrittografare utilizzando qualsiasi elemento AWS KMS key che abbia crittografato il messaggio. Se non specifichi un attributo `discovery`, `discovery` è `false` (impostazione predefinita). L'attributo `discovery` è valido solo nei comandi di `decrypt` e solo quando il messaggio è stato crittografato con. AWS KMS keys

L'attributo `discovery` con un valore di `true` è un'alternativa all'utilizzo dell'attributo `key` per specificare. AWS KMS keys Quando si decrittografa un messaggio crittografato con AWS KMS keys, ogni `--wrapping-keys` parametro deve avere un attributo chiave o un attributo `discovery` con un valore pari a `true`, ma non entrambi.

Quando `discovery` è vero, è consigliabile utilizzare gli attributi `discovery-partition` e `discovery-account` per limitare gli utenti a quelli AWS KMS keys specificati. Account AWS Nell'esempio seguente, gli attributi di rilevamento consentono alla CLI di AWS crittografia di utilizzare qualsiasi AWS KMS key elemento specificato Account AWS.

```
aws-encryption-cli --decrypt --wrapping-keys \  
  discovery=true \  
  discovery-partition=aws \  
  discovery-account=111122223333 \  
  discovery-account=444455556666
```

## Provider: specifica il fornitore della chiave principale

L'attributo del `provider` identifica il [provider della chiave master](#). Il valore predefinito è `aws-kms`, che rappresenta AWS KMS. Se stai usando un altro provider di chiavi master, è necessario utilizzare l'attributo `provider`.

```
--wrapping-keys key=12345 provider=my_custom_provider
```

Per ulteriori informazioni sull'utilizzo di fornitori di chiavi master (non AWS KMS) personalizzate, vedere l'argomento Configurazione avanzata nel file [README](#) per il repository [AWS Encryption CLI](#).

## Regione: specificare una Regione AWS

Utilizzare l'attributo `region` per specificare il valore Regione AWS di un AWS KMS key. Questo attributo è valido solo per i comandi di crittografia e solo quando il provider della chiave master è AWS KMS.

```
--encrypt --wrapping-keys key=alias/primary-key region=us-east-2
```

AWSI comandi CLI di crittografia utilizzano quanto specificato nel valore dell'attributo chiave se include una regione, ad esempio un ARN. Se il valore della chiave specifica a Regione AWS, l'attributo `region` viene ignorato. Regione AWS

L'attributo della regione prevale su altre specifiche relative alle regioni. Se non utilizzi un attributo regionale, i comandi AWS Encryption CLI utilizzano Regione AWS quanto specificato nel [profilo AWS CLI indicato](#), se presente, o il tuo profilo predefinito.

## Profilo: specifica un profilo denominato

Utilizza l'attributo del profilo per specificare un AWS CLI profilo denominato <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html#cli-configure-files-using-profiles>. I profili denominati possono includere credenziali e un Regione AWS. Questo attributo è valido solo quando il provider della chiave master è AWS KMS.

```
--wrapping-keys key=alias/primary-key profile=admin-1
```

È possibile utilizzare l'attributo del profilo per specificare credenziali alternative nei comandi di crittografia e decrittografia. In un comando di crittografia, la CLI di AWS crittografia utilizza il valore Regione AWS nel profilo denominato solo quando il valore della chiave non include una regione e non è presente alcun attributo di regione. In un comando di decriptazione, il profilo Regione AWS in the name viene ignorato.

## Come specificare più chiavi di wrapping

È possibile specificare più chiavi di wrapping (o chiavi master) in ogni comando.

Se si specifica più di una chiave di wrapping, la prima chiave di wrapping genera e crittografa la chiave dati utilizzata per crittografare i dati. Le altre chiavi di wrapping crittografano la stessa chiave dati. Il [messaggio crittografato](#) risultante contiene i dati crittografati («testo cifrato») e una raccolta

di chiavi dati crittografate, una crittografata da ciascuna chiave di wrapping. Qualsiasi involucro può decrittografare una chiave di dati crittografata e quindi decrittografare i dati.

Esistono due modi per specificare più chiavi di wrapping:

- Includi più attributi chiave nel valore del `--wrapping-keys` parametro.

```
$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
```

- Includere molteplici parametri `--wrapping-keys` nello stesso comando. Utilizzate questa sintassi quando i valori degli attributi specificati non si applicano a tutte le chiavi di avvolgimento del comando.

```
--wrapping-keys region=us-east-2 key=alias/test_key \
--wrapping-keys region=us-west-1 key=alias/test_key
```

L'attributo `discovery` con un valore di `true` consente alla CLI di AWS crittografia di utilizzare qualsiasi elemento AWS KMS key che ha crittografato il messaggio. Se si utilizzano più `--wrapping-keys` parametri nello stesso comando, l'utilizzo `discovery=true` di qualsiasi `--wrapping-keys` parametro sostituisce efficacemente i limiti dell'attributo chiave negli altri parametri. `--wrapping-keys`

Ad esempio, nel comando seguente, l'attributo `key` nel primo `--wrapping-keys` parametro limita la CLI di AWS crittografia a quanto specificato. AWS KMS key Tuttavia, l'attributo `discovery` nel secondo `--wrapping-keys` parametro consente alla CLI di AWS crittografia di utilizzare qualsiasi account AWS KMS key negli account specificati per decrittografare il messaggio.

```
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
```

## Come fornire l'input

[L'operazione di crittografia nella CLI di AWS crittografia accetta dati in testo normale come input e restituisce un messaggio crittografato.](#) L'operazione di decrittografia richiede un messaggio crittografato come input e restituisce un dati di testo non crittografato.

Il `--input` parametro (`-i`), che indica alla CLI di AWS crittografia dove trovare l'input, è richiesto in tutti i comandi di AWS Encryption CLI.

Puoi fornire l'input in uno dei seguenti modi:

- Utilizza un file.

```
--input myData.txt
```

- Usa un modello di nome di file.

```
--input testdir/*.xml
```

- Utilizza una directory o un modello di nome di directory. Quando l'input è una directory, il `--recursive` parametro (`-r`, `-R`) è obbligatorio.

```
--input testdir --recursive
```

- Indirizza l'input al comando (stdin). Utilizza un valore di `-` per il parametro `--input`. (Il parametro `--input` è sempre obbligatorio).

```
echo 'Hello World' | aws-encryption-cli --encrypt --input -
```

## Come specificare la posizione dell'output

Il `--output` parametro indica alla CLI di AWS crittografia dove scrivere i risultati dell'operazione di crittografia o decrittografia. È richiesto in ogni comando di AWS Encryption CLI. L'AWS Encryption CLI crea un nuovo file di output per ogni file di input dell'operazione.

Se esiste già un file di output, per impostazione predefinita, AWS Encryption CLI stampa un avviso, quindi sovrascrive il file. Per evitare la sovrascrittura, utilizza il parametro `--interactive` che richiede la conferma prima di sovrascrivere oppure `--no-overwrite`, che salta l'input nel caso in cui l'output comporti una sovrascrittura. Per annullare l'avviso relativo alla sovrascrittura, utilizza

`--quiet`. Per acquisire errori e avvisi dalla CLI di AWS crittografia, utilizza l'operatore di `>&1` reindirizzamento per scriverli nel flusso di output.

### Note

I comandi che sovrascrivano i file di output cominciano con l'eliminare il file di output. Se il comando ha esito negativo, il file di output potrebbe essere già stato eliminato.

Puoi modificare la posizione dell'output in diversi modi.

- Specifica un nome di file. Se specifichi un percorso per il file, tutte le directory nel percorso devono esistere prima di eseguire il comando.

```
--output myEncryptedData.txt
```

- Specifica una directory. La directory di output deve esistere prima di eseguire il comando.

Se l'input contiene sottodirectory, il comando riproduce le sottodirectory nella directory specificata.

```
--output Test
```

Quando la posizione di output è una directory (senza nomi di file), l'AWSEncryption CLI crea i nomi dei file di output in base ai nomi dei file di input più un suffisso. Le operazioni di crittografia aggiungono `.encrypted` al nome del file di input, mentre le operazioni di decrittografia aggiungono `.decrypted`. Per modificare il suffisso, utilizza il parametro `--suffix`.

Ad esempio, se esegui la crittografia `file.txt`, il comando di crittografia crea `file.txt.encrypted`. Se esegui la decrittografia `file.txt.encrypted`, il comando di decrittografia crea `file.txt.encrypted.decrypted`.

- Scrivi nella riga di comando (stdout). Inserisci un valore di `-` per il parametro `--output`. È possibile utilizzare `--output -` per reindirizzare l'output a un altro comando o programma.

```
--output -
```

## Come utilizzare un contesto di crittografia

L'AWSEncryption CLI consente di fornire un contesto di crittografia nei comandi di crittografia e decrittografia. Non è necessario, ma è una best practice crittografica che ti consigliamo.

Un contesto di crittografia è rappresentato da tipi di dati autenticati aggiuntivi arbitrari e non segreti. Nella CLI di AWS crittografia, il contesto di crittografia è costituito da una raccolta di `name=value` coppie. È possibile utilizzare i contenuti nelle coppie, incluse le informazioni sui file, i dati che consentono di individuare le operazioni di crittografia nei log o i dati richiesti da concessioni e policy.

### In un comando di crittografia

Il contesto di crittografia specificato in un comando di crittografia, insieme alle eventuali coppie aggiunte dal [CMM](#), è vincolato a livello crittografico ai dati crittografati. Inoltre è incluso (in testo normale) nel [messaggio crittografato](#) che il comando restituisce. Se si utilizza un AWS KMS key, il contesto di crittografia potrebbe anche apparire in testo normale nei record e nei registri di controllo, ad esempio. AWS CloudTrail

L'esempio seguente mostra un contesto di crittografia con tre coppie `name=value`.

```
--encryption-context purpose=test dept=IT class=confidential
```

### In un comando di decrittazione

In un comando di decrittografia, il contesto di crittografia consente di confermare che stai decrittografando il messaggio crittografato corretto.

Non devi fornire un contesto di crittografia in un comando di decrittografia, anche se, per crittografare, è stato usato un contesto di crittografia. Tuttavia, se lo fai, l'AWSEncryption CLI verifica che ogni elemento nel contesto di crittografia del comando `decrypt` corrisponda a un elemento nel contesto di crittografia del messaggio crittografato. Se non corrisponde alcun elemento, il comando di decrittografia ha esito negativo.

Ad esempio, il comando seguente decrittografa il messaggio crittografato solo se il relativo contesto di crittografia include `dept=IT`.

```
aws-encryption-cli --decrypt --encryption-context dept=IT ...
```

Un contesto di crittografia è una parte importante della tua strategia di sicurezza. Tuttavia, quando scegli un contesto di crittografia, ricordati che i suoi valori non sono segreti. Non includere dati riservati nel contesto di crittografia.

## Per specificare un contesto di crittografia

- In un comando di crittografia, utilizza il parametro `--encryption-context` con una o più coppie `name=value`. Utilizza uno spazio per separare ogni coppia.

```
--encryption-context name=value [name=value] ...
```

- In un comando di decrittografia, il valore del parametro `--encryption-context` può includere coppie `name=value`, elementi `name` (senza valori) o una combinazione di entrambi.

```
--encryption-context name[=value] [name] [name=value] ...
```

Se `name` o `value` in una coppia `name=value` include spazi o caratteri speciali, racchiudi la coppia completa tra virgolette.

```
--encryption-context "department=software engineering" "Regione AWS=us-west-2"
```

Ad esempio, questo comando di crittografia include un contesto di crittografia con due coppie, `purpose=test` e `dept=23`.

```
aws-encryption-cli --encrypt --encryption-context purpose=test dept=23 ...
```

Questo comando di decrittografia avrebbe esito positivo. Il contesto di crittografia in ogni comando è un sottoinsieme del contesto di crittografia originale.

```
\\ 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 ...
```

Tuttavia, questi comandi di decrittografia non avrebbero esito positivo. Il contesto di crittografia nel messaggio crittografato non contiene gli elementi specificati.

```
aws-encryption-cli --decrypt --encryption-context dept=Finance ...
```

```
aws-encryption-cli --decrypt --encryption-context scope ...
```

## Come specificare una politica di impegno

Per impostare la [politica di impegno](#) per il comando, utilizzare il `--commitment-policy` parametro. Questo parametro è stato introdotto nella versione 1.8. x. È valido nei comandi di crittografia e decrittografia. La politica di impegno impostata è valida solo per il comando in cui viene visualizzata. Se non si imposta una politica di impegno per un comando, AWS Encryption CLI utilizza il valore predefinito.

Ad esempio, il seguente valore del parametro imposta la politica di impegno `require-encrypt-allow-decrypt`, che crittografa sempre con l'impegno della chiave, ma decrittografa un testo cifrato con o senza impegno di chiave.

```
--commitment-policy require-encrypt-allow-decrypt
```

## Come archiviare i parametri in un file di configurazione

È possibile risparmiare tempo ed evitare errori di digitazione salvando i parametri e i valori di AWS Encryption CLI utilizzati di frequente nei file di configurazione.

Un file di configurazione è un file di testo che contiene parametri e valori per un comando CLI di AWS crittografia. Quando si fa riferimento a un file di configurazione in un comando AWS Encryption CLI, il riferimento viene sostituito dai parametri e dai valori nel file di configurazione. L'effetto è lo stesso se digitassi il contenuto del file nella riga di comando. Un file di configurazione può avere qualsiasi nome e può essere posizionato in qualsiasi directory a cui l'utente attuale può accedere.

Il seguente file di configurazione di esempio, `key.conf`, ne specifica due AWS KMS keys in regioni diverse.

```
--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
```

Per utilizzare il file di configurazione in un comando, inserisci il prefisso con la chiocciola nel nome del file (`@`). In una PowerShell console, usa un carattere di marcia indietro per sfuggire al segno at (`() `@`).

Questo comando di esempio utilizza il file `key.conf` in un comando di crittografia.

## 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
```

## Regole dei file di configurazione

Le regole per l'utilizzo di file di configurazione sono le seguenti:

- È possibile includere più parametri in ogni file di configurazione ed elencarli in qualsiasi ordine. Elenca ciascun parametro con i relativi valori (se presenti) su una riga separata.
- Utilizza # per aggiungere un commento a tutta o a una parte di una riga.
- È possibile includere riferimenti ad altri file di configurazione. Non usare un segno di spunta per sfuggire al @ cartello, anche dentro. PowerShell
- Se utilizzi le virgolette in un file di configurazione, tale testo non può estendersi su più righe.

Ad esempio, questo è il contenuto di un file `encrypt.conf` di esempio.

```
# 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
```

È inoltre possibile includere più file di configurazione in un comando. In questo esempio il comando utilizza entrambi i file di configurazione `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
```

Prossimo: [Prova gli esempi di AWS Encryption CLI](#)

## Esempi diAWSCLI di crittografia

Utilizza i seguenti esempi per provareAWSCLI di crittografia sulla piattaforma che preferisci. Per assistenza sulle chiavi master e altri parametri, consulta [Come usare la CLI di AWS crittografia](#). Per un riferimento rapido, consulta [Sintassi e riferimento parametri dell'interfaccia a riga di comando AWS Encryption SDK](#).

### Note

Gli esempi seguenti utilizzano la sintassi perAWSCLI di crittografia versione 2.1.x. Le nuove funzionalità di sicurezza sono state originariamente rilasciate inAWSCLI di crittografia versione 1.7.xe 2.0.x. Tuttavia,AWSCLI di crittografia versione 1.8.xsostituisce la versione 1.7.xeAWSCLI di crittografia 2.1.xsostituisce 2.0.x. Per ulteriori informazioni, consultare la relativa[Advisory sulla sicurezza](#)nella[aws-encryption-sdk-clirepository](#) suGitHub.

Per un esempio che mostra come utilizzare la funzionalità di sicurezza che limita le chiavi dati crittografate, vedere[Limitazione delle chiavi di dati crittografate](#).

Per un esempio che mostra come usareAWS KMSChiavi multiregione, consulta[Utilizzo di più regioni AWS KMS keys](#).

### Argomenti

- [Crittografia di un file](#)
- [Decrittazione di un file](#)
- [Crittografia di tutti i file in una directory](#)
- [Decrittazione di tutti i file in una directory](#)
- [Crittografia e decrittazione nella riga di comando](#)
- [Utilizzo di più tasti master](#)
- [Crittografia e decrittazione negli script](#)

- [Utilizzo del caching della chiave dei dati](#)

## Crittografia di un file

In questo esempio viene utilizzato il `AWSSCLI` di crittografia per crittografare il contenuto del `hello.txt`, che contiene una stringa «Hello World».

Quando si esegue un comando di crittografia su un file, `AWS CLI` di crittografia ottiene il contenuto del file, genera un unico [chiave di dati](#), crittografa il contenuto del file sotto la chiave dati e quindi scrive il [messaggio crittografato](#) su un nuovo file.

Il primo comando salva la chiave ARN di una `AWS KMS key` nella `$keyArn` variabile. Quando si crittografa con un `AWS KMS key`, puoi identificarlo utilizzando un ID chiave, una chiave ARN, un nome dell'alias o un ARN di alias. Per informazioni dettagliate `AWS KMS key`, consulta [Identificatori chiave](#) nella `AWS Key Management Service Guida per gli sviluppatori`.

Il secondo comando crittografa il contenuto del file. Il comando utilizza il parametro `--encrypt` per specificare l'operazione e il parametro `--input` per indicare il file da crittografare. Il [parametro --wrapping-keys](#) e l'attributo della chiave obbligatorio, comunicano al comando di utilizzare la `AWS KMS key` rappresentata dall'ARN di chiave.

Il comando utilizza il parametro `--metadata-output` per specificare un file di testo per i metadati relativi all'operazione di crittografia. Come best practice, il comando utilizza il parametro `--encryption-context` per specificare un [contesto di crittografia](#).

Questo comando utilizza anche il [--commitment-policy](#) parametro per definire esplicitamente la politica di impegno. Nella versione 1.8.x, questo parametro è obbligatorio quando si utilizza `--wrapping-keys` Parametro. A partire dalla versione 2.1.x, il `--commitment-policy` parametro opzionale, ma consigliato.

Il valore del parametro `--output`, un punto (`.`), consente al comando di scrivere il file di output nella directory corrente.

### 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 il comando di crittografia va a buon fine, non viene restituito alcun output. Per determinare se il comando è riuscito, controlla il valore Booleano nella variabile  `$?` . Quando il comando ha esito positivo, il valore di  `$?`  è `0` (Bash) o `True` (PowerShell). Quando il comando non riesce, il valore di  `$?`  è diverso da zero (Bash) o `False` (PowerShell).

## Bash

```
$ echo $?
0
```

## PowerShell

```
PS C:\> $?
True
```

È inoltre possibile utilizzare un comando di elenco di directory per visualizzare se il comando di crittografia ha creato un nuovo file, `hello.txt.encrypted`. Poiché il comando di crittografia non ha specificato un nome di file per l'output, AWS La CLI di crittografia ha scritto l'output su un file

con lo stesso nome del file di input più un `.encrypted` suffisso. Per utilizzare un suffisso diverso o eliminarlo, utilizza il parametro `--suffix`.

Il file `hello.txt.encrypted` contiene un [messaggio crittografato](#) che include il testo cifrato del file `hello.txt`, una copia cifrata della chiave dei dati e metadati aggiuntivi, tra cui il contesto di crittografia.

## 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
```

## Decrittazione di un file

In questo esempio viene utilizzato il `AWSCLI` di crittografia per decrittografare il contenuto del `hello.txt.encrypted` file crittografato nell'esempio precedente.

Il comando di decrittografia utilizza il parametro `--decrypt` per specificare l'operazione e il parametro `--input` per indicare il file da decrittografare. Il valore del parametro `--output` è un punto che rappresenta la directory attuale.

La `--wrapping-key` parametro con `key` attributo specifica la chiave di wrapping utilizzata per decrittografare il messaggio crittografato. Nel decrittografare i comandi con `AWS KMS keys`, il valore dell'attributo chiave deve essere un [ARN della chiave](#). La `--wrapping-key` parametro è obbligatorio in un comando di decrittografia. Se stai usando `AWS KMS keys` è possibile utilizzare l'attributo `key` da specificare `AWS KMS key` per decrittografare o `scope` attributo con un valore di `true` (ma non entrambi). Se stai utilizzando un provider di chiave master personalizzato, `key-provider` Gli attributi sono obbligatori.

La [--commitment-policy](#) parametro è opzionale a partire dalla versione 2.1.x ma è il valore consigliato. Il suo utilizzo rende esplicitamente chiaro il tuo intento, anche se specifichi il valore predefinito, `require-encrypt-require-decrypt`.

Il parametro `--encryption-context` è facoltativo nel comando di decrittografia, anche quando un [contesto di crittografia](#) viene fornito nel comando di crittografia. In questo caso, il comando di decrittografia usa lo stesso contesto di crittografia fornito nel relativo comando. Prima di decrittografare, il CLI di crittografia verifica che il contesto di crittografia nel messaggio crittografato includa `unapurpose=test`. In caso contrario, il comando di decrittografia ha esito negativo.

Il parametro `--metadata-output` specifica un file di metadati per l'operazione di decrittografia. Il valore del parametro `--output`, un punto (`.`), scrive il file di output nella directory corrente.

Come best practice, utilizzare il `--max-encrypted-data-keys` parametro per evitare di decrittografare un messaggio malformato con un numero eccessivo di chiavi dati crittografate. Specificare il numero previsto di chiavi dati crittografate (una per ogni chiave di wrapping utilizzata nella crittografia) o un massimo ragionevole (ad esempio 5). Per dettagli, consultare [Limitazione delle chiavi di dati crittografate](#).

La `--buffer` restituisce testo semplice solo dopo l'elaborazione di tutti gli input, inclusa la verifica della firma digitale se presente.

## 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 un comando di decrittografia va a buon fine, non viene restituito alcun output. Per stabilire se il comando è riuscito, ottieni il valore della variabile \$? . È inoltre possibile utilizzare un comando di elenco di directory per visualizzare se il comando ha creato un nuovo file con il suffisso `.decrypted`. Per vedere i contenuti di testo normale, utilizza un comando per ottenere il contenuto del file, ad esempio `cat` o [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
```

## Crittografia di tutti i file in una directory

In questo esempio viene utilizzato il `AWSCLI` di crittografia per crittografare i contenuti di tutti i file in una directory.

Quando un comando influisce su più file, `AWS` La crittografia CLI elabora ogni file individualmente. Ottiene i contenuti dei file e una [chiave di dati](#) univoca per il file da una chiave master, crittografa i contenuti del file nella chiave dei dati e scrive i risultati in un nuovo file nella directory di output. Di conseguenza, è possibile decrittografare i file di output in modo indipendente.

Questo elenco della directory `TestDir` mostra i file di testo non crittografato che vogliamo crittografare.

### 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
```

Il primo comando salva il [Amazon Resource Name \(ARN\)](#) di un `AWS KMS` key nella `$keyArn` variabile.

Il secondo comando crittografa i contenuti dei file nella directory `TestDir` e scrive i file di contenuti crittografati nella directory `TestEnc`. Se la directory `TestEnc` non esiste, il comando ha esito negativo. Poiché la posizione di input è una directory, il parametro `--recursive` è obbligatorio.

La [--wrapping-keys](#) parametro, e la sua richiesta chiave attributo, specificare la chiave di avvolgimento da utilizzare. Il comando di crittografia include un [contesto di crittografia](#), `dept=IT`.

Quando viene specificato un contesto di crittografia in un comando che crittografa più file, lo stesso contesto di crittografia viene utilizzato per tutti i file.

Il comando ha anche un `--metadata-output` parametro per dire all'AWS CLI di crittografia dove scrivere i metadati sulle operazioni di crittografia. La CLI di crittografia scrive un record di metadati per ogni file crittografato.

La `--commitment-policy` [parameter](#) è opzionale a partire dalla versione 2.1.x ma è il valore consigliato. Se il comando o lo script non riesce perché non è in grado di decrittografare un testo cifrato, l'impostazione esplicita del criterio di impegno potrebbe aiutarti a rilevare rapidamente il problema.

Quando il comando viene completato, l'AWS CLI di crittografia scrive i file crittografati sul `TestEnc` directory, ma non restituisce alcun output.

Il comando finale elenca i file nella directory `TestEnc`. C'è un file di output di contenuti crittografati per ogni file di input di contenuti di testo non crittografato. Poiché il comando non ha specificato un altro suffisso, il comando di crittografia ha aggiunto `.encrypted` a ciascuno dei nomi dei file di input.

## 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

```

## Decrittazione di tutti i file in una directory

Questo esempio decrittografa tutti i file in una directory. Inizia con i file nella directory TestEnc che sono stati crittografati nell'esempio precedente.

### 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`

Questo comando decrittografa tutti i file nella `TestEncdirectory` e scrive i file di testo normale nella `TestDecdirectory`. La `--wrapping-keys` parametro con `achiaveattributo` e [ARN della chiave](#) il valore dice il `AWSCLI` di crittografia che `AWS KMS keys` da utilizzare per decrittografare i file. Il comando utilizza l'`--interactive` parametro per dire al `AWSCLI` di crittografia per richiedere l'utente prima di sovrascrivere un file con lo stesso nome.

Questo comando utilizza inoltre il contesto di crittografia fornito quando i file sono stati crittografati. Quando esegui la decrittografia di più file, `AWS CLI` di crittografia verifica il contesto di crittografia di ogni file. Se il controllo del contesto di crittografia su qualsiasi file non riesce, `AWS CLI` di crittografia rifiuta il file, scrive un avviso, registra l'errore nei metadati, quindi continua a verificare gli altri file. Se il file `AWS CLI` di crittografia non riesce a decrittografare un file per qualsiasi altro motivo, l'intero comando di decrittografia ha esito negativo immediatamente.

In questo esempio, i messaggi crittografati in tutti i file di input contengono l'elemento del contesto di crittografia `dept=IT`. Tuttavia, se stai decrittografando messaggi con diversi contesti di crittografia, potresti comunque essere in grado di verificare parte del contesto di crittografia. Ad esempio, se alcuni messaggi avessero un contesto di crittografia di `dept=finance` e altri di `dept=IT`, potresti verificare che il contesto di crittografia contenga sempre un nome `dept` senza specificare il valore. Se volessi essere più specifico, potresti decrittografare i file in comandi distinti.

Il comando di decrittografia non restituisce alcun output, ma è possibile utilizzare un comando per visualizzare l'elenco delle directory e verificare se sono stati creati nuovi file con il suffisso `.decrypted`. Per vedere i contenuti di testo non crittografato, utilizza un comando per ottenere il contenuto del file.

## 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

```

## Crittografia e decrittazione nella riga di comando

Questi esempi illustrano come reindirizzare l'input per i comandi (stdin) e scrivere l'output nella riga di comando (stdout). Viene mostrato come rappresentare stdin e stdout in un comando e come utilizzare gli strumenti di codifica Base64 integrati per evitare che la shell interpreti non correttamente i caratteri non ASCII.

Questo esempio reindirizza una stringa di testo non crittografato a un comando di crittografia e salva il messaggio crittografato in una variabile. Quindi, reindirizza il messaggio crittografato nella variabile a un comando di decrittografia, che scrive l'output alla pipeline (stdout).

L'esempio è costituito da tre comandi:

- Il primo comando salva il [ARN della chiave](#) di un AWS KMS key nella `$keyArn` variabile.

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'
```

- Il secondo comando reindirizza la stringa `Hello World` al comando di crittografia e salva il risultato nella variabile `$encrypted`.

La `--input` e `--output` parametri sono obbligatori in tutti i comandi della CLI di crittografia. Per indicare che l'input è in fase di reindirizzamento al comando (stdin), utilizza un trattino (-) per il valore del parametro `--input`. Per inviare l'output alla riga di comando (stdout), utilizza un trattino per il valore del parametro `--output`.

Il parametro `--encode` codifica l'output con Base64 prima di restituirlo. In questo modo si impedisce alla shell di interpretare erroneamente i caratteri non ASCII nel messaggio crittografato.

Poiché questo comando è solo un proof of concept, abbiamo ommesso il contesto di crittografia e soppresso i metadati (`-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
```

- Il terzo comando reindirizza il messaggio crittografato nella variabile `$encrypted` al comando di decrittografia.

Questo comando di decrittografia utilizza `--input -` per indicare che l'input proviene dalla pipeline (stdin) e `--output -` per inviare l'output alla pipeline (stdout). (Il parametro di input richiede la posizione dell'input, non i byte effettivi dell'input, perciò non è possibile utilizzare la variabile `$encrypted` come valore del parametro `--input`).

In questo esempio viene utilizzato il `scoperta` attributo del `--wrapping-keys` parametro per consentire il `AWSCLI` di crittografia per utilizzare qualsiasi `AWS KMS key` per decrittografare i dati. Non specifica un [politica di impegno](#), quindi utilizza il valore predefinito per la versione 2.1.xe successivamente, `require-encrypt-require-decrypt`.

Poiché l'output è stato crittografato e poi codificato, il comando di decrittografia utilizza il parametro `--decode` per decodificare l'input codificato con Base64 prima di decrittografarlo. È inoltre possibile utilizzare il parametro `--decode` per decodificare l'input codificato con Base64 prima di crittografarlo.

Come in precedenza, il comando omette il contesto di crittografia e sopprime i metadati (`-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
```

È inoltre possibile eseguire le operazioni di crittografia e decrittografia in un singolo comando senza l'intervento della variabile.

Come nell'esempio precedente, i parametri `--input` e `--output` hanno un valore `-` e il comando utilizza il parametro `--encode` per codificare l'output e il parametro `--decode` per decodificare l'input.

## 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
```

## Utilizzo di più tasti master

Questo esempio illustra come utilizzare più chiavi master per crittografare e decrittografare i dati nellaAWSCLI di crittografia.

Quando si utilizzano più chiavi master per crittografare i dati, una qualsiasi delle chiavi master può essere utilizzata per decrittografare i dati. Questa strategia garantisce che tu possa decrittografare

i dati anche se una delle chiavi master non è disponibile. Se si memorizzano i dati crittografati in piùRegioni AWS, questa strategia ti consente di utilizzare una chiave master nella stessa regione per decrittografare i dati.

Quando esegui la crittografia con più chiavi master, la prima chiave master ricopre un ruolo speciale. Genera la chiave di dati utilizzata per crittografare i dati. Le restanti chiavi master crittografano la chiave dei dati di testo non crittografato. Il [messaggio crittografato](#) risultante include i dati crittografati e una raccolta di chiavi di dati crittografati, una per ciascuna chiave master. Anche se la prima chiave master ha generato la chiave dei dati, una delle chiavi master può decrittografare una delle chiavi dei dati, che può essere utilizzata per decrittografare i dati.

### Crittografia con tre chiavi master

Questo comando di esempio utilizza tre chiavi di wrapping per crittografare il `Finance.log` file, uno su treRegioni AWS.

Scrive il messaggio crittografato nella directory `Archive`. Il comando utilizza il parametro `--suffix` con nessun valore per sopprimere il suffisso, in modo che i nomi dei file di input e di output saranno gli stessi.

Il comando utilizza il parametro `--wrapping-keys` con tre attributi della chiave. È inoltre possibile utilizzare più parametri `--wrapping-keys` nello stesso comando.

Per crittografare il file di registro, AWSLa CLI di crittografia chiede la prima chiave di wrapping nell'elenco, `$key1`, per generare la chiave di dati utilizzata per crittografare i dati. Quindi, utilizza ognuna delle altre chiavi di wrapping per crittografare una copia non crittografata della stessa chiave di dati. Il messaggio crittografato nel file di output include tutte e tre le chiavi di dati crittografati.

### 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

```

Questo comando decrittografa la copia crittografata del file `Finance.log` e la scrive su un file `Finance.log.clear` nella directory `Finance`. Per decrittografare i dati crittografati sotto i tre AWS KMS keys, è possibile specificare gli stessi tre AWS KMS keys o qualsiasi sottoinsieme di essi. Questo esempio specifica uno dei AWS KMS keys.

Per raccontare il CLI di crittografia che AWS KMS keys utilizzare per decrittografare i dati, utilizzare l'attributo del `--wrapping-keys` Parametro. Quando esegui la decrittografia con AWS KMS keys, il valore dell'attributo deve essere un [ARN della chiave](#).

Devi disporre dell'autorizzazione per chiamare il [Decrittografare l'API](#) sul AWS KMS keys specificate. Per ulteriori informazioni, consulta [Autenticazione e controllo degli accessi per AWS KMS](#).

Come best practice, questi esempi utilizzano `--max-encrypted-data-keys` parametro per evitare di decrittografare un messaggio malformato con un numero eccessivo di chiavi dati crittografate. Anche se questo esempio utilizza una sola chiave wrapping per la decrittografia, il messaggio crittografato ha tre (3) chiavi dati crittografate; una per ciascuna delle tre chiavi di wrapping utilizzate durante la crittografia. Specificare il numero previsto di chiavi dati crittografate o un valore massimo ragionevole, ad esempio 5. Se si specifica un valore massimo inferiore a 3, il comando ha esito negativo. Per dettagli, consultare [Limitazione delle chiavi di dati crittografate](#).

## 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
```

## Crittografia e decrittazione negli script

In questo esempio viene illustrato come utilizzare l'AWSCLI di crittografia negli script. È possibile scrivere script solo per crittografare e decrittografare i dati o script che crittografano o decrittografano come parte di un processo di gestione dei dati.

In questo esempio, lo script ottiene una raccolta di file di log, li comprime, li crittografa, quindi copia i file crittografati in un bucket Amazon S3. Questo script elabora ciascun file separatamente, in modo che tu possa decrittografarli ed espanderli in modo indipendente.

Quando comprimi e crittografi i file, assicurati di eseguire la compressione prima della crittografia. I dati crittografati correttamente non sono comprimibili.

### Warning

Fai attenzione durante la compressione dei dati, che include sia segreti sia dati che potrebbero essere controllati da malintenzionati. Le dimensioni finali dei dati compressi potrebbero inavvertitamente rivelare informazioni sensibili sui contenuti.

## 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"))
            {

```



## Utilizzo del caching della chiave dei dati

Questo esempio utilizza il [caching della chiave dei dati](#) in un comando che crittografa un numero elevato di file.

Per impostazione predefinita, ilAWSCLI di crittografia (e altre versioni deAWS Encryption SDK) genera una chiave di dati univoca per ogni file che crittografa. Anche se utilizzare una chiave di dati univoca per ciascuna operazione è una best practice crittografica, il riutilizzo limitato delle chiavi dei dati è accettabile in alcune situazioni. Se stai pensando al caching della chiave dei dati, consulta un ingegnere che si occupa di sicurezza per comprendere i requisiti di sicurezza dell'applicazione e stabilire le relative soglie per le tue esigenze.

In questo esempio, il caching della chiave dei dati velocizza l'operazione di crittografia riducendo la frequenza delle richieste al provider della chiave master.

Il comando in questo esempio crittografa una directory di grandi dimensioni con più sottodirectory che contengono un totale di circa 800 piccoli file di log. Il primo comando salva l'ARN della AWS KMS key in una variabile `keyArn`. Il secondo comando crittografa tutti i file nella directory di input (in modo ricorsivo) e li scrive in una directory di archivio. Il comando utilizza il parametro `--suffix` per specificare il suffisso `.archive`.

Il parametro `--caching` consente di eseguire il caching della chiave dei dati. L'attributo di capacità, che limita il numero di chiavi di dati nella cache, è impostato su 1, perché l'elaborazione dei file di serie non impiega mai più di una chiave di dati alla volta. L'attributo `max_age`, che stabilisce per quanto tempo è possibile utilizzare la chiave di dati memorizzati nella cache, è impostato su 10 secondi.

L'attributo opzionale `max_messages_encrypted` è impostato su 10 messaggi, perciò una singola chiave dei dati non viene mai utilizzata per crittografare più di 10 file. Limitando il numero di file crittografati da ciascuna chiave di dati si riduce il numero di file interessati nell'improbabile caso in cui una chiave di dati sia stata compromessa.

Per eseguire questo comando nei file di log generati dal tuo sistema operativo, potresti aver bisogno delle autorizzazioni da amministratore (sudo in Linux; Esegui come amministratore in 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
```

Per verificare l'effetto della cache delle chiavi di dati, questo esempio utilizza [Misura-comando](#) cmdlet in PowerShell. Quando esegui questo esempio senza il caching della chiave dei dati, sono necessari circa 25 secondi per il completamento. Questo processo genera una nuova chiave di dati per ciascun file nella directory.

```
PS C:\> Measure-Command {aws-encryption-cli --encrypt `
    --input C:\Windows\Logs --recursive `
    --output $home\Archive --suffix '.archive' `
    --wrapping-keys key=$keyARN `
    --encryption-context class=log `
    --suppress-metadata }

Days           : 0
Hours          : 0
Minutes       : 0
Seconds       : 25
Milliseconds  : 453
```

```

Ticks           : 254531202
TotalDays       : 0.000294596298611111
TotalHours      : 0.00707031116666667
TotalMinutes    : 0.42421867
TotalSeconds    : 25.4531202
TotalMilliseconds : 25453.1202

```

Il caching della chiave di dati rende il processo più rapido, anche quando limiti ogni chiave di dati a un massimo di 10 file. Il comando ora impiega meno di 12 secondi per il completamento e riduce il numero di chiamate al provider della chiave master a 1/10 del valore originale.

```

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   : 0.1968877333333333
TotalSeconds   : 11.813264
TotalMilliseconds : 11813.264

```

Se elimini la restrizione `max_messages_encrypted`, tutti i file sono crittografati con la stessa chiave dei dati. Questa modifica aumenta il rischio di riutilizzare le chiavi dei dati senza accelerare il processo. Tuttavia, riduce il numero di chiamate al provider di chiavi master a 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
```

## Sintassi e riferimento parametri dell'interfaccia a riga di comando AWS Encryption SDK

Questo argomento fornisce diagrammi di sintassi e brevi descrizioni dei parametri per aiutarti a utilizzare l'interfaccia a riga di comando (CLI) AWS Encryption SDK. Per informazioni sulla disposizione delle chiavi e su altri parametri, consulta [Come usare la CLI di AWS crittografia](#). Per alcuni esempi, consulta [Esempi diAWSCLI di crittografia](#). Per la documentazione completa, consulta [Leggi i documenti](#).

### Argomenti

- [AWSSintassi CLI di crittografia](#)
- [AWSParametri della riga di comando della crittografia](#)
- [Parametri avanzati](#)

### AWSSintassi CLI di crittografia

Questi diagrammi di sintassi della CLI diAWS crittografia mostrano la sintassi per ogni attività eseguita con la CLI diAWS crittografia. Rappresentano la sintassi consigliata nella versione 2.1 dell'interfaccia a riga di comando diAWS crittografia. x e versioni successive.

Le nuove funzionalità di sicurezza sono state originariamente rilasciate nelle versioni 1.7 dell'interfaccia a riga di comando di AWS crittografia. x e 2.0. x. Tuttavia, AWS Encryption CLI versione 1.8. x sostituisce la versione 1.7. x e AWS Encryption CLI 2.1. x sostituisce 2.0. x. Per i dettagli, [consulta l'avviso di sicurezza](#) pertinente nel [aws-encryption-sdk-cli repository](#) su GitHub.

### Note

A meno che non sia indicato nella descrizione del parametro, ogni parametro o attributo può essere utilizzato una sola volta in ogni comando.

Se si utilizza un attributo non supportato da un parametro, la CLI di AWS crittografia ignora l'attributo non supportato senza alcun avviso o errore.

## Chiedere aiuto

Per ottenere la sintassi completa della CLI di AWS crittografia con le descrizioni dei parametri, usa `--help` or `-h`.

```
aws-encryption-cli (--help | -h)
```

## Ottenere la versione

Per ottenere il numero di versione dell'installazione della CLI di AWS crittografia, usa `--version`. Assicurati di includere la versione quando fai domande, segnali problemi o condividi suggerimenti sull'uso della CLI di AWS crittografia.

```
aws-encryption-cli --version
```

## Crittografare i dati

Il seguente diagramma di sintassi mostra i parametri utilizzati da un 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]

```

## Decrittare i dati

Il seguente diagramma di sintassi mostra i parametri utilizzati da un comando `decrypt`.

Nella versione 1.8. x, il `--wrapping-keys` parametro è facoltativo durante la decrittografia, ma consigliato. A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è necessario per la crittografia e la decrittografia. Infatti AWS KMS keys, è possibile utilizzare l'attributo `key` per specificare le chiavi di wrapping (procedura consigliata) o impostare l'attributo `discovery` su `true`, il che non limita le chiavi di wrapping che la CLI di AWS crittografia può utilizzare.

```

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]

```

## Utilizzare i file di configurazione

Puoi consultare i file di configurazione che contengono i parametri e i relativi valori. Ciò equivale a digitare i parametri e i valori nel comando. Per un esempio, consulta [Come archiviare i parametri in un file di configurazione](#).

```
aws-encryption-cli @<configuration_file>

# In a PowerShell console, use a backtick to escape the @.
aws-encryption-cli `@<configuration_file>
```

## AWSParametri della riga di comando della crittografia

Questo elenco fornisce una descrizione di base dei parametri del comando AWS Encryption CLI. Per una descrizione completa, consulta la [aws-encryption-sdk-clidocumentazione](#).

### --crittografia (-e)

Crittografia dei dati di input. Ogni comando deve avere un `--decrypt-unsigned` parametro `--encrypt` o `--decrypt`, o o.

### --decrittografia (-d)

Decrittografia dei dati di input. Ogni comando deve avere un `--decrypt-unsigned` parametro `--encrypt` o `--decrypt`, o o.

### --decrypt-unsigned [Introdotta nelle versioni 1.9. x e 2.2. x]

Il `--decrypt-unsigned` parametro decrittografa il testo cifrato e garantisce che i messaggi non siano firmati prima della decrittografia. Utilizza questo parametro se hai utilizzato il `--algorithm` parametro e hai selezionato una suite di algoritmi senza firma digitale per crittografare i dati. Se il testo cifrato è firmato, la decrittografia fallisce.

È possibile utilizzare `--decrypt` o `--decrypt-unsigned` per la decrittografia, ma non entrambi.

### --wrapping-keys (-w) [Introdotta nella versione 1.8. x]

Specifica le [chiavi di wrapping](#) (o chiavi master) utilizzate nelle operazioni di crittografia e decrittografia. È possibile utilizzare [più --wrapping-keys parametri](#) in ogni comando.

A partire dalla versione 2.1. x, il `--wrapping-keys` parametro è obbligatorio nei comandi di crittografia e decrittografia. Nella versione 1.8. x, i comandi di crittografia richiedono un `--master-keys` parametro `--wrapping-keys` or. Nella versione 1.8. x decrypt i comandi, un `--wrapping-keys` parametro è facoltativo ma consigliato.

Quando si utilizza un provider di chiave master personalizzato, i comandi di crittografia e decrittografia richiedono gli attributi della chiave e del provider. Quando si utilizzano AWS KMS keys, i comandi di crittografia richiedono un attributo chiave. I comandi di decrittografia richiedono un attributo chiave o un attributo discovery con un valore pari a `true` (ma non entrambi). L'utilizzo dell'attributo `key` durante la decrittografia è una [procedura AWS Encryption SDK consigliata](#). È particolarmente importante se stai decifrando batch di messaggi sconosciuti, come quelli in un bucket Amazon S3 o in una coda Amazon SQS.

Per un esempio che mostra come utilizzare le chiavi AWS KMS multiregione come chiavi di confezionamento, vedere [Utilizzo di più regioni AWS KMS keys](#).

Attributi: il valore del parametro `--wrapping-keys` consiste nei seguenti attributi. Il formato è `attribute_name=value`.

`key`

Identifica la chiave di wrapping utilizzata nell'operazione. Il formato è una coppia chiave=ID. È possibile specificare più attributi delle chiavi in ogni valore del parametro `--wrapping-keys`.

- Comandi di crittografia: tutti i comandi di crittografia richiedono l'attributo `key`. Quando utilizzi un comando AWS KMS key in un ciasas, il valore dell'attributo della chiave, l'ARN della chiave, il nome della chiave, il nome della chiave, il nome della chiave, il nome della chiave, l'ARN della chiave, il nome della chiave, il nome della chiave, l'ARN della chiave, il nome della chiave, il nome della chiave Per le descrizioni degli identificatori AWS KMS chiave, consulta [Identificatori chiave](#) nella Guida per gli AWS Key Management Service sviluppatori.
- Comandi di decrittografia: quando si decodifica con AWS KMS keys, il `--wrapping-keys` parametro richiede un attributo chiave [con un valore ARN](#) chiave o un attributo discovery con un valore di `true` (ma non entrambi). L'utilizzo dell'attributo chiave è una [procedura AWS Encryption SDK consigliata](#). Quando si esegue la decrittografia con un provider di chiave master personalizzato, è necessario l'attributo `key`.

**Note**

Per specificare una chiave di AWS KMS wrapping in un comando di decrittografia, il valore dell'attributo `key` deve essere un ARN chiave. Se utilizzi un ID chiave, un nome o un ARN dell'alias, la CLI della AWS crittografia non riconoscerà la chiave di `Writer`.

È possibile specificare più attributi delle chiavi in ogni valore del parametro `--wrapping-keys`. Tuttavia, qualsiasi attributo di provider, regione e profilo in un `--wrapping-keys` parametro si applica a tutte le chiavi di wrapping in quel valore del parametro. Per specificare chiavi di wrapping con valori di attributo diversi, utilizzate più `--wrapping-keys` parametri nel comando.

**scoperta**

Consente alla CLI di AWS crittografia di utilizzarne una qualsiasi AWS KMS key per decrittografare il messaggio. Il valore di `scoperta` può essere `true` o `false`. Il valore di default è `false`. L'attributo `discovery` è valido solo nei comandi di decrittografia e solo quando lo è il provider della chiave master AWS KMS.

Quando si decodifica con AWS KMS keys, il `--wrapping-keys` parametro richiede un attributo chiave o un attributo di scoperta con un valore di `true` (ma non entrambi). Se si utilizza l'attributo `key`, è possibile utilizzare un attributo `discovery` con un valore di `false` per rifiutare esplicitamente la scoperta.

- `False` (impostazione predefinita) — Quando l'attributo `discovery` non è specificato o il suo valore lo è `false`, la CLI di AWS crittografia decrittografa il messaggio utilizzando solo AWS KMS keys quanto specificato dall'attributo chiave del `--wrapping-keys` parametro. Se non si specifica un attributo chiave durante il rilevamento `false`, il comando `decrypt` ha esito negativo. Questo valore supporta una [best practice](#) della CLI di AWS crittografia.
- `True` — Quando il valore dell'attributo `discovery` è `true`, la CLI di AWS crittografia ottiene i metadati AWS KMS keys dal messaggio crittografato e li utilizza AWS KMS keys per decrittografare il messaggio. L'attributo `discovery` con un valore pari a `true` si comporta come le versioni della CLI di AWS crittografia precedenti alla versione 1.8. x che non permetteva di specificare una chiave di wrapping durante la decrittografia. Tuttavia, la tua intenzione di utilizzarne una AWS KMS key è esplicita. Se si specifica un attributo chiave durante il rilevamento `true`, il comando `decrypt` ha esito negativo.

Il `true` valore potrebbe far sì che la CLI di AWS crittografia AWS KMS keys venga utilizzata in diverse Account AWS aree geografiche o tentare di utilizzare AWS KMS keys qualcosa che l'utente non è autorizzato a utilizzare.

In caso di rilevamento `true`, è consigliabile utilizzare gli attributi `discovery-partition` e `discovery-account` per limitare l'uso di AWS KMS keys a quelli Account AWS specificati.

#### Reader di individuazione

Limita l'uso di AWS KMS keys per la decrittografia a quelli specificati Account AWS. L'unico valore valido per questo attributo è un [Account AWS ID](#).

Questo attributo è facoltativo e valido solo nei comandi di decrittografia in AWS KMS keys cui l'attributo `discovery` è impostato su `true` e viene specificato l'attributo `discovery-partition`.

Ogni attributo `discovery-account` richiede un solo Account AWS ID, ma puoi specificare più attributi `discovery-account` nello stesso `--wrapping-keys` parametro. Tutti gli account specificati in un determinato `--wrapping-keys` parametro devono trovarsi nella AWS partizione specificata.

#### Reader di individuazione

Specifica la AWS partizione per gli account nell'attributo `discovery-account`. Il suo valore deve essere una AWS partizione, ad esempio `aws-cn`, `aws-gov-cloud`. Per informazioni, consulta [Amazon Resource Names](#) nei Riferimenti generali di AWS.

Questo attributo è obbligatorio quando si utilizza l'attributo `discovery-account`. È possibile specificare un solo attributo `discovery-partition` in ogni `--wrapping-keys` parametro. Per specificare Account AWS in più partizioni, utilizzare un `--wrapping-keys` parametro aggiuntivo.

#### provider

Identifica il [provider della chiave master](#). Il formato è una coppia `provider=ID`. Il valore di default, `aws-kms`, rappresenta AWS KMS. Questo attributo è obbligatorio solo quando il provider della chiave master non è AWS KMS.

#### Regione

Identifica la Regione AWS di un AWS KMS key. Questo attributo è valido solo per AWS KMS keys. È utilizzato solo quando l'identificatore della chiave non specifica una regione. In caso contrario, verrà ignorato. Quando viene utilizzato, sovrascrive la regione predefinita nel profilo denominato dell'interfaccia a riga di comando AWS.

## profile

Identifica un AWS CLI denominato profilo <https://docs.aws.amazon.com/cli/latest/userguide/cli-multiple-profiles.html>. Questo attributo è valido solo per AWS KMS keys. La regione nel profilo è utilizzata solo quando l'identificatore chiave non consente di specificare una regione e non è previsto alcun attributo regione nel comando.

## --input (-i)

Specifica la posizione dei dati da crittografare o decrittografare. Questo parametro è obbligatorio. Il valore può essere un percorso a un file o a una directory o un modello di nome di file. Se stai reindirizzando input al comando (stdin), utilizza `-`.

Se l'input non esiste, il comando viene completato correttamente, senza errori o avvertenze.

## --recursive (-r, -R)

Esegue l'operazione sul file nella directory di input e nelle relative sottodirectory. Questo parametro è obbligatorio quando il valore di `--input` è una directory.

## --decode

Decodifica input codificati Base64.

Se stai decrittografando un messaggio che è stato crittografato e quindi codificato, è necessario decodificare il messaggio prima di decrittografarlo. Questo parametro lo fa per te.

Ad esempio, se utilizzi il parametro `--encode` in un comando di crittografia, utilizza il parametro `--decode` nel comando di decrittografia corrispondente. È inoltre possibile utilizzare questo parametro per decodificare l'input codificati Base64 prima di crittografarlo.

## --output (-o)

Specifica una destinazione per l'output. Questo parametro è obbligatorio. Il valore può essere un nome di file, una directory esistente oppure `-`, che scrive l'output sulla riga di comando (stdout).

Se la directory di output specificata non esiste, il comando ha esito negativo. Se l'input contiene sottodirectory, la CLI di AWS crittografia riproduce le sottodirectory nella directory di output specificata.

Per impostazione predefinita, la CLI di AWS crittografia sovrascrive i file con lo stesso nome. Per modificare questo comportamento, utilizza i parametri `--interactive` o `--no-overwrite`. Per annullare l'avvertenza relativa alla sovrascrittura, utilizza il parametro `--quiet`.

**Note**

Se un comando che sovrascrive un file di output ha esito negativo, il file di output viene eliminato.

**--interattivo**

Chiede prima di sovrascrivere il file.

**--nessuna sovrascrittura**

Non sovrascrive i file. Invece, se il file di output esiste, l'AWSEncryption CLI ignora l'input corrispondente.

**--suffisso**

Specifica un suffisso di nome file personalizzato per i file creati dalla CLI diAWS crittografia. Per indicare l'assenza di un suffisso, utilizza il parametro con nessun valore (`--suffix`).

Per impostazione predefinita, quando il parametro `--output` non specifica un nome di file, il nome del file di output ha lo stesso nome del nome del file di input più il suffisso. Il suffisso per i comandi di crittografia è `.encrypted`. Il suffisso per i comandi di decrittografia è `.decrypted`.

**--encode**

Applica la codifica Base64 (da binario a testo) all'output. La codifica evita che il programma host shell interpreti erroneamente i caratteri non ASCII nel testo di output.

Utilizzate questo parametro quando scrivete un output crittografato su stdout (`--output -`), specialmente in una PowerShell console, anche quando reindirizzate l'output a un altro comando o lo salvate in una variabile.

**--metadata-output**

Specifica una posizione per i metadati relativi alle operazioni di crittografia. Inserisci un percorso e un nome di file. Se la directory non esiste, il comando ha esito negativo. Per scrivere i metadati nella riga di comando (stdout), utilizza `-`.

Non è possibile scrivere l'output di comando (`--output`) e l'output dei metadati (`--metadata-output`) su stdout nello stesso comando. Inoltre, quando il valore di `--input` o di `--output`

è una directory (senza nomi di file), non è possibile scrivere l'output dei metadati nella stessa directory o in qualsiasi sottodirectory di tale directory.

Se si specifica un file esistente, per impostazione predefinita, la CLI diAWS crittografia aggiunge nuovi record di metadati a qualsiasi contenuto del file. Questa funzione consente di creare un singolo file che contiene i metadati per tutte le operazioni di crittografia. Per sovrascrivere i contenuti in un file esistente, utilizza il parametro `--overwrite-metadata`.

La CLI diAWS crittografia restituisce un record di metadati in formato JSON per ogni operazione di crittografia o decrittografia eseguita dal comando. Ogni record dei metadati include i percorsi completi al file di input e di output, il contesto di crittografia, la suite di algoritmi e altre informazioni utili che puoi utilizzare per rivedere l'operazione e verificare che soddisfi gli standard di sicurezza.

`--overwrite-metadata`

Sovrascrive i contenuti nel file di output dei metadati. Per impostazione predefinita, il parametro `--metadata-output` aggiunge i metadati a qualsiasi contenuto esistente nel file.

`--suppress-metadata (-S)`

Sopprime i metadati relativi all'operazione di crittografia o decrittografia.

`--policy-impegno`

Specifica la [politica di impegno](#) per i comandi di crittografia e decrittografia. La politica di impegno determina se il messaggio è crittografato e decrittografato con la funzionalità di sicurezza [chiave dell'impegno](#).

Il `--commitment-policy` parametro è stato introdotto nella versione 1.8. x. È valido nei comandi di crittografia e decrittografia.

Nella versione 1.8. x, la CLI diAWS crittografia utilizza la politica di `forbid-encrypt-allow-decrypt` impegno per tutte le operazioni di crittografia e decrittografia. Quando si utilizza il `--wrapping-keys` parametro in un comando di crittografia o decrittografia, è necessario un `--commitment-policy` parametro con il `forbid-encrypt-allow-decrypt` valore. Se non si utilizza il `--wrapping-keys` parametro, il `--commitment-policy` parametro non è valido. L'impostazione di una politica di impegno impedisce esplicitamente che la politica di impegno venga modificata automaticamente `require-encrypt-require-decrypt` quando si esegue l'aggiornamento alla versione 2.1. x

A partire dalla versione 2.1. x, tutti i valori della politica di impegno sono supportati. Il `--commitment-policy` parametro è facoltativo e il valore predefinito è `require-encrypt-require-decrypt`.

Questo parametro ha i seguenti valori:

- `forbid-encrypt-allow-decrypt`— Non è possibile crittografare con impegno chiave. Può decrittografare i testi cifrati con o senza impegno chiave.

Nella versione 1.8. x, questo è l'unico valore valido. La CLI diAWS crittografia utilizza la politica di `forbid-encrypt-allow-decrypt` impegno per tutte le operazioni di crittografia e decrittografia.

- `require-encrypt-allow-decrypt`— Crittografia solo con impegno chiave. Decifra con e senza impegno chiave. Questo valore è stato introdotto nella versione 2.1. x.
- `require-encrypt-require-decrypt`(impostazione predefinita): crittografia e decrittografia solo con l'impegno della chiave. Questo valore è stato introdotto nella versione 2.1. x. È il valore predefinito nelle versioni 2.1. x e versioni successive. Con questo valore, la CLI diAWS crittografia non decrittograferà alcun testo cifrato con versioni precedenti diAWS Encryption SDK.

Per informazioni dettagliate sull'impostazione di impostazione di impegno, consulta [Migrazione della tuaAWS Encryption SDK](#).

`--encryption-context (-c)`

Specifica un [contesto di crittografia](#) per l'operazione. Questo parametro non è obbligatorio, ma è consigliato.

- In un comando `--encrypt`, immetti una o più coppie `name=value`. Utilizza gli spazi per separare le coppie.
- In un `--decrypt` comando, immetti `name=value` coppie, `name` elementi senza valori o entrambi.

Se `name` o `value` in una coppia `name=value` include spazi o caratteri speciali, racchiudi la coppia completa tra virgolette. Ad esempio, `--encryption-context "department=software development"`.

`--buffer (-b)` [Introdotto nelle versioni 1.9. x e 2.2. x]

Restituisce il testo in chiaro solo dopo l'elaborazione di tutti gli input, inclusa la verifica della firma digitale, se presente.

`--max-encrypted-data-keys` [Introdotto nelle versioni 1.9. x e 2.2. x]

Specifica il numero massimo di chiavi di dati crittografate in un messaggio crittografato. Questo parametro è facoltativo.

I valori validi sono compresi tra 1 e 65.535. Se ometti questo parametro, la CLI diAWS Encryption non applicherà alcun valore. Un messaggio crittografato può contenere fino a 65.535 ( $2^{16} - 1$ ) chiavi di dati crittografate.

È possibile utilizzare questo parametro nei comandi di crittografia per evitare un messaggio malformato. Puoi usarlo nei comandi di decrittografia per rilevare messaggi dannosi ed evitare di decrittografare i messaggi con numerose chiavi di dati crittografate che non puoi decrittografare. Per informazioni dettagliate e un esempio, consulta [Limitazione delle chiavi di dati crittografate](#).

`--help (-h)`

Stampa utilizzo e sintassi nella riga di comando.

`--versione`

Ottiene la versione della CLI diAWS crittografia.

`-v | -vv | -vvv | -vvvv`

Visualizza informazioni, avvisi e messaggi di debug verbosi. Il dettaglio nell'output aumenta con il numero di v nel parametro. L'impostazione più dettagliata (`-vvvv`) restituisce i dati a livello di debug dalla CLI diAWS crittografia e da tutti i componenti che utilizza.

`--quiet (-q)`

Sopprime messaggi di avviso, ad esempio il messaggio visualizzato quando si sovrascrive un file di output.

`--master-keys (-m) [Obsoleto]`

#### Note

Il parametro `--master-keys` è obsoleto nella versione 1.8. x e rimosso nella versione 2.1. x. Utilizzate invece il parametro [--wrapping-keys](#).

Specifica le [chiavi master](#) utilizzate nelle operazioni di crittografia e decrittografia. È possibile utilizzare i parametri di più chiavi master in ogni comando.

Il parametro `--master-keys` è obbligatorio nei comandi di crittografia. È necessaria nei comandi di decrittografia solo quando si utilizza un provider di chiavi master (nonAWS KMS) personalizzate.

Attributi: il valore del parametro `--master-keys` consiste nei seguenti attributi. Il formato è `attribute_name=value`.

### key

Identifica la [chiave di wrapping](#) utilizzata nell'operazione. Il formato è una coppia chiave=ID. L'attributo chiave è obbligatorio in tutti i comandi di crittografia.

Quando utilizzi un comando `AWS KMS key` in un ciascas, il valore dell'attributo della chiave, l'ARN della chiave, il nome della chiave, il nome della chiave, il nome della chiave, il nome della chiave, l'ARN della chiave, il nome della chiave, il nome della chiave, il nome della chiave Per dettagli sugli identificatori `AWS KMS` chiave, consulta [Identificatori chiave](#) nella Guida per gli `AWS Key Management Service` sviluppatori.

L'attributo `key` è obbligatorio nei comandi di decrittografia quando il fornitore della chiave master non lo è `AWS KMS`. L'attributo `key` non è consentito nei comandi che decrittografano i dati crittografati con un `AWS KMS key`.

È possibile specificare più attributi delle chiavi in ogni valore del parametro `--master-keys`. Tuttavia, qualsiasi attributo di provider, regione e profilo si applica a tutte le chiavi master nel valore del parametro. Per specificare le chiavi master con differenti valori degli attributi, utilizza più parametri `--master-keys` nel comando.

### provider

Identifica il [provider della chiave master](#). Il formato è una coppia provider=ID. Il valore di default, `aws-kms`, rappresenta `AWS KMS`. Questo attributo è obbligatorio solo quando il provider della chiave master non è `AWS KMS`.

### Regione

Identifica il Regione `AWS` di un `AWS KMS key`. Questo attributo è valido solo per `AWS KMS keys`. È utilizzato solo quando l'identificatore della chiave non specifica una regione. In caso contrario, verrà ignorato. Quando viene utilizzato, sovrascrive la regione predefinita nel profilo denominato dell'interfaccia a riga di comando `AWS`.

### profile

Identifica un `AWS CLI` denominato profilo <https://docs.aws.amazon.com/cli/latest/userguide/cli-multiple-profiles.html>. Questo attributo è valido solo per `AWS KMS keys`. La regione nel profilo è utilizzata solo quando l'identificatore chiave non consente di specificare una regione e non è previsto alcun attributo regione nel comando.

## Parametri avanzati

### --algorithm

Specifica una [suite di algoritmi](#) alternativa. Questo parametro è facoltativo ed è valido solo nei comandi di crittografia.

Se si omette questo parametro, l'AWSEncryption CLI utilizza una delle suite di algoritmi predefinite per quelle AWS Encryption SDK introdotte nella versione 1.8. x. Entrambi gli algoritmi predefiniti utilizzano AES-GCM con un [HKDF](#), una firma ECDSA e una chiave di crittografia a 256 bit. Uno usa l'impegno chiave; l'altro no. La scelta della suite di algoritmi predefinita è determinata dalla [politica di impegno](#) per il comando.

Le suite di algoritmi predefinite sono consigliate per la maggior parte delle operazioni di crittografia. Per un elenco dei valori validi, consulta i valori per il parametro `algorithm` in [Leggi i documenti](#).

### --frame-length

Crea output con una lunghezza frame specificata. Questo parametro è facoltativo ed è valido solo nei comandi di crittografia.

Inserisci un valore in byte. I valori validi sono 0 e  $1 - 2^{31} - 1$ . Un valore pari a 0 indica i dati senza cornice. L'impostazione predefinita è 4096 (byte).

#### Note

Quando possibile, usa dati incorniciati. AWS Encryption SDK supporta dati senza frame solo per uso legacy. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza cornice. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza cornice.

### --max-length

Indica la dimensione massima del frame (o la lunghezza massima dei contenuti per messaggi non framed) in byte per leggere i messaggi crittografati. Questo parametro è facoltativo ed è valido solo nei comandi di decrittografia. È concepito per proteggerti dalla decrittografia di testo cifrato dannoso di grandi dimensioni.

Inserisci un valore in byte. Se ometti questo parametro, AWS Encryption SDK non limiterà la dimensione del frame durante la decodifica.

--caching

Abilita la funzionalità di [caching della chiave dei dati](#), che riutilizza le chiavi di dati, invece di generare una nuova chiave di dati per ogni file di input. Questo parametro supporta uno scenario avanzato. Assicurati di leggere la documentazione [Caching della chiave dei dati](#) prima di utilizzare questa funzionalità.

Il parametro --caching ha i seguenti attributi.

capacità (obbligatorio)

Stabilisce il numero massimo di voci nella cache.

Il valore minimo è 1. Non è previsto un valore massimo.

max\_age (obbligatorio)

Determina per quanto tempo vengono utilizzate le voci della cache, in secondi, a partire da quando vengono aggiunte alla cache.

Immetti un valore superiore a 0. Non è previsto un valore massimo.

max\_messages\_encrypted (opzionale)

Stabilisce il numero massimo di messaggi che una voce nella cache è in grado di crittografare.

I valori validi sono compresi tra 1 e  $2^{32}$ . Il valore predefinito è  $2^{32}$  (messaggi).

max\_bytes\_encrypted (opzionale)

Stabilisce il numero massimo di byte che una voce nella cache è in grado di crittografare.

I valori validi sono 0 e  $1 - 2^{63} - 1$ . Il valore predefinito è  $2^{63} - 1$  (messaggi). Il valore 0 consente di utilizzare il caching della chiave di dati solo quando stai crittografando stringhe di messaggio vuote.

## Versioni dell'AWS Crittografia CLI

È consigliabile utilizzare la versione più recente della AWS Crittografia CLI.

### Note

Versioni dell'AWS Encryption CLI precedenti alla 4.0.0 sono nella CLI di [end-of-support fase](#). È possibile eseguire l'aggiornamento dalla versione 2.1.xe successivamente alla versione più recente di AWS Crittografia CLI senza modifiche al codice o ai dati. Tuttavia, [nuove funzioni di sicurezza](#) introdotte nella versione 2.1.x non sono compatibili con le versioni precedenti. Per eseguire l'aggiornamento dalla versione 1.7.xo versioni precedenti, è necessario eseguire l'aggiornamento alla versione più recente.x versione dell'AWS Crittografia CLI. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Le nuove funzionalità di sicurezza sono state inizialmente rilasciate in AWS Encryption CLI versioni 1.7.xe 2.0.x. Tuttavia, AWS Crittografia CLI versione 1.8.x sostituisce la versione 1.7.xe AWS Crittografia CLI 2.1.x sostituita 2.0.x. Per informazioni dettagliate, consulta la sezione relativa [consulenza di sicurezza](#) nel [aws-encryption-sdk-cli](#) repository su GitHub.

Per informazioni sulle versioni significative di AWS Encryption SDK, consulta [Versioni del AWS Encryption SDK](#).

Quale versione devo usare?

Se non hai mai usato la AWS CLI di crittografia, utilizza la versione più recente.

Per decifrare i dati crittografati da una versione di AWS Encryption SDK versioni precedenti alla 1.7.x, eseguire la migrazione alla versione più recente della AWS Crittografia CLI. Make [tutte le modifiche consigliate](#) prima dell'aggiornamento alla versione 2.1.xo dopo. Per dettagli, consulta [Migrazione della tua AWS Encryption SDK](#).

Ulteriori informazioni

- Per informazioni dettagliate sulle modifiche e sulle linee guida per la migrazione a queste nuove versioni, consulta [Migrazione della tua AWS Encryption SDK](#).
- Per le descrizioni del nuovo AWS Parametri e attributi della CLI di crittografia, vedere [Sintassi e riferimento parametri dell'interfaccia a riga di comando AWS Encryption SDK](#).

Gli elenchi seguenti descrivono la modifica dell'AWS Encryption CLI nelle versioni 1.8.xe 2.1.x.

### Versione 1.8.x modifiche alla AWS Crittografia CLI

- Obsoleta il `--master-keys` parametro. Utilizza invece il parametro `--wrapping-keys`.

- Aggiunta il `--wrapping-keys(-wparametro)`. Supporta tutti gli attributi del `--master-keysparametro`. Aggiunge anche i seguenti attributi opzionali, validi solo quando si decrittografa con AWS KMS keys.
  - scoperta
  - individuazione della partizione
  - conto di individuazione

Per i fornitori di chiavi master personalizzate, `--encrypt` `--decrypt` comandi richiedono un `--wrapping-keysparametro` o `--master-keysparametro` (ma non entrambi). Inoltre, un `--encrypt` comando con AWS KMS keys richiede un `--wrapping-keysparametro` o `--master-keysparametro` (ma non entrambi).

In un `--decrypt` comando con AWS KMS keys, il `--wrapping-keys` parametro è facoltativo, ma consigliato, perché è richiesto nella versione 2.1.x. Se lo si utilizza, è necessario specificare `o.chiaveattributo` o `ilscopertaattributo` con un valore di `true` (ma non entrambi).

- Aggiunta il `--commitment-policyparametro`. L'unico valore valido è `forbid-encrypt-allow-decrypt`. Il `forbid-encrypt-allow-decrypt` politica di impegno viene utilizzata in tutti i comandi di crittografia e decrittografia.

Nella versione 1.8.x, quando si utilizza il `--wrapping-keysparametro`, `--commitment-policyparametro` con `forbid-encrypt-allow-decrypt` il valore è obbligatorio. L'impostazione del valore impedisce esplicitamente [la politica di impegno](#) dalla modifica automatica `require-encrypt-require-decrypt` quando si esegue l'aggiornamento alla versione 2.1.x.

## Versione 2.1.x modifiche al AWS Criptazione CLI

- Rimozione del `--master-keysparametro`. Utilizza invece il parametro `--wrapping-keys`.
- Il `--wrapping-keys` parametro è richiesto in tutti i comandi di crittografia e decrittografia. È necessario specificare `o.chiaveattributo` o `ilscopertaattributo` con un valore di `true` (ma non entrambi).
- Il `--commitment-policyparametro` supporta i seguenti valori. Per dettagli, consulta [Impostazione della politica di impegno](#).
  - `forbid-encrypt-allow-decrypt`
  - `require-encrypt-allow-decrypt`
  - `require-encrypt-require-decrypt` L'impostazione predefinita.

- Il `--commitment-policy` parametro è opzionale nella versione 2.1.x. Il valore di default è `require-encrypt-require-decrypt`.

### Versione 1.9.x e 2.2.x modifiche al AWS Criptazione CLI

- Aggiunta il `--decrypt-unsigned` parametro. Per dettagli, consulta [Versione 2.2. x](#).
- Aggiunta il `--buffer` parametro. Per dettagli, consulta [Versione 2.2. x](#).
- Aggiunta il `--max-encrypted-data-keys` parametro. Per dettagli, consulta [Limitazione delle chiavi di dati crittografate](#).

### Versione 3.0.x modifiche al AWS Criptazione CLI

- Aggiunge il supporto per AWS KMS Chiavi multi-regione. Per informazioni dettagliate, consulta [Utilizzo di più regioni AWS KMS keys](#).

# Caching della chiave dei dati

Il caching della chiave di dati memorizza le [chiavi di dati](#) e i [relativi materiali crittografici](#) in una cache. Quando crittografi o decripti i dati, AWS Encryption SDK cerca una chiave di dati corrispondente nella cache. Se trova una corrispondenza, utilizza la chiave di dati memorizzata nella cache anziché generarne una nuova. Il caching della chiave di dati è in grado di migliorare le prestazioni, ridurre i costi e consentirti di rimanere entro limiti di servizio durante il dimensionamento dell'applicazione.

La tua applicazione è in grado di sfruttare i vantaggi offerti dal caching della chiave dei dati se:

- Può riutilizzare le chiavi di dati.
- Genera numerose chiavi di dati.
- Le operazioni di crittografia sono troppo lente, costose, limitate o utilizzano una quantità eccessiva di risorse.

Il caching può ridurre il tuo utilizzo di servizi di crittografia, come, ad esempio AWS Key Management Service (AWS KMS). Se stai raggiungendo il [AWS KMS requests-per-second limite](#), la memorizzazione nella cache può aiutarti. La tua applicazione può utilizzare le chiavi memorizzate nella cache per soddisfare alcune delle richieste delle chiavi di dati invece di chiamare AWS KMS. (Puoi anche creare un caso nel [AWSSupport Center](#) per aumentare il limite per il tuo account.)

AWS Encryption SDK consente di creare e gestire la cache della chiave di dati. [Fornisce una cache locale e un gestore di materiali crittografici per la memorizzazione nella cache \(memorizzazione nella cache CMM\) che interagisce con la cache e applica le soglie di sicurezza impostate dall'utente.](#) Lavorando insieme, questi componenti consentono di trarre vantaggio dall'efficienza di riutilizzare le chiavi dei dati, mantenendo, al contempo, la sicurezza del sistema.

Il caching della chiave di dati è una caratteristica opzionale di AWS Encryption SDK che devi usare con cautela. Per impostazione predefinita, AWS Encryption SDK genera una nuova chiave di dati per ogni operazione di crittografia. Questa tecnica supporta le best practice di crittografia, che scoraggiano il riutilizzo eccessivo delle chiavi di dati. In generale, è possibile usare il caching delle chiavi dei dati solo quando è necessario per soddisfare gli obiettivi di performance. Quindi, utilizza le [soglie di sicurezza](#) del caching della chiave di dati per assicurarti di utilizzare la quantità minima di caching necessaria per soddisfare i tuoi obiettivi di costi e prestazioni.

[La memorizzazione nella cache delle chiavi di dati non è supportata da for .NET.](#) AWS Encryption SDK Versione 3. x of the SDK di crittografia AWS per Java depreca la CMM con memorizzazione

nella cache. Tuttavia, la versione 4. x di AWS Encryption SDK per .NET e versione 3. x del SDK di crittografia AWS per Java supporto supporta il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

Per una discussione dettagliata di questi compromessi in materia di sicurezza, vedi [AWS Encryption SDK: Come decidere se la memorizzazione nella cache delle chiavi di dati è adatta alla tua applicazione](#) nel blog sulla sicurezza. AWS

## Argomenti

- [Come utilizzare il caching della chiave di dati](#)
- [Impostazione delle soglie di sicurezza della cache](#)
- [Dettagli di caching della chiave dei dati](#)
- [Esempio di caching della chiave dei dati](#)

## Come utilizzare il caching della chiave di dati

Questo argomento illustra come utilizzare il caching della chiave di dati all'interno dell'applicazione. Questo argomento consente di eseguire il processo passo per passo. Quindi, combina i passaggi in un semplice esempio che utilizza il caching della chiave dei dati in un'operazione per crittografare una stringa.

Gli esempi in questa sezione mostrano come utilizzare la [versione 2.0.](#) x e versioni successive di AWS Encryption SDK. Per esempi che utilizzano versioni precedenti, trova la tua versione nell'elenco delle [versioni](#) del GitHub repository per il tuo [linguaggio di programmazione](#).

Per esempi completi e testati sull'utilizzo del caching della chiave di dati nel AWS Encryption SDK, consulta:

- C/C++: [caching\\_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript Browser: [caching\\_cmm.ts](#)
- JavaScript Node.js: [caching\\_cmm.ts](#)
- Python: [data\\_key\\_caching\\_basic.py](#)

The [AWS Encryption SDK for .NET](#) non supporta la memorizzazione nella cache delle chiavi di dati.

## Argomenti

- [Utilizzo della memorizzazione nella cache delle chiavi dati: S tep-by-step](#)
- [Esempio di caching della chiave di dati: crittografare una stringa](#)

## Utilizzo della memorizzazione nella cache delle chiavi dati: S tep-by-step

Queste step-by-step istruzioni mostrano come creare i componenti necessari per implementare la memorizzazione nella cache delle chiavi di dati.

- [Creare una cache della chiave di dati](#). In questi esempi, utilizziamo la cache locale AWS Encryption SDK fornita da loro. Limitiamo la cache a 10 chiavi di dati.

### C

```
// Cache capacity (maximum number of entries) is required
size_t cache_capacity = 10;
struct aws_allocator *allocator = aws_default_allocator();

struct aws_cryptosdk_materials_cache *cache =
    aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);
```

### Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java depreca la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

```
// 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)
```

- Crea un [provider di chiavi principali](#) (Java e Python) o un [portachiavi](#) (C e JavaScript). JavaScript [Questi esempi utilizzano un provider di chiavi master AWS Key Management Service \(AWS KMS\) o un portachiavi compatibile AWS KMS.](#)

## C

```
// Create an AWS KMS keyring
// The input is the Amazon Resource Name (ARN)
// of an AWS KMS key

struct aws_cryptosdk_keyring *kms_keyring =
    Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);
```

## Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java depreca la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

```
// 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

Nel browser, inserisci le credenziali in modo sicuro. Questo esempio definisce le credenziali in un webpack (`kms.webpack.config`) che risolve le credenziali in fase di esecuzione. Crea un'istanza del provider client AWS KMS da un client AWS KMS e le credenziali. Quindi, quando crea il portachiavi, passa il provider del client al costruttore insieme 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])
```

- [Crea un gestore di materiali crittografici con memorizzazione nella cache \(memorizzazione nella cache CMM\)](#).

Associate la CMM per la memorizzazione nella cache alla cache e al vostro provider di chiavi principali o portachiavi. Quindi, [imposta le soglie di sicurezza della cache sulla CMM che memorizza nella cache](#).

## C

In SDK di crittografia AWS per C, è possibile creare una CMM memorizzata nella cache da una CMM sottostante, ad esempio la CMM predefinita, o da un portachiavi. Questo esempio crea il CMM di caching da un keyring.

Dopo aver creato la CMM con memorizzazione nella cache, è possibile rilasciare i riferimenti al portachiavi e alla cache. Per informazioni dettagliate, consultare [the section called “Conteggio dei riferimenti”](#).

```
// Create the caching CMM
// Set the partition ID to NULL.
// Set the required maximum age value to 60 seconds.
struct aws_cryptosdk_cmm *caching_cmm =
    aws_cryptosdk_caching_cmm_new_from_keyring(allocator, cache, kms_keyring, NULL,
        60, AWS_TIMESTAMP_SECS);

// Add an optional message threshold
// The cached data key will not be used for more than 10 messages.
aws_status = aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, 10);
```

```
// Release your references to the cache and the keyring.
aws_cryptosdk_materials_cache_release(cache);
aws_cryptosdk_keyring_release(kms_keyring);
```

## Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java non supporta la memorizzazione nella cache delle chiavi di dati, ma supporta il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa di memorizzazione nella cache dei materiali crittografici.

```
/*
 * 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  
)
```

Questo è tutto ciò che occorre fare. Quindi, lascia che il AWS Encryption SDK gestisca la cache per te o aggiungi la tua logica di gestione della cache.

Se desideri utilizzare la memorizzazione nella cache delle chiavi di dati in una chiamata per crittografare o decrittografare i dati, specifica la CMM per la memorizzazione nella cache anziché un provider di chiavi master o un'altra CMM.

### Note

Se stai crittografando flussi di dati o qualsiasi tipo di dato di dimensioni sconosciute, assicurati di specificare la dimensione dei dati nella richiesta. AWS Encryption SDK non utilizza il caching della chiave dei dati per crittografare i dati di dimensione sconosciuta.

## C

Nel SDK di crittografia AWS per C devi creare una sessione con il CMM di caching, quindi elaborare la sessione.

Per impostazione predefinita, quando le dimensioni del messaggio non sono note e vincolate, l'AWS Encryption SDK non memorizza nella cache le chiavi di dati. Per consentire il caching quando non si conoscono le dimensioni dei dati esatte, utilizza il metodo `aws_cryptosdk_session_set_message_bound` per impostare una dimensione massima per il messaggio. Imposta il limite su un valore superiore rispetto alle dimensioni stimate. Se le dimensioni effettive del messaggio superano il limite, l'operazione di crittografia non riuscirà.

```
/* Create a session with the caching CMM. Set the session mode to encrypt. */
struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
    caching_cmm);

/* Set a message bound of 1000 bytes */
aws_status = aws_cryptosdk_session_set_message_bound(session, 1000);

/* Encrypt the message using the session with the caching CMM */
aws_status = aws_cryptosdk_session_process(
    session, output_buffer, output_capacity, &output_produced,
    input_buffer, input_len, &input_consumed);

/* Release your references to the caching CMM and the session. */
aws_cryptosdk_cmm_release(caching_cmm);
```

```
aws_cryptosdk_session_destroy(session);
```

## Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java depreca la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

```
// 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

Quando si utilizza la CMM SDK di crittografia AWS per JavaScript per la memorizzazione nella cache di Node.js, il `encrypt` metodo richiede la lunghezza del testo in chiaro. Se non la fornisci, la chiave di dati non viene memorizzata nella cache. Se fornisci una lunghezza, ma i dati di testo normale forniti superano tale lunghezza, l'operazione di crittografia non riesce. Se non conosci la lunghezza esatta del testo normale, ad esempio per un flusso di dati, specifica il valore massimo previsto.

```
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
```

)

## Esempio di caching della chiave di dati: crittografare una stringa

Questo semplice esempio di codice utilizza il caching della chiave di dati per crittografare una stringa. Combina il codice della [step-by-step procedura](#) nel codice di test che è possibile eseguire.

L'esempio crea una [cache locale](#) e un [provider](#) o [portachiavi principale](#) per un AWS KMS key. [Quindi, utilizza la cache locale e il provider o portachiavi principale per creare una CMM con memorizzazione nella cache con soglie di sicurezza appropriate. In Java e Python, la richiesta di crittografia specifica la CMM di memorizzazione nella cache, i dati in chiaro da crittografare e un contesto di crittografia.](#)

In C, il CMM di caching è specificato nella sessione e la sessione viene fornita alla richiesta di crittografia.

Per eseguire questi esempi, devi fornire l'[Amazon Resource Name \(ARN\) di un AWS KMS key](#). Assicurati di avere le [autorizzazioni per utilizzare la AWS KMS key](#) e generare una chiave di dati.

Per esempi più dettagliati e reali di creazione e utilizzo di una cache di chiavi di dati, consulta. [Codice di esempio di memorizzazione nella cache delle chiavi dati](#)

C

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"). You may not use
 * this file except in compliance with the License. A copy of the License is
 * located at
 *
 *     http://aws.amazon.com/apache2.0/
 *
 * or in the "license" file accompanying this file. This file is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <aws/cryptosdk/cache.h>
#include <aws/cryptosdk/cpp/kms_keyring.h>
#include <aws/cryptosdk/session.h>
```

```

void encrypt_with_caching(
    uint8_t *ciphertext,    // output will go here (assumes ciphertext_capacity
bytes already allocated)
    size_t *ciphertext_len, // length of output will go here
    size_t ciphertext_capacity,
    const char *kms_key_arn,
    int max_entry_age,
    int cache_capacity) {
    const uint64_t MAX_ENTRY_MSGS = 100;

    struct aws_allocator *allocator = aws_default_allocator();

    // Load error strings for debugging
    aws_cryptosdk_load_error_strings();

    // Create a keyring
    struct aws_cryptosdk_keyring *kms_keyring =
Aws::Cryptosdk::KmsKeyring::Builder().Build(kms_key_arn);

    // Create a cache
    struct aws_cryptosdk_materials_cache *cache =
aws_cryptosdk_materials_cache_local_new(allocator, cache_capacity);

    // Create a caching CMM
    struct aws_cryptosdk_cmm *caching_cmm =
aws_cryptosdk_caching_cmm_new_from_keyring(
    allocator, cache, kms_keyring, NULL, max_entry_age, AWS_TIMESTAMP_SECS);
    if (!caching_cmm) abort();

    if (aws_cryptosdk_caching_cmm_set_limit_messages(caching_cmm, MAX_ENTRY_MSGS))
abort();

    // Create a session
    struct aws_cryptosdk_session *session =
    aws_cryptosdk_session_new_from_cmm_2(allocator, AWS_CRYPTOSDK_ENCRYPT,
caching_cmm);
    if (!session) abort();

    // Encryption context
    struct aws_hash_table *enc_ctx =
aws_cryptosdk_session_get_enc_ctx_ptr_mut(session);
    if (!enc_ctx) abort();
    AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_key, "purpose");
    AWS_STATIC_STRING_FROM_LITERAL(enc_ctx_value, "test");

```

```

    if (aws_hash_table_put(enc_ctx, enc_ctx_key, (void *)enc_ctx_value, NULL))
        abort();

    // Plaintext data to be encrypted
    const char *my_data = "My plaintext data";
    size_t my_data_len = strlen(my_data);
    if (aws_cryptosdk_session_set_message_size(session, my_data_len)) abort();

    // When the session uses a caching CMM, the encryption operation uses the data
    key cache
    // specified in the caching CMM.
    size_t bytes_read;
    if (aws_cryptosdk_session_process(
        session,
        ciphertext,
        ciphertext_capacity,
        ciphertext_len,
        (const uint8_t *)my_data,
        my_data_len,
        &bytes_read))
        abort();
    if (!aws_cryptosdk_session_is_done(session) || bytes_read != my_data_len)
        abort();

    aws_cryptosdk_session_destroy(session);
    aws_cryptosdk_cmm_release(caching_cmm);
    aws_cryptosdk_materials_cache_release(cache);
    aws_cryptosdk_keyring_release(kms_keyring);
}

```

## Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java depreca la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

```

// 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
```

## Impostazione delle soglie di sicurezza della cache

Quando implementi il caching della chiave di dati, è necessario configurare le soglie di sicurezza che [ilmemorizzazione nella cache CMM](#) applica.

Le soglie di sicurezza consentono di limitare per quanto tempo viene utilizzata ciascuna chiave di dati memorizzata nella cache e la quantità di dati protetta in ciascuna chiave di dati. Il CMM di memorizzazione nella cache restituisce le chiavi dei dati memorizzate nella cache solo quando la voce della cache è conforme a tutte le soglie di sicurezza. Se la voce della cache supera qualsiasi soglia, non viene utilizzata per l'operazione corrente e viene esclusa immediatamente dalla cache. Il primo utilizzo di ciascuna chiave di dati (prima del caching) è esente da queste soglie.

In generale, è possibile usare la quantità minima di caching obbligatoria per soddisfare gli obiettivi in termini di costi e prestazioni.

AWS Encryption SDK memorizza solo le chiavi dei dati che vengono crittografate utilizzando una [funzione di derivazione della chiave](#). Inoltre, stabilisce i limiti massimi per alcuni valori di soglia. Queste restrizioni assicurano che le chiavi dei dati non vengano riutilizzate oltre i limiti crittografici. Tuttavia, poiché le chiavi di dati di testo non crittografato vengono memorizzate nella cache (in memoria, per impostazione predefinita), prova a ridurre al minimo il tempo durante il quale le chiavi vengono salvate. Inoltre, prova a limitare i dati che potrebbero essere esposti se una chiave è compromessa.

Per esempi di impostazione delle soglie di sicurezza della cache, vedere [AWS Encryption SDK: Come decidere se il caching della chiave di dati è adatto alla tua applicazione](#) nella AWS Blog di sicurezza.

### Note

Il CMM del caching applica tutte le seguenti soglie. Se non si specifica un valore opzionale, il CMM del caching usa il valore predefinito.

Per disabilitare temporaneamente il caching della chiave di dati, le implementazioni Java e Python di AWS Encryption SDK fornire uncache null dei materiali crittografici(cache null). La

cache null restituisce un vuoto per ogni richiesta GET e non risponde alle richieste PUT. È consigliabile utilizzare la cache null invece di impostare la [capacità della cache](#) o le soglie di sicurezza su 0. Per ulteriori informazioni, consulta gli argomenti relativi alla cache null in [Java](#) e [Python](#).

### Età massima (obbligatoria)

Stabilisce per quanto tempo può essere utilizzata una voce di cache, dal momento in cui è stata aggiunta. Questo valore è obbligatorio. Immetti un valore superiore a 0. L'AWS Encryption SDK non limita il valore massimo per l'età.

Tutte le implementazioni linguistiche AWS Encryption SDK definire l'età massima in secondi, tranne che per SDK di crittografia AWS per JavaScript, che utilizza millisecondi.

Utilizza l'intervallo più breve che consente ancora alla tua applicazione di trarre vantaggio dalla cache. È possibile usare la soglia dell'età massima come una policy di rotazione delle chiavi. Utilizzala per limitare il riutilizzo delle chiavi dei dati, ridurre al minimo l'esposizione di materiali crittografici ed eliminare le chiavi di dati le cui policy potrebbero essere cambiate mentre sono state memorizzate nella cache.

### Il numero massimo di messaggi crittografati (opzionale)

Stabilisce il numero massimo di messaggi che una chiave di dati nella cache è in grado di crittografare. Questo valore è facoltativo. Inserisci un valore tra 1 e  $2^{32}$  messaggi. Il valore predefinito è  $2^{32}$  messaggi.

Imposta il numero di messaggi protetti da ciascuna chiave memorizzata nella cache in modo che sia sufficiente per ottenere valore dal riutilizzo, ma abbastanza contenuto per limitare il numero di messaggi che potrebbero essere esposti se una chiave è compromessa.

### Il numero massimo di byte crittografati (opzionale)

Stabilisce il numero massimo di byte che una chiave di dati nella cache è in grado di crittografare. Questo valore è facoltativo. Inserisci un valore tra 0 e  $2^{63} - 1$ . Il valore predefinito è  $2^{63} - 1$ . Il valore 0 consente di utilizzare il caching della chiave di dati solo quando stai crittografando stringhe di messaggio vuote.

I byte nella richiesta corrente sono inclusi durante la valutazione di questa soglia. Se i byte elaborati, più i byte attuali, superano la soglia, la chiave di dati memorizzata viene rimossa dalla cache, anche se è possibile che sia stata utilizzata per una richiesta di dimensioni ridotte.

## Dettagli di caching della chiave dei dati

La maggior parte delle applicazioni possono utilizzare l'impostazione predefinita del caching della chiavi dei dati senza la necessità di scrivere codice personalizzato. Questa sezione descrive l'implementazione predefinita e alcuni dettagli sulle opzioni.

### Argomenti

- [In che modo funziona il caching della chiave dei dati](#)
- [Creazione di una cache di materiali crittografici](#)
- [Creazione di un responsabile della cache di materiali crittografici](#)
- [Cosa c'è in una voce della cache della chiave di dati?](#)
- [Contesto di crittografia: Come selezionare le voci della cache](#)
- [La mia applicazione utilizza chiavi dati memorizzate nella cache?](#)

## In che modo funziona il caching della chiave dei dati

Quando utilizzi il caching della chiave dei dati in una richiesta di crittografia o decrittografia dei dati, AWS Encryption SDK prima cerca la cache per una chiave di dati in grado di soddisfare la richiesta. Se trova una corrispondenza valida, utilizza la chiave di dati memorizzati per crittografare i dati. In caso contrario, genera una nuova chiave di dati, come farebbe in assenza della cache.

Il caching della chiave di dati non viene utilizzato per i dati di dimensione sconosciuta, come, ad esempio i dati in streaming. Ciò consente al CMM di caching di applicare correttamente il [soglia massima di byte](#). Per evitare questo comportamento, aggiungi le dimensioni del messaggio alla richiesta di crittografia.

Oltre a una cache, il caching della chiave di dati utilizza un [Gestore di materiali crittografici](#) (memorizzazione nella cache di CMM). Il caching CMM è specializzato [Responsabile di materiali crittografici \(CMM\)](#) che interagisce con un [cache](#) e un sottostante [CMM](#). Quando specifichi un [Provider di chiavi master](#) portachiavi, il [AWS Encryption SDK](#) crea un CMM predefinito per te.) CMM di caching memorizza nella cache le chiavi di dati restituite dal CMM sottostante. CMM di caching applica anche le soglie di sicurezza della cache che hai impostato.

Per evitare che venga selezionata la chiave di dati errata dalla cache, tutti i CMM di caching compatibili richiedono che le proprietà seguenti dei materiali crittografici memorizzati nella cache corrispondano alla richiesta di materiali.

- [Suite di algoritmi](#)
- [Contesto di crittografia](#) (anche se vuoto)
- Nome della partizione (stringa che identifica il CMM di caching)
- (Solo decrittazione) Chiavi di dati crittografate

#### Note

AWS Encryption SDK memorizza nella cache le chiavi dei dati solo quando la [suite di algoritmi](#) utilizza una [funzione di derivazione della chiave](#).

I seguenti flussi di lavoro mostrano il modo in cui una richiesta di crittografia dei dati viene elaborata con e senza il caching della chiave dei dati. Questi mostrano il modo in cui i componenti di caching che crei, tra cui la cache e il CMM di caching, vengono utilizzati nel processo.

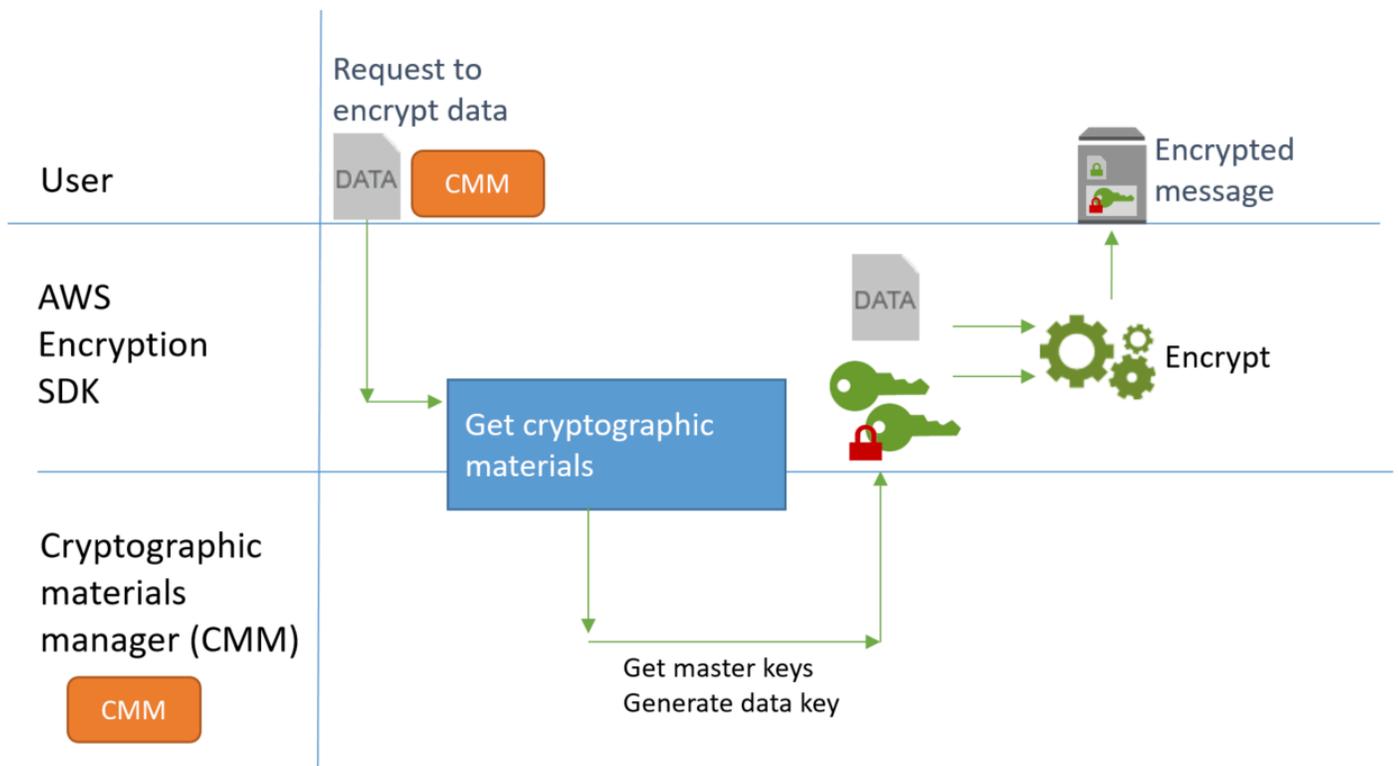
## Crittografia dei dati senza caching

Per ottenere i materiali di crittografia senza caching:

1. Un'applicazione chiede a AWS Encryption SDK di crittografare i dati.

La richiesta specifica un fornitore di chiavi master o keyring. LaAWS Encryption SDKcrea un CMM predefinito che interagisce con il provider di chiavi master o keyring.

2. LaAWS Encryption SDKchiede al CMM i materiali di crittografia (ottiene materiali crittografici).
3. Il CMM chiede[Keyring](#)(C eJavaScript) o[Provider di chiavi master](#)(Java e Python) per materiali crittografici. Ciò potrebbe comportare una chiamata a un servizio crittografico, come AWS Key Management Service (AWS KMS). CMM restituisce i materiali di crittografia all'AWS Encryption SDK.
4. L'AWS Encryption SDK utilizza la chiave di dati in testo normale per crittografare i dati. Archivia i dati crittografati e le chiavi di dati crittografate in un [messaggio crittografato](#) che viene restituito all'utente.



## Crittografia dei dati con il caching

Per ottenere i materiali di crittografia con il caching della chiave di dati:

1. Un'applicazione chiede a AWS Encryption SDK di crittografare i dati.

La richiesta specifica un [Caching di materiali crittografici \(caching CMM\)](#) associato a un gestore di materiali crittografici sottostante (CMM). Quando specifichi un provider di chiavi master o keyring, AWS Encryption SDK crea un CMM predefinito per te.

2. L'SDK richiede i materiali di crittografia al CMM di caching specificato.

3. CMM di caching richiede i materiali di crittografia dalla cache.

a. Se la cache trova una corrispondenza, aggiorna l'età, utilizza i valori della voce della cache corrispondente e restituisce i materiali di crittografia al CMM di caching.

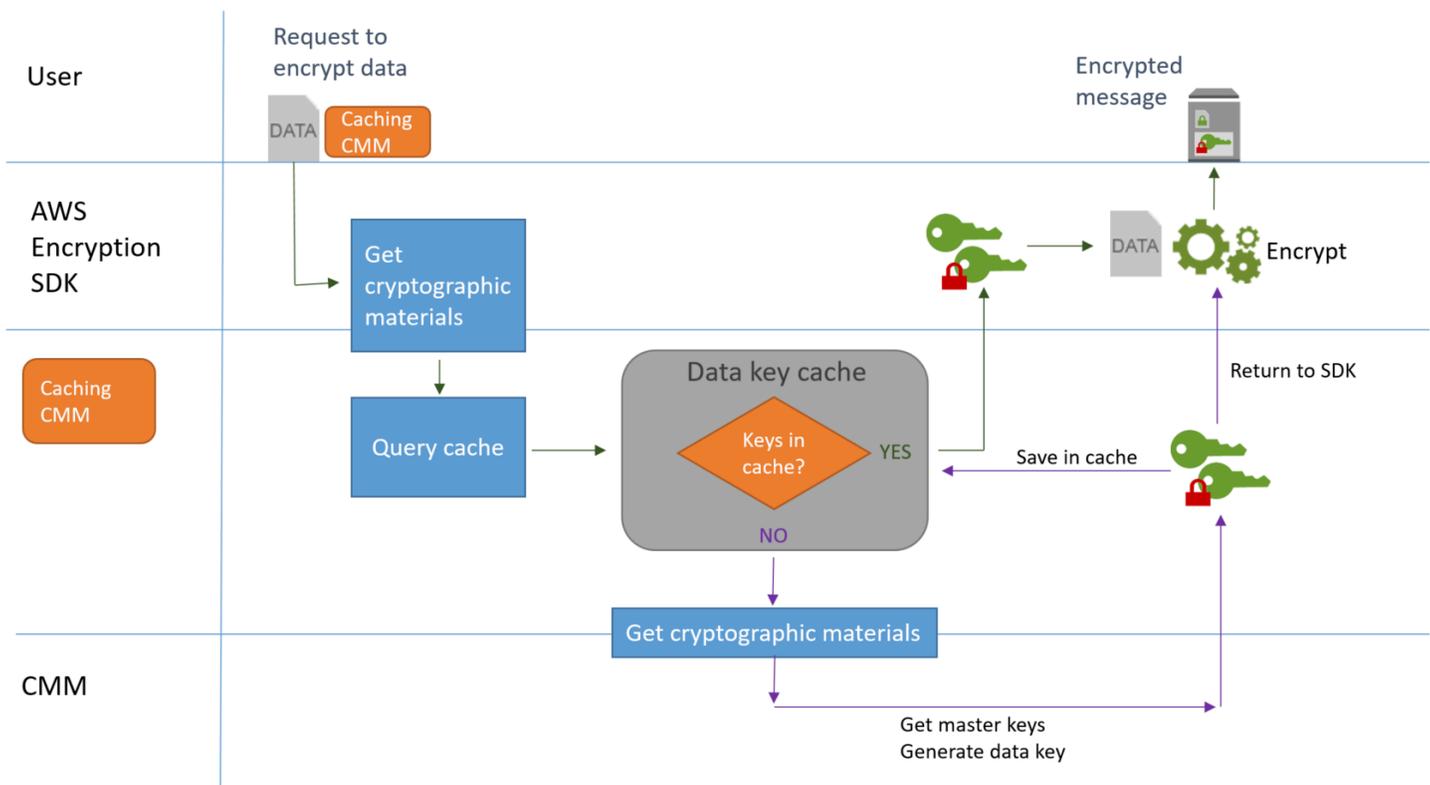
Se la voce cache è conforme alla sua [Soglie di sicurezza](#), il CMM di caching lo restituisce all'SDK. In caso contrario, comunica alla cache di eliminare la voce e di procedere come se non ci fossero corrispondenze.

b. Se la cache non è in grado di trovare una corrispondenza valida, il CMM di caching chiede al CMM sottostante di generare una nuova chiave di dati.

Il CMM sottostante ottiene i materiali crittografici dal proprio keyring (C e JavaScript) o fornitore di chiavi master (Java e Python). Questo potrebbe comportare una chiamata a un servizio, come AWS Key Management Service. Il CMM sottostante restituisce copie di testo crittografato e non crittografato della chiave di dati al CMM di caching.

CMM di caching salva i nuovi materiali di crittografia nella cache.

4. CMM di caching restituisce i materiali di crittografia all'AWS Encryption SDK.
5. L'AWS Encryption SDK utilizza la chiave di dati in testo normale per crittografare i dati. Archivia i dati crittografati e le chiavi di dati crittografate in un [messaggio crittografato](#) che viene restituito all'utente.



## Creazione di una cache di materiali crittografici

AWS Encryption SDK definisce i requisiti per una cache di materiali di crittografia utilizzata nel caching della chiave dei dati. Fornisce anche una cache locale, che è configurabile, in memoria, [Cache utilizzata meno di recente \(LRU\)](#). Per creare un'istanza della cache locale, utilizzare il `LocalCryptoMaterialsCache` costruttore

di Java e Python, il `getLocalCryptographicMaterialsCache` funzione in JavaScript, o `ilaws_cryptosdk_materials_cache_local_new` Costruttore di C.

La cache locale include la logica per la gestione della cache di base, incluso l'aggiunta, la cancellazione e la corrispondenza di voci memorizzate e mantenute nella cache. Non è necessario scrivere una logica di gestione della cache personalizzata. È possibile utilizzare la cache locale così com'è, personalizzarla o sostituirla con qualsiasi cache compatibile.

Quando crei una cache locale, la imposti capacità. Cioè, il numero massimo di voci che la cache può contenere. Questa impostazione consente di progettare una cache efficiente con un riutilizzo della chiave dei dati limitato.

La SDK di crittografia AWS per Java e la SDK di crittografia AWS per Python fornire anche una cache di materiali crittografici (`NullCryptoMaterialsCache`). La `NullCryptoMaterialsCache` restituisce un vuoto per tutte le operazioni GET e non risponde alle operazioni PUT. Puoi utilizzare la `NullCryptoMaterialsCache` per i test o per disabilitare temporaneamente il caching in un'applicazione che include il codice di memorizzazione nella cache.

Nella AWS Encryption SDK ogni cache di materiali crittografici è associata a un [Gestore di materiali crittografici](#) (memorizzazione nella cache di CMM). CMM di caching ottiene le chiavi dei dati dalla cache, mette le chiavi dei dati nella cache e applica [Soglie di sicurezza](#) che hai impostato. Quando viene creato un CMM di caching, è necessario specificare la cache utilizzata e il CMM sottostante o il provider della chiave master che genera le chiavi di dati che memorizza.

## Creazione di un responsabile della cache di materiali crittografici

Per abilitare il caching della chiave di dati, crea un [cache](#) e un [Gestore di materiali crittografici](#) (memorizzazione nella cache di CMM). Quindi, nelle tue richieste per crittografare o decrittografare i dati, è necessario specificare un CMM di caching, invece di uno standard [Responsabile di materiali crittografici \(CMM\)](#), oppure [Provider di chiavi master](#) [Keyring](#).

Esistono due tipi di CMM. Entrambi ottengono le chiavi dei dati (e relativo materiale crittografico), ma in modi diversi, come segue:

- Un CMM è associato a un keyring (C o C o JavaScript) o un provider di chiavi master (Java e Python). Quando l'SDK richiede i materiali di crittografia o decrittazione al CMM, il CMM ottiene i materiali dal proprio keyring o fornitore di chiavi master. In Java e Python, il CMM utilizza le chiavi master per generare, crittografare o decrittare le chiavi di dati. In C e JavaScript, il keyring genera, crittografa e restituisce i materiali crittografici.

- Un CMM di caching è associato a una cache, ad esempio a [cache locale](#) e un CMM sottostante. Quando l'SDK richiede i materiali crittografici al CMM, il caching CMM prova a ottenerli dalla cache. Se non è in grado di trovare una corrispondenza, il CMM di caching richiede i materiali al CMM sottostante. Quindi, memorizza nella cache i nuovi materiali crittografici prima di restituirli all'intermediario.

Anche il CMM di memorizzazione nella cache [Soglie di sicurezza](#) che hai impostato per ogni voce della cache. Poiché le soglie di sicurezza sono impostate e applicate dal CMM nella cache, è possibile usare qualsiasi cache compatibile, anche se la cache non è progettata per materiali sensibili.

## Cosa c'è in una voce della cache della chiave di dati?

Il caching della chiave di dati memorizza le chiavi di dati e i relativi materiali crittografici in una cache. Ogni voce include gli elementi elencati di seguito. Queste informazioni potrebbero esserti utili quando decidi di utilizzare la funzionalità di caching della chiave dei dati e quando imposti le soglie di sicurezza su cache di materiali crittografici (caching CMM).

### Voci nella cache per le richieste di crittografia

Le voci che vengono aggiunte a una cache della chiave di dati in seguito a un'operazione di crittografia includono i seguenti elementi:

- Chiave di dati di testo non crittografato
- Chiavi di dati crittografati (una o più)
- [Contesto di crittografia](#)
- Chiave di firma del messaggio (se ne viene utilizzata una)
- [Suite di algoritmi](#)
- Metadati, inclusi i contatori di utilizzo per applicare le soglie di sicurezza

### Voci nella cache per le richieste di decrittografia

Le voci che vengono aggiunte a una cache della chiave di dati in seguito a un'operazione di decrittografia includono i seguenti elementi:

- Chiave di dati di testo non crittografato
- Chiave di verifica della firma (se ne viene utilizzata una)

- Metadati, inclusi i contatori di utilizzo per applicare le soglie di sicurezza

## Contesto di crittografia: Come selezionare le voci della cache

È possibile specificare un contesto di crittografia in qualsiasi richiesta per crittografare i dati. Tuttavia, il contesto di crittografia svolge un ruolo speciale nel contesto del caching della chiave dei dati. Ti consente di creare sottogruppi di chiavi di dati nella cache, anche quando le chiavi di dati provengono dalla stessa cache.

Un [contesto di crittografia](#) è un set di coppie chiave-valore che contiene dati arbitrari non segreti. Durante la crittografia, il contesto di crittografia è legato ai dati crittografati, in modo che lo stesso contesto di crittografia è necessario per decrittografare i dati. In AWS Encryption SDK, il contesto di crittografia viene memorizzato nel [messaggio crittografato](#) insieme ai dati crittografati e alle chiavi di dati.

Quando utilizzi una cache di chiavi di dati, è anche possibile utilizzare il contesto di crittografia per selezionare chiavi di dati specifiche nella cache per le tue operazioni di crittografia. Il contesto di crittografia viene salvato nella voce della cache con la chiave dei dati (fa parte dell'ID della voce della cache). Le chiavi dei dati nella cache vengono riutilizzate solo quando i contesti di crittografia corrispondono. Se desideri riutilizzare alcune chiavi dei dati per una richiesta di crittografia, specificare lo stesso contesto di crittografia. Se desideri evitare tali chiavi di dati, specifica un contesto di crittografia diverso.

Il contesto di crittografia è sempre facoltativo, ma consigliato. Se non specifichi un contesto di crittografia nella tua richiesta, un contesto di crittografia vuoto è incluso nell'identificatore della voce di cache e corrisponde a ogni richiesta.

## La mia applicazione utilizza chiavi dati memorizzate nella cache?

La memorizzazione nella cache delle chiavi dati è una strategia di ottimizzazione molto efficace per determinate applicazioni e carichi di lavoro. Tuttavia, poiché comporta un certo rischio, è importante determinare quanto sia efficace per la propria situazione e quindi decidere se i benefici superano i rischi.

Poiché la memorizzazione nella cache delle chiavi dati riutilizza le chiavi dati, l'effetto più evidente è ridurre il numero di chiamate per generare nuove chiavi dati. Quando viene implementata la memorizzazione nella cache delle chiavi dati, AWS Encryption SDK richiama l'operazione `GenerateDataKey` AWS KMS solo per creare la chiave dati iniziale e quando la cache non viene

eseguita. Tuttavia, la memorizzazione nella cache migliora sensibilmente le prestazioni solo nelle applicazioni che generano numerose chiavi dati con le stesse caratteristiche, tra cui lo stesso contesto di crittografia e suite di algoritmi.

Per determinare se l'implementazione di AWS Encryption SDK utilizza effettivamente le chiavi dati dalla cache, provare le seguenti tecniche.

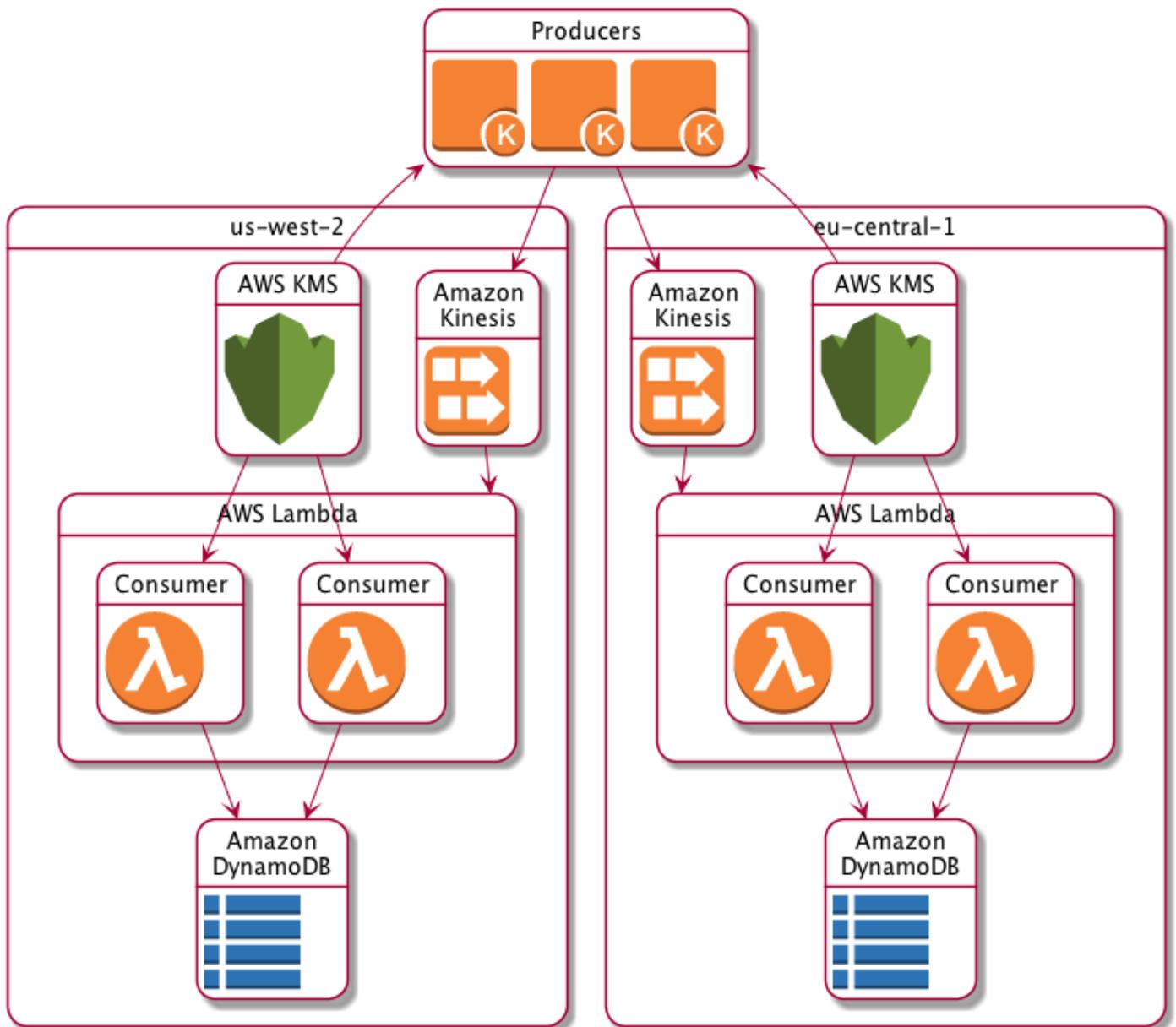
- Nei registri dell'infrastruttura della chiave master, controllare la frequenza delle chiamate per creare nuove chiavi dati. Quando la memorizzazione nella cache delle chiavi dati è efficace, il numero di chiamate per creare nuove chiavi dovrebbe diminuire sensibilmente. Ad esempio, se si sta utilizzando un AWS KMS master key provider o portachiavi, cerca nel CloudTrail Registri per [GenerateDataChiave](#) chiama.
- Confrontare i [messaggi crittografati](#) restituiti da AWS Encryption SDK in risposta a diverse richieste di crittografia. Ad esempio, se si sta utilizzando il SDK di crittografia AWS per Java, confrontare il [ParsedCiphertext](#) oggetto da diverse chiamate crittografate. Nella SDK di crittografia AWS per JavaScript confrontare i contenuti della `encryptedDataKeys` proprietà del [MessageHeader](#). Quando le chiavi dati vengono riutilizzate, le chiavi dati crittografate nel messaggio crittografato sono identiche.

## Esempio di caching della chiave dei dati

Questo esempio utilizza [Caching della chiave dei dati](#) con un [cache locale](#) per velocizzare un'applicazione in cui i dati generati da più dispositivi siano crittografati e memorizzati in regioni diverse.

In questo scenario, più produttori di dati generano dati, li crittografano e scrivono su una [Kinesis Streams](#) in ciascuna regione. [AWS Lambda](#) Le funzioni\* (consumatori) decrittografano i flussi e scrivono dati di testo semplice su una tabella DynamoDB nella regione. I produttori di dati e i consumatori utilizzano AWS Encryption SDK e un [AWS KMS Provider di chiavi master](#). Per ridurre le chiamate a KMS, ogni produttore e consumatore ha la propria cache locale.

Il codice sorgente per questi esempi si trova in [Java e Python](#). Il campione include anche un modello AWS CloudFormation che definisce le risorse per gli esempi.



## Risultati della cache locale

La tabella riportata di seguito mostra che una cache locale riduce le chiamate totali a KMS (al secondo per regione) in questo esempio all'1% del valore originale.

Richieste del produttore

Richieste al secondo per client	Client per regione	Richieste medie al
---------------------------------	--------------------	--------------------

	Generare la chiave di dati (us-west-2)	Crittografare la chiave di dati (eu-central-1)	Totale (per regione)		secondo per regione
Nessuna cache	1	1	1	500	500
Cache locale	1 rps/100 usi	1 rps/100 usi	1 rps/100 usi	500	5

### Richieste dei consumatori

	Richieste al secondo per client			Client per regione	Richieste medie al secondo per regione
	Decrittografare la chiave dei dati	Producer	Total		
Nessuna cache	1 rps per produttore	500	500	2	1.000
Cache locale	1 rps per produttore/100 usi	500	5	2	10

## Codice di esempio di memorizzazione nella cache delle chiavi dati

Questo esempio di codice crea una semplice implementazione della memorizzazione nella cache delle chiavi di dati con una [cache locale](#) in Java e Python. Il codice crea due istanze di una cache locale: una per [i produttori di dati](#) che crittografano i dati e l'altra per [i consumatori di dati \(AWS Lambdafunzioni\) che decifrano i dati](#). Per i dettagli sull'implementazione della memorizzazione nella cache delle chiavi di dati in ogni lingua, consulta la documentazione di [Javadoc](#) e [Python](#) per AWS Encryption SDK

La memorizzazione nella cache delle chiavi dati è disponibile per tutti i [linguaggi di programmazione](#) supportati da AWS Encryption SDK

Per esempi completi e testati sull'utilizzo del caching della chiave di dati nel AWS Encryption SDK, consulta:

- C/C++: [caching\\_cmm.cpp](#)
- Java: [SimpleDataKeyCachingExample.java](#)
- JavaScript Browser: [caching\\_cmm.ts](#)
- JavaScript Node.js: [caching\\_cmm.ts](#)
- Python: [data\\_key\\_caching\\_basic.py](#)

## Producer

[Il produttore ottiene una mappa, la converte in JSON, usa la per crittografarla AWS Encryption SDK e invia il record di testo cifrato a un flusso Kinesis in ciascuna di esse.](#) Regione AWS

[Il codice definisce un gestore di materiali crittografici per la memorizzazione nella cache \(caching CMM\) e lo associa a una cache locale e a un provider di chiavi master sottostante.](#) [AWS KMS](#) La CMM con memorizzazione nella cache memorizza nella cache le chiavi di dati (e i [relativi materiali crittografici](#)) del fornitore della chiave principale. Interagisce inoltre con la cache per conto di SDK e applica le soglie di sicurezza impostate.

Poiché la chiamata al metodo di crittografia specifica una CMM memorizzata nella cache, anziché un normale [gestore di materiali crittografici \(CMM\) o un fornitore di chiavi master](#), la crittografia utilizzerà la memorizzazione nella cache delle chiavi di dati.

## Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java depreca la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

```
/*
 * 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
        )
```

## Consumer

Il data consumer è una [AWS Lambda](#) funzione attivata dagli eventi [Kinesis](#). Decodifica e deserializza ogni record e scrive il record in testo semplice in una tabella Amazon [DynamoDB](#) nella stessa regione.

Come il codice del produttore, il codice consumer consente la memorizzazione nella cache delle chiavi di dati utilizzando un gestore di materiali crittografici (caching CMM) nelle chiamate al metodo di decrittografia.

Il codice Java crea un provider di chiavi master in modalità rigorosa con un valore specificato. AWS KMS key [La modalità rigorosa non è richiesta per la decrittografia, ma è una buona pratica](#). Il codice Python utilizza la modalità di rilevamento, che consente di AWS Encryption SDK utilizzare qualsiasi chiave di wrapping che ha crittografato una chiave di dati per decrittografarla.

### Java

L'esempio seguente utilizza la versione 2. x di SDK di crittografia AWS per Java. Versione 3. x of the SDK di crittografia AWS per Java deprecia la chiave dati che memorizza nella cache CMM. Con la versione 3. x, puoi anche usare il [portachiavi AWS KMS Hierarchical](#), una soluzione alternativa per la memorizzazione nella cache dei materiali crittografici.

Questo codice crea un provider di chiavi principali per la decrittografia in modalità rigorosa. AWS Encryption SDK possono utilizzare solo il codice specificato dall'AWS KMS key utente per decrittografare il messaggio.

```
/*
 * 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

Questo codice Python viene decrittografato con un provider di chiavi master in modalità di scoperta. Consente di AWS Encryption SDK utilizzare qualsiasi chiave di wrapping che ha crittografato una chiave di dati per decrittografarla. [La modalità rigorosa, in cui si specificano le chiavi di wrapping che possono essere utilizzate per la decrittografia, è una procedura consigliata.](#)

```
"""
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)

```

## Esempio di caching della chiave dei dati:AWS CloudFormationmodello

QuestoAWS CloudFormationmodello imposta tutto il necessarioAWS risorse per riprodurre [i esempi di caching della chiave dei dati](#).

### JSON

```

{
  "Parameters": {
    "SourceCodeBucket": {
      "Type": "String",
      "Description": "S3 bucket containing Lambda source code zip files"
    },
    "PythonLambdaS3Key": {
      "Type": "String",
      "Description": "S3 key containing Python Lambda source code zip file"
    },
    "PythonLambdaObjectVersionId": {
      "Type": "String",
      "Description": "S3 version id for S3 key containing Python Lambda source code zip file"
    },
    "JavaLambdaS3Key": {
      "Type": "String",
      "Description": "S3 key containing Python Lambda source code zip file"
    },
    "JavaLambdaObjectVersionId": {
      "Type": "String",
      "Description": "S3 version id for S3 key containing Python Lambda source code zip file"
    },
    "KeyAliasSuffix": {
      "Type": "String",

```

```
    "Description": "Suffix to use for KMS key Alias (ie: alias/
<KeyAliasSuffix>)"
  },
  "StreamName": {
    "Type": "String",
    "Description": "Name to use for Kinesis Stream"
  }
},
"Resources": {
  "InputStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
      "Name": {
        "Ref": "StreamName"
      },
      "ShardCount": 2
    }
  },
  "PythonLambdaOutputTable": {
    "Type": "AWS::DynamoDB::Table",
    "Properties": {
      "AttributeDefinitions": [
        {
          "AttributeName": "id",
          "AttributeType": "S"
        }
      ],
      "KeySchema": [
        {
          "AttributeName": "id",
          "KeyType": "HASH"
        }
      ],
      "ProvisionedThroughput": {
        "ReadCapacityUnits": 1,
        "WriteCapacityUnits": 1
      }
    }
  },
  "PythonLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
```

```

    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
      }
    ],
    "ManagedPolicyArns": [
      "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
    ],
    "Policies": [
      {
        "PolicyName": "PythonLambdaAccess",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchWriteItem"
              ],
              "Resource": {
                "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}"
              }
            },
            {
              "Effect": "Allow",
              "Action": [
                "dynamodb:PutItem"
              ],
              "Resource": {
                "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*"
              }
            },
            {
              "Effect": "Allow",
              "Action": [

```

```

        "kinesis:GetRecords",
        "kinesis:GetShardIterator",
        "kinesis:DescribeStream",
        "kinesis:ListStreams"
    ],
    "Resource": {
        "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
    }
}
]
}
]
}
},
"PythonLambdaFunction": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
        "Description": "Python consumer",
        "Runtime": "python2.7",
        "MemorySize": 512,
        "Timeout": 90,
        "Role": {
            "Fn::GetAtt": [
                "PythonLambdaRole",
                "Arn"
            ]
        },
        "Handler":
"aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler",
        "Code": {
            "S3Bucket": {
                "Ref": "SourceCodeBucket"
            },
            "S3Key": {
                "Ref": "PythonLambdaS3Key"
            },
            "S3ObjectVersion": {
                "Ref": "PythonLambdaObjectVersionId"
            }
        },
        "Environment": {
            "Variables": {

```

```

        "TABLE_NAME": {
            "Ref": "PythonLambdaOutputTable"
        }
    }
},
"PythonLambdaSourceMapping": {
    "Type": "AWS::Lambda::EventSourceMapping",
    "Properties": {
        "BatchSize": 1,
        "Enabled": true,
        "EventSourceArn": {
            "Fn::Sub": "arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}"
        },
        "FunctionName": {
            "Ref": "PythonLambdaFunction"
        },
        "StartingPosition": "TRIM_HORIZON"
    }
},
"JavaLambdaOutputTable": {
    "Type": "AWS::DynamoDB::Table",
    "Properties": {
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            }
        ],
        "KeySchema": [
            {
                "AttributeName": "id",
                "KeyType": "HASH"
            }
        ],
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 1,
            "WriteCapacityUnits": 1
        }
    }
},
"JavaLambdaRole": {

```

```

    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "ManagedPolicyArns": [
        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
      ],
      "Policies": [
        {
          "PolicyName": "JavaLambdaAccess",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "dynamodb:DescribeTable",
                  "dynamodb:BatchWriteItem"
                ],
                "Resource": {
                  "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}"
                }
              },
              {
                "Effect": "Allow",
                "Action": [
                  "dynamodb:PutItem"
                ],
                "Resource": {
                  "Fn::Sub": "arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}*"
                }
              }
            ]
          }
        }
      ]
    }
  }

```



```

    }
  },
  "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": {

```



```

    "RegionKinesisCMKAlias": {
      "Type": "AWS::KMS::Alias",
      "Properties": {
        "AliasName": {
          "Fn::Sub": "alias/${KeyAliasSuffix}"
        },
        "TargetKeyId": {
          "Ref": "RegionKinesisCMK"
        }
      }
    }
  }
}

```

## YAML

```

Parameters:
  SourceCodeBucket:
    Type: String
    Description: S3 bucket containing Lambda source code zip files
  PythonLambdaS3Key:
    Type: String
    Description: S3 key containing Python Lambda source code zip file
  PythonLambdaObjectVersionId:
    Type: String
    Description: S3 version id for S3 key containing Python Lambda source code
zip file
  JavaLambdaS3Key:
    Type: String
    Description: S3 key containing Python Lambda source code zip file
  JavaLambdaObjectVersionId:
    Type: String
    Description: S3 version id for S3 key containing Python Lambda source code
zip file
  KeyAliasSuffix:
    Type: String
    Description: 'Suffix to use for KMS CMK Alias (ie: alias/<KeyAliasSuffix>)'
  StreamName:
    Type: String
    Description: Name to use for Kinesis Stream
Resources:
  InputStream:
    Type: AWS::Kinesis::Stream

```

```

    Properties:
      Name: !Ref StreamName
      ShardCount: 2
    PythonLambdaOutputTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          -
            AttributeName: id
            AttributeType: S
        KeySchema:
          -
            AttributeName: id
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
    PythonLambdaRole:
      Type: AWS::IAM::Role
      Properties:
        AssumeRolePolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Principal:
                Service: lambda.amazonaws.com
              Action: sts:AssumeRole
        ManagedPolicyArns:
          - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
        Policies:
          -
            PolicyName: PythonLambdaAccess
            PolicyDocument:
              Version: 2012-10-17
              Statement:
                -
                  Effect: Allow
                  Action:
                    - dynamodb:DescribeTable
                    - dynamodb:BatchWriteItem
                  Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}

```

```

        Effect: Allow
        Action:
          - dynamodb:PutItem
        Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${PythonLambdaOutputTable}*
      -
        Effect: Allow
        Action:
          - kinesis:GetRecords
          - kinesis:GetShardIterator
          - kinesis:DescribeStream
          - kinesis:ListStreams
        Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    PythonLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Description: Python consumer
        Runtime: python2.7
        MemorySize: 512
        Timeout: 90
        Role: !GetAtt PythonLambdaRole.Arn
        Handler:
aws_crypto_examples.kinesis_datakey_caching.consumer.lambda_handler
        Code:
          S3Bucket: !Ref SourceCodeBucket
          S3Key: !Ref PythonLambdaS3Key
          S3ObjectVersion: !Ref PythonLambdaObjectVersionId
        Environment:
          Variables:
            TABLE_NAME: !Ref PythonLambdaOutputTable
    PythonLambdaSourceMapping:
      Type: AWS::Lambda::EventSourceMapping
      Properties:
        BatchSize: 1
        Enabled: true
        EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
        FunctionName: !Ref PythonLambdaFunction
        StartingPosition: TRIM_HORIZON
    JavaLambdaOutputTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:

```

```

    -
      AttributeName: id
      AttributeType: S
    KeySchema:
      -
        AttributeName: id
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
  JavaLambdaRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      Policies:
        -
          PolicyName: JavaLambdaAccess
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - dynamodb:DescribeTable
                  - dynamodb:BatchWriteItem
                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}
              -
                Effect: Allow
                Action:
                  - dynamodb:PutItem
                Resource: !Sub arn:aws:dynamodb:${AWS::Region}:
${AWS::AccountId}:table/${JavaLambdaOutputTable}*
              -
                Effect: Allow

```

```

        Action:
            - kinesis:GetRecords
            - kinesis:GetShardIterator
            - kinesis:DescribeStream
            - kinesis:ListStreams
        Resource: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
    JavaLambdaFunction:
        Type: AWS::Lambda::Function
        Properties:
            Description: Java consumer
            Runtime: java8
            MemorySize: 512
            Timeout: 90
            Role: !GetAtt JavaLambdaRole.Arn
            Handler:
com.amazonaws.crypto.examples.kinesisdatakeycaching.LambdaDecryptAndWrite::handleRequest
            Code:
                S3Bucket: !Ref SourceCodeBucket
                S3Key: !Ref JavaLambdaS3Key
                S3ObjectVersion: !Ref JavaLambdaObjectVersionId
            Environment:
                Variables:
                    TABLE_NAME: !Ref JavaLambdaOutputTable
                    CMK_ARN: !GetAtt RegionKinesisCMK.Arn
    JavaLambdaSourceMapping:
        Type: AWS::Lambda::EventSourceMapping
        Properties:
            BatchSize: 1
            Enabled: true
            EventSourceArn: !Sub arn:aws:kinesis:${AWS::Region}:
${AWS::AccountId}:stream/${InputStream}
            FunctionName: !Ref JavaLambdaFunction
            StartingPosition: TRIM_HORIZON
    RegionKinesisCMK:
        Type: AWS::KMS::Key
        Properties:
            Description: Used to encrypt data passing through Kinesis Stream in this
region
            Enabled: true
            KeyPolicy:
                Version: 2012-10-17
                Statement:
                    -

```

```

    Effect: Allow
    Principal:
      AWS: !Sub arn:aws:iam::${AWS::AccountId}:root
    Action:
      # Data plane actions
      - kms:Encrypt
      - kms:GenerateDataKey
      # Control plane actions
      - kms:CreateAlias
      - kms>DeleteAlias
      - kms:DescribeKey
      - kms:DisableKey
      - kms:EnableKey
      - kms:PutKeyPolicy
      - kms:ScheduleKeyDeletion
      - kms:UpdateAlias
      - kms:UpdateKeyDescription
    Resource: '*'
  -
    Effect: Allow
    Principal:
      AWS:
        - !GetAtt PythonLambdaRole.Arn
        - !GetAtt JavaLambdaRole.Arn
    Action: kms:Decrypt
    Resource: '*'
RegionKinesisCMKAlias:
  Type: AWS::KMS::Alias
  Properties:
    AliasName: !Sub alias/${KeyAliasSuffix}
    TargetKeyId: !Ref RegionKinesisCMK

```

# Versioni del AWS Encryption SDK

Le implementazioni del AWS Encryption SDK linguaggio utilizzano il controllo delle [versioni semantico](#) per semplificare l'identificazione dell'entità delle modifiche in ogni versione. Una modifica del numero di versione principale, ad esempio 1. x. da x a 2. x. x, indica una modifica sostanziale che probabilmente richiederà modifiche al codice e una distribuzione pianificata. Le modifiche introdotte in una nuova versione potrebbero non influire su tutti i casi d'uso, consulta le note di rilascio per vedere se ne risentono. Una modifica in una versione secondaria, ad esempio x .1. da x a x .2. x, è sempre retrocompatibile, ma potrebbe includere elementi obsoleti.

Quando possibile, usa la versione più recente di AWS Encryption SDK nel linguaggio di programmazione prescelto. La [politica di manutenzione e supporto](#) per ciascuna versione differisce tra le implementazioni del linguaggio di programmazione. [Per informazioni dettagliate sulle versioni supportate nel linguaggio di programmazione preferito, consultate il SUPPORT\\_POLICY.rst file nel relativo GitHub repository.](#)

Quando gli aggiornamenti includono nuove funzionalità che richiedono una configurazione speciale per evitare errori di crittografia o decrittografia, forniamo una versione intermedia e istruzioni dettagliate per il suo utilizzo. Ad esempio, le versioni 1.7. x e 1.8. x sono progettate per essere versioni transitorie che consentono l'aggiornamento da versioni precedenti alla 1.7. x alle versioni 2.0. x e versioni successive. Per informazioni dettagliate, vedi [Migrazione della tuaAWS Encryption SDK](#).

## Note

La x in un numero di versione rappresenta qualsiasi patch della versione principale e secondaria. Ad esempio, la versione 1.7. x rappresenta tutte le versioni che iniziano con 1.7, incluse 1.7.1 e 1.7.9.

Le nuove funzionalità di sicurezza sono state originariamente rilasciate nelle versioni 1.7 di AWS Encryption CLI. x e 2.0. x. Tuttavia, AWS Encryption CLI versione 1.8. x sostituisce la versione 1.7. x e AWS Encryption CLI 2.1. x sostituisce 2.0. x. Per i dettagli, [consulta l'avviso di sicurezza](#) pertinente nel [aws-encryption-sdk-clirepository](#) su GitHub

Le tabelle seguenti forniscono una panoramica delle principali differenze tra le versioni supportate di AWS Encryption SDK per ogni linguaggio di programmazione.

## C

Per una descrizione dettagliata di tutte le modifiche, vedere [ChangeLog.md](#) nel repository su. [aws-encryption-sdk-c](#) GitHub

Versione principale	Dettagli	Fase del ciclo di vita della versione principale dell'SDK
1.x	1.0	Initial release.
	1.7	Updates to the AWS Encryption SDK that help users of earlier versions upgrade to versions 2.0.x and later. For more information, see <a href="#">versione 1.7. x.</a>
2.x	2.0	Updates to the AWS Encryption SDK. For more information, see <a href="#">versione 2.0. x.</a>
	2.2	Improvements to the message decryption process.
	2.3	Adds support for AWS KMS multi-Region keys.

## C# /.NET

Per una descrizione dettagliata di tutte le modifiche, consulta [ChangeLog.md](#) nel repository su. [aws-encryption-sdk-net](#) GitHub

Versione principale	Dettagli		Fase del ciclo di vita della versione principale dell'SDK
3.x	3.0	Initial release.	<a href="#">Disponibilità generale (GA)</a>  La versione 3.x di per.NET entrerà in modalità manutenzione il 13 maggio 2024. AWS Encryption SDK
4.x	4.0	Adds support for the AWS KMS Hierarchical keyring, the required encryption context CMM, and asymmetric RSA AWS KMS keyrings.	<a href="#">Disponibilità generale (GA)</a>

## Interfaccia a riga di comando (CLI)

Per una descrizione dettagliata di tutte le modifiche, consulta [Versioni delAWS Criptazione CLI](#) e il file [ChangeLog.rst](#) nel repository su. [aws-encryption-sdk-cli](#) GitHub

Versione principale	Dettagli		Fase del ciclo di vita della versione principale dell'SDK
1.x	1.0	Initial release.	<a href="#">Fase di fine del supporto</a>
	1.7	Updates to the AWS Encryption SDK that help users of earlier versions upgrade	

		to versions 2.0.x and later. For more information, see <a href="#">versione 1.7. x.</a>	
2.x	2.0	Updates to the AWS Encryption SDK. For more information, see <a href="#">versione 2.0. x.</a>	<a href="#">Fase di fine del supporto</a>
	2.1	Rimuove il --discovery parametro e lo sostituisce con l'discovery attributo del parametro. --wrapping-keys  La versione 2.1.0 della AWS Encryption CLI è equivalente alla versione 2.0 in altri linguaggi di programmazione.	
	2.2	Improvements to the message decryption process.	
3.x	3.0	Adds support for AWS KMS multi-Region keys.	<a href="#">Fase di fine del supporto</a>

4.x	4.0	The AWS Encryption CLI no longer supports Python 2 or Python 3.4. As of major version 4.x of the AWS Encryption CLI, only Python 3.5 or later is supported.	<a href="#">Disponibilità generale (GA)</a>
	4.1	The AWS Encryption CLI no longer supports Python 3.5. As of version 4.1.x of the AWS Encryption CLI, only Python 3.6 or later is supported.	
	4.2	The AWS Encryption CLI no longer supports Python 3.6. As of version 4.2.x of the AWS Encryption CLI, only Python 3.7 or later is supported.	

## Java

Per una descrizione dettagliata di tutte le modifiche, consulta [ChangeLog.rst](#) nel repository su. [aws-encryption-sdk-java](#) GitHub

Versione principale	Dettagli	Fase del ciclo di vita della versione principale dell'SDK
1.x	1.0	Initial release. <a href="#">Fase di fine del supporto</a>

	1.3	Adds support for cryptographic materials manager and data key caching. Moved to deterministic IV generation.	
	1.6.1	Deprecates and replaces <code>encryptString()</code> and <code>decryptString()</code> with <code>encryptData()</code> and <code>decryptData()</code> .	
	1.7	Updates to the AWS Encryption SDK that help users of earlier versions upgrade to versions 2.0.x and later. For more information, see <a href="#">versione 1.7. x</a> .	
2.x	2.0	Updates to the AWS Encryption SDK. For more information, see <a href="#">versione 2.0. x</a> .	<a href="#">Disponibilità generale (GA)</a>
	2.2	Improvements to the message decryption process.	La versione 2.x di SDK di crittografia AWS per Java entrerà in modalità manutenzione nel 2024.

	2.3	Adds support for AWS KMS multi-Region keys.	
	2.4	Adds support for AWS SDK for Java 2.x.	
3.x	3.0	<p>Si integra SDK di crittografia AWS per Java con la libreria dei fornitori di materiali.</p> <p>Aggiunge il supporto per portachiavi RSA simmetrici e asimmetrici, portachiavi AWS KMS gerarchici, AWS KMS portachiavi Raw AES, portachiavi Raw RSA, portachiavi multipli e il contesto di crittografia richiesto CMM.</p>	<a href="#">Disponibilità generale (GA)</a>

## JavaScript

[Per una descrizione dettagliata di tutte le modifiche, consulta ChangeLog.md nel repository su. aws-encryption-sdk-javascript](#) [GitHub](#)

Versione principale	Dettagli		Fase del ciclo di vita della versione principale dell'SDK
1.x	1.0	Initial release.	<a href="#">Fase di fine del supporto</a>
	1.7	Updates to the AWS Encryption SDK that	

		help users of earlier versions upgrade to versions 2.0.x and later. For more information, see <a href="#">versione 1.7. x.</a>	
2.x	2.0	Updates to the AWS Encryption SDK. For more information, see <a href="#">versione 2.0. x.</a>	<a href="#">Fase di fine del supporto</a>
	2.2	Improvements to the message decryption process.	
	2.3	Adds support for AWS KMS multi-Region keys.	
3.x	3.0	Removes CI coverage for Node 10. Upgrades dependencies to no longer support Node 8 and Node 10.	<a href="#">Maintenance</a> (Manutenzione)  Il supporto per la versione 3.x di SDK di crittografia AWS per JavaScript terminerà il 17 gennaio 2024.
4.x	4.0	Requires version 3 of the SDK di crittografia AWS per JavaScript's <code>client kms</code> to use the AWS KMS keyring.	<a href="#">Disponibilità generale</a> (GA)

# Python

Per una descrizione dettagliata di tutte le modifiche, consulta [ChangeLog.rst](#) nel repository su [aws-encryption-sdk-python](#) GitHub

Versione principale	Dettagli	Fase del ciclo di vita della versione principale dell'SDK
1.x	1.0	Initial release.
	1.3	Adds support for cryptographic materials manager and data key caching. Moved to deterministic IV generation.
	1.7	Updates to the AWS Encryption SDK that help users of earlier versions upgrade to versions 2.0.x and later. For more information, see <a href="#">versione 1.7. x</a> .
2.x	2.0	Updates to the AWS Encryption SDK. For more information, see <a href="#">versione 2.0. x</a> .
	2.2	Improvements to the message decryption process.

	2.3	Adds support for AWS KMS multi-Region keys.	
3.x	3.0	The SDK di crittografia AWS per Python no longer supports Python 2 or Python 3.4. As of major version 3.x of the SDK di crittografia AWS per Python, only Python 3.5 or later is supported.	<a href="#">Disponibilità generale (GA)</a>

## Dettagli della versione

L'elenco seguente descrive le principali differenze tra le versioni supportate di AWS Encryption SDK.

### Argomenti

- [Versioni precedenti alla 1.7. x](#)
- [Versione 1.7. x](#)
- [Versione 2.0. x](#)
- [Versione 2.2. x](#)
- [Versione 2.3. x](#)

### Versioni precedenti alla 1.7. x

#### Note

Tutto 1. x. le versioni x di AWS Encryption SDK sono in [end-of-supportfase](#). Effettua l'aggiornamento all'ultima versione disponibile di AWS Encryption SDK per il tuo linguaggio di programmazione non appena possibile. Per eseguire l'aggiornamento da una AWS Encryption SDK versione precedente alla 1.7. x, è necessario prima eseguire l'aggiornamento

alla versione 1.7. x. Per informazioni dettagliate, vedi [Migrazione della tuaAWS Encryption SDK](#).

Versioni AWS Encryption SDK precedenti alla 1.7. x forniscono importanti funzionalità di sicurezza, tra cui la crittografia con l'algoritmo Advanced Encryption Standard in Galois/Counter Mode (AES-GCM), una funzione di derivazione delle extract-and-expand chiavi basata su HMAC (HKDF), la firma e una chiave di crittografia a 256 bit. [Tuttavia, queste versioni non supportano le migliori pratiche consigliate, incluso l'impegno chiave.](#)

## Versione 1.7. x

### Note

Tutto 1. x. le versioni x di AWS Encryption SDK sono in [end-of-supportfase](#).

Versione 1.7. x è progettato per aiutare gli utenti delle versioni precedenti di AWS Encryption SDK a eseguire l'aggiornamento alle versioni 2.0. x e versioni successive. Se non conosci ilAWS Encryption SDK, puoi saltare questa versione e iniziare con l'ultima versione disponibile nel tuo linguaggio di programmazione.

Versione 1.7. x è completamente compatibile con le versioni precedenti; non introduce modifiche sostanziali né modifica il comportamento di. AWS Encryption SDK È anche compatibile con le versioni precedenti; consente di aggiornare il codice in modo che sia compatibile con la versione 2.0. x. Include nuove funzionalità, ma non le abilita completamente. Inoltre, richiede valori di configurazione che impediscano di adottare immediatamente tutte le nuove funzionalità finché non si è pronti.

Versione 1.7. x include le seguenti modifiche:

### AWS KMSaggiornamenti del provider di chiavi principali (obbligatori)

Versione 1.7. x introduce nuovi costruttori SDK di crittografia AWS per Java e SDK di crittografia AWS per Python che creano esplicitamente fornitori di chiavi AWS KMS principali in modalità rigorosa o di scoperta. Questa versione aggiunge modifiche simili all'interfaccia della AWS Encryption SDK riga di comando (CLI). Per informazioni dettagliate, vedi [Aggiornamento dei provider di chiaviAWS KMS principali](#).

- In modalità rigorosa, i fornitori di chiavi AWS KMS master richiedono un elenco di chiavi di wrapping e crittografano e decrittografano solo con le chiavi di wrapping specificate dall'utente. Si tratta di una procedura AWS Encryption SDK consigliata che garantisce l'utilizzo delle chiavi di wrapping che si intende utilizzare.
- In modalità discovery, i fornitori di chiavi AWS KMS master non accettano chiavi di wrapping. Non è possibile utilizzarli per la crittografia. Durante la decrittografia, possono utilizzare qualsiasi chiave di wrapping per decrittografare una chiave di dati crittografata. Tuttavia, è possibile limitare le chiavi di wrapping utilizzate per la decrittografia a quelle in particolare. Account AWS Il filtraggio degli account è facoltativo, ma è una [best practice](#) che consigliamo.

I costruttori che creano versioni precedenti dei provider di chiavi AWS KMS principali sono obsoleti nella versione 1.7. x e rimosso nella versione 2.0. x. Questi costruttori istanziano i fornitori di chiavi principali che crittografano utilizzando le chiavi di wrapping specificate. Tuttavia, decriptano le chiavi di dati crittografate utilizzando la chiave di wrapping che le ha crittografate, indipendentemente dalle chiavi di wrapping specificate. Gli utenti potrebbero decifrare involontariamente i messaggi con chiavi di wrapping che non intendono utilizzare, anche in altre aree geografiche. AWS KMS keys Account AWS

Non sono state apportate modifiche ai costruttori delle chiavi master. AWS KMS Durante la crittografia e la decrittografia, le chiavi AWS KMS master utilizzano solo quelle specificate dall'AWS KMS keyutente.

### AWS KMSaggiornamenti dei portachiavi (facoltativi)

Versione 1.7. x aggiunge un nuovo filtro alle SDK di crittografia AWS per JavaScript implementazioni SDK di crittografia AWS per C e limita i [portachiavi AWS KMS Discovery a particolari](#). Account AWS Questo nuovo filtro per gli account è facoltativo, ma è una [best practice](#) che consigliamo. Per informazioni dettagliate, vedi [Aggiornamento in corsoAWS KMSportachiavi](#).

Non sono state apportate modifiche ai costruttori dei AWS KMS portachiavi. I AWS KMS portachiavi standard si comportano come fornitori di chiavi principali in modalità rigorosa. AWS KMSi portachiavi discovery vengono creati esplicitamente in modalità discovery.

### Passare un ID chiave a Decrypt AWS KMS

A partire dalla versione 1.7. x, [quando decrittografa le chiavi di dati crittografate, specifica AWS Encryption SDK sempre un AWS KMS key nelle sue chiamate all'AWS KMSoperazione Decrypt](#). AWS Encryption SDKottiene il valore dell'ID della chiave AWS KMS key dai metadati in ogni chiave di dati crittografata. Questa funzionalità non richiede alcuna modifica al codice.

[Non AWS KMS key è necessario specificare l'ID della chiave per decrittografare il testo cifrato crittografato con una chiave KMS di crittografia simmetrica, ma è una procedura consigliata.](#)

[AWS KMS](#) Analogamente a specificare le chiavi di wrapping nel provider di chiavi, questa pratica garantisce che la decrittografia venga decrittografata solo utilizzando la chiave di wrapping che si intende utilizzare AWS KMS.

## Decrittografa il testo cifrato con un impegno chiave

Versione 1.7. x [può decrittografare il testo cifrato che è stato crittografato con o senza l'impegno della chiave](#). Tuttavia, non può crittografare il testo cifrato con un impegno chiave. Questa proprietà consente di implementare completamente le applicazioni in grado di decrittografare il testo cifrato crittografato con impegno chiave prima che incontrino tale testo cifrato. Poiché questa versione decrittografa i messaggi crittografati senza impegno di chiave, non è necessario crittografare nuovamente il testo cifrato.

Per implementare questo comportamento, versione 1.7. x include una nuova impostazione di configurazione [della politica di impegno](#) che determina se è AWS Encryption SDK possibile crittografare o decrittografare con l'impegno della chiave. Nella versione 1.7. x, l'unico valore valido per la policy di impegno `ForbidEncryptAllowDecrypt`, viene utilizzato in tutte le operazioni di crittografia e decrittografia. Questo valore AWS Encryption SDK impedisce la crittografia con una delle nuove suite di algoritmi che includono l'impegno chiave. Consente di AWS Encryption SDK decrittografare il testo cifrato con e senza impegno chiave.

Sebbene nella versione 1.7 esista un solo valore di policy di impegno valido. x, richiediamo che tu possa impostare questo valore in modo esplicito quando utilizzi le nuove API introdotte in questa versione. L'impostazione esplicita del valore impedisce che la politica di impegno venga modificata automaticamente al `require-encrypt-require-decrypt` momento dell'aggiornamento alla versione 2.1. x. Puoi invece [migrare la tua politica di impegno](#) in più fasi.

## Suite di algoritmi con impegno chiave

Versione 1.7. x include due nuove [suite di algoritmi](#) che supportano l'impegno chiave. Una include la firma, l'altra no. Come le suite di algoritmi supportate in precedenza, entrambe queste nuove suite di algoritmi includono la crittografia con AES-GCM, una chiave di crittografia a 256 bit e una funzione di derivazione delle chiavi basata su HMAC ( `extract-and-expand HKDF`).

Tuttavia, la suite di algoritmi predefinita utilizzata per la crittografia non cambia. Queste suite di algoritmi vengono aggiunte alla versione 1.7. x per preparare l'applicazione a utilizzarli nelle versioni 2.0. x e versioni successive.

## Modifiche all'implementazione della CMM

Versione 1.7. x introduce modifiche all'interfaccia CMM (Default cryptographic materials manager) per supportare l'impegno chiave. Questa modifica ha effetto solo se hai scritto una CMM personalizzata. Per i dettagli, consulta la documentazione o l' [GitHub](#) archivio dell'API per il tuo [linguaggio di programmazione](#).

## Versione 2.0. x

Versione 2.0. x supporta le nuove funzionalità di sicurezza offerte da AWS Encryption SDK, tra cui chiavi di avvolgimento specifiche e Key Commitment. Per supportare queste funzionalità, la versione 2.0. x include modifiche sostanziali per le versioni precedenti di AWS Encryption SDK. È possibile prepararsi a queste modifiche distribuendo la versione 1.7. x. Versione 2.0. x include tutte le nuove funzionalità introdotte nella versione 1.7. x con le seguenti aggiunte e modifiche.

### Note

Versione 2. x. x [di SDK di crittografia AWS per Python](#) [SDK di crittografia AWS per JavaScript](#), [e la CLI di AWS crittografia](#) sono in fase di elaborazione. [end-of-support](#)

Per informazioni sul [supporto e la manutenzione](#) di questa AWS Encryption SDK versione nel linguaggio di programmazione preferito, consultate il `SUPPORT_POLICY.rst` file nel relativo [GitHub repository](#).

## AWS KMS fornitori di chiavi principali

I costruttori originali del provider di chiavi AWS KMS principali che erano obsoleti nella versione 1.7. x vengono rimossi nella versione 2.0. x. È necessario creare esplicitamente fornitori di chiavi AWS KMS principali in [modalità rigorosa o in modalità di scoperta](#).

## Crittografa e decrittografa il testo cifrato con impegno chiave

Versione 2.0. x [può crittografare e decrittografare il testo cifrato con o senza l'impegno di una chiave](#). Il suo comportamento è determinato dall'impostazione della politica di impegno. Per impostazione predefinita, crittografa sempre con l'impegno della chiave e decrittografa solo il testo cifrato crittografato con l'impegno della chiave. A meno che non si modifichi la politica di commit, non AWS Encryption SDK decrittogherà i testi cifrati crittografati con nessuna versione precedente di, inclusa la versione 1.7. AWS Encryption SDK x.

**⚠ Important**

Per impostazione predefinita, versione 2.0. x non decrittograferà alcun testo cifrato che è stato crittografato senza l'impegno di una chiave. Se nell'applicazione viene rilevato un testo cifrato che è stato crittografato senza l'utilizzo di una chiave, impostate un valore relativo alla policy di commit con. `AllowDecrypt`

Nella versione 2.0. x, l'impostazione della politica di impegno ha tre valori validi:

- `ForbidEncryptAllowDecrypt`— AWS Encryption SDK Non possono crittografare con un impegno chiave. Può decrittografare testi cifrati crittografati con o senza impegno di chiave.
- `RequireEncryptAllowDecrypt`— AWS Encryption SDK Devono crittografare con impegno chiave. Può decrittografare testi cifrati crittografati con o senza impegno di chiave.
- `RequireEncryptRequireDecrypt`(impostazione predefinita): AWS Encryption SDK deve crittografare con un impegno chiave. Decrypta i testi cifrati solo con un impegno chiave.

Se stai migrando da una versione precedente di alla versione 2.0. AWS Encryption SDK x, imposta la politica di impegno su un valore che garantisca la possibilità di decrittografare tutti i testi cifrati esistenti che l'applicazione potrebbe incontrare. È probabile che modificherai questa impostazione nel tempo.

## Versione 2.2. x

Aggiunge il supporto per le firme digitali e limita le chiavi di dati crittografate.

**i Note**

Versione 2. x. x [di SDK di crittografia AWS per Python](#) [SDK di crittografia AWS per JavaScript](#), e la [CLI di AWS crittografia sono in fase di elaborazione. end-of-support](#)

Per informazioni sul [supporto e la manutenzione](#) di questa AWS Encryption SDK versione nel linguaggio di programmazione preferito, consultate il `SUPPORT_POLICY.rst` file nel relativo [GitHub repository](#).

## Firme digitali

Per migliorare la gestione delle [firme digitali](#) durante la decrittografia, AWS Encryption SDK include le seguenti funzionalità:

- Modalità non streaming: restituisce il testo in chiaro solo dopo aver elaborato tutti gli input, inclusa la verifica della firma digitale, se presente. Questa funzionalità impedisce di utilizzare testo in chiaro prima di verificare la firma digitale. Utilizza questa funzionalità ogni volta che decifri i dati crittografati con firme digitali (la suite di algoritmi predefinita). Ad esempio, poiché l'AWSEncryption CLI elabora sempre i dati in modalità streaming, utilizza il `-buffer` parametro per decrittografare il testo cifrato con firme digitali.
- Modalità di decrittografia solo senza segno: questa funzionalità decrittografa solo testo cifrato non firmato. Se la decrittografia rileva una firma digitale nel testo cifrato, l'operazione ha esito negativo. Utilizzate questa funzionalità per evitare l'elaborazione involontaria del testo in chiaro dei messaggi firmati prima di verificare la firma.

### Limitazione delle chiavi dati crittografate

È possibile [limitare il numero di chiavi dati crittografate](#) in un messaggio crittografato. Questa funzionalità può aiutarti a rilevare un provider di chiavi master o un portachiavi configurato in modo errato durante la crittografia o a identificare un testo cifrato dannoso durante la decrittografia.

È necessario limitare le chiavi di dati crittografate quando si decifrano i messaggi da una fonte non attendibile. Previene chiamate inutili, costose e potenzialmente esaustive all'infrastruttura chiave.

## Versione 2.3. x

Aggiunge il supporto per le chiavi AWS KMS multiregionali. Per informazioni dettagliate, vedi [Utilizzo di più regioni AWS KMS keys](#).

### Note

L'AWSEncryption CLI supporta chiavi multiregionali a partire dalla versione 3.0. x.

Versione 2. x. x [di SDK di crittografia AWS per PythonSDK di crittografia AWS per JavaScript, e la CLI di AWS crittografia sono in fase end-of-support](#).

Per informazioni sul [supporto e la manutenzione](#) di questa AWS Encryption SDK versione nel linguaggio di programmazione preferito, consultate il `SUPPORT_POLICY.rs` file nel relativo [GitHub repository](#).

# Migrazione della tua AWS Encryption SDK

Il AWS Encryption SDK supporta più interoperabili [implementazioni del linguaggio di programmazione](#), ognuno dei quali è sviluppato in un repository open source su GitHub. Come un [best practice](#), si consiglia di utilizzare la versione più recente di AWS Encryption SDK per ogni lingua.

È possibile eseguire l'aggiornamento in sicurezza dalla versione 2.0.xo successivamente di AWS Encryption SDK alla versione più recente. Tuttavia, il 2.0.x versione del AWS Encryption SDK introduce nuove importanti funzionalità di sicurezza, alcune delle quali sono cambiamenti epocali. Per eseguire l'aggiornamento da versioni precedenti alla 1.7.x alle versioni 2.0.xe successivamente, devi prima effettuare l'aggiornamento alla versione più recente 1.x versione. Gli argomenti di questa sezione sono pensati per aiutarti a comprendere le modifiche, selezionare la versione corretta per la tua applicazione e migrare in modo sicuro e corretto alle versioni più recenti di AWS Encryption SDK.

Per informazioni sulle versioni significative di AWS Encryption SDK, vedi [Versioni del AWS Encryption SDK](#).

## Important

Non eseguire l'aggiornamento direttamente da una versione precedente alla 1.7.x alla versione 2.0.xo successivamente senza prima effettuare l'aggiornamento alla versione più recente 1.x versione. Se si esegue l'aggiornamento direttamente alla versione 2.0.xo successivamente e abilita immediatamente tutte le nuove funzionalità, il AWS Encryption SDK non sarà in grado di decrittografare il testo crittografata in versioni precedenti di AWS Encryption SDK.

## Note

La prima versione di AWS Encryption SDK per .NET è versione 3.0.x. Tutte le versioni di AWS Encryption SDK per .NET supporta le migliori pratiche di sicurezza introdotte nella versione 2.0.x del AWS Encryption SDK. Puoi eseguire l'aggiornamento alla versione più recente in tutta sicurezza senza modifiche al codice o ai dati.

AWSCrittografia CLI: Quando leggi questa guida alla migrazione, usa la versione 1.7.x istruzioni di migrazione per AWSCrittografia CLI 1.8.xe usa il 2.0.x istruzioni di migrazione per AWSCrittografia CLI 2.1.x. Per dettagli, consulta [Versioni del AWS Criptazione CLI](#).

Le nuove funzionalità di sicurezza sono state inizialmente rilasciate in AWS Crittografia CLI versione 1.7.x. Tuttavia AWS Encryption CLI versione 1.8.x sostituisce versione 1.7.x e AWS Crittografia CLI 2.1.x sostituisce 2.0.x. Per informazioni dettagliate, consulta la sezione corrispondente [consulenza sulla sicurezza](#) nel [aws-encryption-sdk-cli](#) repository su GitHub.

## Nuovi utenti

Se non hai mai usato il plugin AWS Encryption SDK, installa la versione più recente di AWS Encryption SDK per il tuo linguaggio di programmazione. I valori predefiniti abilitano tutte le funzionalità di sicurezza di AWS Encryption SDK, inclusa la crittografia con firma, derivazione delle chiavi e [impegno chiave](#). di AWS Encryption SDK

## Utenti attuali

Si consiglia di aggiornare la versione corrente alla versione più recente disponibile il prima possibile. Tutti i versioni del AWS Encryption SDK sono nella [end-of-support fase](#), così come le versioni successive in alcuni linguaggi di programmazione. Per informazioni dettagliate sullo stato di supporto e manutenzione del AWS Encryption SDK nel tuo linguaggio di programmazione, vedi [Support e manutenzione](#).

AWS Encryption SDK versioni 2.0.x in seguito fornisce nuove funzionalità di sicurezza per proteggere i dati. Tuttavia AWS Encryption SDK versione 2.0.x include modifiche importanti che non sono retrocompatibili. Per garantire una transizione sicura, inizia con la migrazione dalla versione corrente alla versione più recente 1.x nel tuo linguaggio di programmazione. Quando il tuo ultimo 1.x la versione è completamente distribuita e funziona correttamente, è possibile migrare in sicurezza alle versioni 2.0.x successive. Questo [processo in due fasi](#) è fondamentale soprattutto per le applicazioni distribuite.

Per ulteriori informazioni sulla AWS Encryption SDK caratteristiche di sicurezza alla base di queste modifiche, vedi [Crittografia lato client migliorata: Esplicito KeyIds e impegno chiave](#) nel AWS Blog sulla sicurezza.

In cerca di aiuto per l'utilizzo di SDK di crittografia AWS per Java con il AWS SDK for Java 2.x? Per informazioni, consulta [Prerequisiti](#).

## Argomenti

- [Come migrare e implementare il AWS Encryption SDK](#)

- [Aggiornamento dei provider di chiaviAWS KMS principali](#)
- [Aggiornamento in corsoAWS KMSportachiavi](#)
- [Impostazione della politica di impegno](#)
- [Risoluzione dei problemi relativi alla migrazione alle versioni più recenti](#)

## Come migrare e implementare ilAWS Encryption SDK

Durante la migrazione da unAWS Encryption SDKversione precedente alla 1.7.xalla versione 2.0.xo successivamente, è necessario passare in modo sicuro alla crittografia con[impegno chiave](#). In caso contrario, l'applicazione incontrerà testi cifrati che non potrà decifrare. Se utilizziAWS KMSprovider di chiavi master, è necessario eseguire l'aggiornamento a nuovi costruttori che creano provider di chiavi master in modalità rigorosa o modalità di rilevamento.

### Note

Questo argomento è destinato agli utenti che eseguono la migrazione da versioni precedenti diAWS Encryption SDKalla versione 2.0.xo successive. Se sei nuovo aAWS Encryption SDK, puoi iniziare a utilizzare immediatamente l'ultima versione disponibile con le impostazioni predefinite.

Per evitare una situazione critica in cui non è possibile decrittografare il testo cifrato che è necessario leggere, si consiglia di eseguire la migrazione e la distribuzione in più fasi distinte. Verifica che ogni fase sia completa e completamente implementata prima di iniziare la fase successiva. Ciò è particolarmente importante per le applicazioni distribuite con più host.

## Fase Aggiornare l'applicazione alla versione più recente 1.xversione

Aggiornare alla versione più recente 1.xversione per il tuo linguaggio di programmazione. Prova con attenzione, distribuisce le modifiche e conferma che l'aggiornamento si sia propagato a tutti gli host di destinazione prima di iniziare la fase 2.

### Important

Verificare che il tuo ultimo file 1.xversione è versione 1.7.xo più tardi delAWS Encryption SDK.

L'ultimo 1.x versioni del AWS Encryption SDK sono retrocompatibili con le versioni precedenti di AWS Encryption SDK e successivamente compatibile con le versioni 2.0.x successive. Includono le nuove funzionalità presenti in versione 2.0.x, ma includono impostazioni predefinite sicure progettate per questa migrazione. Ti consentono di aggiornare il tuo AWS KMS fornitori di chiavi master, se necessario, e da implementare completamente con suite di algoritmi in grado di decrittografare il testo cifrato con l'impegno della chiave.

- Sostituisci gli elementi obsoleti, inclusi i costruttori con quelli precedenti AWS KMS provider di chiavi master. In [Python](#), assicurati di attivare gli avvisi di obsolescenza. Elementi di codice obsoleti nell'ultima versione 1.x le versioni vengono rimosse dalle versioni 2.0.x successive.
- Imposta esplicitamente la tua politica di impegno su `ForbidEncryptAllowDecrypt`. Sebbene questo sia l'unico valore valido nell'ultimo valore 1.x versioni, questa impostazione è richiesta quando si utilizzano le API introdotte in questa versione. Impedisce all'applicazione di rifiutare il testo cifrato crittografato senza l'impegno della chiave durante la migrazione alla versione 2.0.x successive. Per dettagli, consulta [the section called "Impostazione della politica di impegno"](#).
- Se utilizzi AWS KMS fornitori di chiavi master, è necessario aggiornare i provider di chiavi master esistenti con i provider di chiavi master che supportano modalità rigorosa e modalità Discovery. Questo aggiornamento è necessario per il SDK di crittografia AWS per Java, SDK di crittografia AWS per Python e il AWS Crittografia CLI. Se si utilizzano i provider di chiavi master in modalità discovery, si consiglia di implementare il filtro di rilevamento che limita le chiavi di wrapping utilizzate a quelle in particolare Account AWS. Questo aggiornamento è facoltativo, ma è un [best practice](#) che consigliamo. Per dettagli, consulta [Aggiornamento dei provider di chiavi AWS KMS principali](#).
- Se utilizzi [AWS KMS port chiavi Discovery](#), ti consigliamo di includere un filtro di rilevamento che limiti le chiavi di wrapping utilizzate per la decrittografia a quelle in particolare Account AWS. Questo aggiornamento è facoltativo, ma è un [best practice](#) che consigliamo. Per dettagli, consulta [Aggiornamento in corso AWS KMS port chiavi](#).

## Fase Aggiornare l'applicazione alla versione più recente

Dopo aver implementato l'ultima versione 1.x versione con successo per tutti gli host, è possibile eseguire l'aggiornamento alle versioni 2.0.x dopo. Versione 2.0 include modifiche sostanziali per tutte le versioni precedenti di AWS Encryption SDK. Tuttavia, se si apportano le modifiche al codice consigliate nella Fase 1, è possibile evitare errori durante la migrazione alla versione più recente.

Prima di eseguire l'aggiornamento alla versione più recente, verifica che la politica di impegno sia impostata in modo coerente su `ForbidEncryptAllowDecrypt`. Quindi, a seconda della configurazione dei dati, puoi migrare secondo i tuoi ritmi verso `RequireEncryptAllowDecrypt` poi all'impostazione predefinita, `RequireEncryptRequireDecrypt`. Consigliamo una serie di passaggi di transizione come il seguente schema.

1. Inizia con il tuo [politica impegno](#) impostata su `ForbidEncryptAllowDecrypt`. Il `AWS Encryption SDK` può decifrare i messaggi con l'impegno della chiave, ma non li crittografa ancora con l'impegno della chiave.
2. Al termine, aggiornare la propria politica impegno ad `RequireEncryptAllowDecrypt`. Il `AWS Encryption SDK` inizia a crittografare i dati con [impegno chiave](#). Può decifrare il testo cifrato con e senza impegno chiave.

Prima di aggiornare la tua politica di impegno a `RequireEncryptAllowDecrypt`, verifica che il tuo ultimo file sia 1.x la versione viene distribuita su tutti gli host, inclusi gli host di qualsiasi applicazione che decifra il testo cifrato prodotto. Versioni del `AWS Encryption SDK` precedente alla versione 1.7.x non può decifrare i messaggi crittografati con l'impegno della chiave.

Questo è anche un buon momento per aggiungere metriche alla tua applicazione per misurare se stai ancora elaborando testo cifrato senza alcun impegno chiave. Questo ti aiuterà a determinare quando è sicuro aggiornare le impostazioni della tua politica di impegno su `RequireEncryptRequireDecrypt`. Per alcune applicazioni, come quelle che crittografano i messaggi in una coda Amazon SQS, ciò potrebbe significare attendere abbastanza a lungo affinché tutto il testo cifrato crittografato nelle versioni precedenti sia stato nuovamente crittografato o eliminato. Per altre applicazioni, come gli oggetti S3 crittografati, potrebbe essere necessario scaricare, crittografare nuovamente e caricare nuovamente tutti gli oggetti.

3. Quando sei sicuro di non avere messaggi crittografati senza impegno chiave, puoi aggiornare la tua politica di impegno su `RequireEncryptRequireDecrypt`. Questo valore garantisce che i tuoi dati siano sempre crittografati e decrittografati con un impegno chiave. Questa è l'impostazione predefinita, quindi non è necessario impostarla in modo esplicito, ma è consigliabile. Un'impostazione esplicita lo farà [aiuto per il debug di aiuto](#) e qualsiasi potenziale rollback che potrebbe essere necessario se l'applicazione rileva testo cifrato crittografato senza impegno di chiave.

# Aggiornamento dei provider di chiaviAWS KMS principali

Per migrare alla versione più recente 1. versione x diAWS Encryption SDK, e quindi alla versione 2.0. x o versione successiva, è necessario sostituire i provider di chiaviAWS KMS master esistenti con provider di chiavi master creati esplicitamente in [modalità rigorosa o modalità discovery](#). I vecchi fornitori di chiavi master sono obsoleti nella versione 1.7. x e rimosso nella versione 2.0. x. Questa modifica è necessaria per le applicazioni e gli script che utilizzano l'interfaccia a riga di [SDK di crittografia AWS per Java](#) comando [SDK di crittografia AWS per Python](#), e la [CLI diAWS crittografia](#). Gli esempi in questa sezione ti mostreranno come aggiornare il codice.

## Note

In Python, [attiva gli avvisi di deprecazione](#). Questo ti aiuterà a identificare le parti del codice che devi aggiornare.

Se si utilizza una chiaveAWS KMS master (non un provider di chiavi master), è possibile saltare questo passaggio. AWS KMS le chiavi master non sono obsolete o rimosse. Crittografano e decifrano solo con le chiavi di wrapping specificate.

Gli esempi in questa sezione si concentrano sugli elementi del codice che è necessario modificare. Per un esempio completo del codice aggiornato, consulta la sezione Esempi del GitHub repository relativo al tuo [linguaggio di programmazione](#). Inoltre, questi esempi utilizzano in genere gli ARN chiave per rappresentareAWS KMS keys. Quando si crea un provider di chiave master per la crittografia, è possibile utilizzare qualsiasi [identificatore diAWS KMS chiave](#) valido per rappresentare unAWS KMS key. Quando si crea un provider di chiavi master per la decrittografia, è necessario utilizzare una chiave ARN.

Ulteriori informazioni sulla migrazione

Per tuttiAWS Encryption SDK gli utenti, scopri come impostare la tua politica di impegno in [the section called "Impostazione della politica di impegno"](#).

PerSDK di crittografia AWS per C gliSDK di crittografia AWS per JavaScript utenti finali, scopri un aggiornamento opzionale dei portachiavi in [Aggiornamento in corsoAWS KMSportachiavi](#).

Argomenti

- [Migrazione alla modalità](#)
- [Migrazione alla modalità di rilevamento](#)

## Migrazione alla modalità

Dopo l'aggiornamento alla versione più recente 1. versione x di AWS Encryption SDK, sostituisci i tuoi vecchi provider di chiavi master con fornitori di chiavi master in modalità rigorosa. In modalità rigorosa, è necessario specificare le chiavi di wrapping da utilizzare durante la crittografia e la decrittografia. AWS Encryption SDK utilizza solo le chiavi di wrapping specificate. I provider di chiavi master obsolete possono decrittografare i dati utilizzando qualsiasi AWS KMS key chiave di dati crittografata, anche AWS KMS keys in diverse regioni Account AWS e regioni.

I fornitori di chiavi principali in modalità rigorosa vengono introdotti nella AWS Encryption SDK versione 1.7. x. Sostituiscono i vecchi fornitori di chiavi master, che sono obsoleti nella versione 1.7. x e rimosso in 2.0. x. L'utilizzo di provider di chiavi master in modalità rigorosa è una AWS Encryption SDK [buona pratica](#).

Il codice seguente crea un provider di chiavi master in modalità rigorosa che è possibile utilizzare per la crittografia e la decrittografia.

### Java

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.6.2 o precedente di SDK di crittografia AWS per Java.

Questo codice utilizza il `KmsMasterKeyProvider.builder()` metodo per creare un'istanza di un provider di chiavi AWS KMS master che ne utilizza una AWS KMS key come chiave di wrapping.

```
// 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();
```

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.7. x o versione successiva del SDK di crittografia AWS per Java. Per un esempio completo, vedere [BasicEncryptionExample.java](#).

`Builder.withKeysForEncryption()` I metodi `Builder.build()` and utilizzati nell'esempio precedente sono obsoleti nella versione 1.7. x e vengono rimossi dalla versione 2.0. x.

Per l'aggiornamento a un provider di chiavi master in modalità rigorosa, questo codice sostituisce le chiamate ai metodi obsoleti con una chiamata al nuovo `builder.buildStrict()` metodo. Questo esempio ne specifica una AWS KMS key come chiave di avvolgimento, ma il `builder.buildStrict()` metodo può accettare un elenco di più chiavi AWS KMS keys.

```
// Create a master key provider in strict mode
// Replace the example key ARN with a valid one from your Account AWS.
String awsKmsKey = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

KmsMasterKeyProvider masterKeyProvider = KmsMasterKeyProvider.builder()
    .buildStrict(awsKmsKey);
```

## Python

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.4.1 di SDK di crittografia AWS per Python. Questo codice utilizza `KMSMasterKeyProvider`, che è obsoleto nella versione 1.7. x e rimosso dalla versione 2.0. x. Durante la decrittografia, utilizza qualsiasi chiave di dati crittografata indipendentemente da AWS KMS key quella AWS KMS keys specificata.

Nota che non `KMSMasterKey` è obsoleto o rimosso. Durante la crittografia e la decrittografia, utilizza solo AWS KMS key i dati specificati.

```
# 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]
)
```

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.7. x del SDK di crittografia AWS per Python. Per un esempio completo, vedere [basic\\_encryption.py](#).

Per l'aggiornamento a un provider di chiavi master in modalità rigorosa, questo codice sostituisce la chiamata a `KMSMasterKeyProvider()` con una chiamata a `StrictAwsKmsMasterKeyProvider()`.

```
# Create a master key provider in strict mode
# Replace the example key ARNs with valid values from your Account 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

Questo esempio mostra come crittografare e decrittografare utilizzando laAWS Encryption CLI versione 1.1.7 o precedente.

Nella versione 1.1.7 e precedenti, durante la crittografia, si specificano una o più chiavi master (o chiavi di wrapping), ad esempio unaAWS KMS key. Durante la decrittografia, non è possibile specificare alcuna chiave di wrapping a meno che non si utilizzi un provider di chiavi master personalizzato. La CLI diAWS crittografia può utilizzare qualsiasi chiave di wrapping che crittografi una chiave di dati.

```
\\ 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 .
```

Questo esempio mostra come crittografare e decrittografare utilizzando la AWS Encryption CLI versione 1.7. x o versioni successive. Per esempi completi, vedere [Esempi di AWS CLI di crittografia](#).

Il `--master-keys` parametro è obsoleto nella versione 1.7. x e rimosso nella versione 2.0. x. È stato sostituito dal `--wrapping-keys` parametro, necessario nei comandi di crittografia e decrittografia. Questo parametro supporta la modalità rigorosa e la modalità di rilevamento. La modalità rigorosa è una AWS Encryption SDK procedura consigliata che garantisce l'utilizzo della chiave di wrapping desiderata.

Per passare alla modalità rigorosa, utilizzate l'attributo `key` del `--wrapping-keys` parametro per specificare una chiave di wrapping durante la crittografia e la decrittografia.

```
\\ 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 .
```

## Migrazione alla modalità di rilevamento

A partire dalla versione 1.7. x, è consigliabile utilizzare [la](#) modalità rigorosa per i provider di chiavi AWS KMS master, ovvero specificare le chiavi di wrapping durante la crittografia e la decrittografia. AWS Encryption SDK È necessario specificare sempre le chiavi di wrapping durante la crittografia. Ma ci sono situazioni in cui specificare gli ARN chiave AWS KMS keys per la decrittografia non è pratico. Ad esempio, se si utilizzano alias per identificare AWS KMS keys durante la crittografia, si perde il vantaggio degli alias se è necessario elencare gli ARN chiave durante la decrittografia.

Inoltre, poiché i provider di chiavi master in modalità discovery si comportano come i provider di chiavi master originali, potresti utilizzarli temporaneamente come parte della tua strategia di migrazione e quindi passare ai provider di chiavi master in modalità rigorosa in un secondo momento.

In casi come questo, puoi utilizzare i provider di chiavi master in modalità discovery. Questi provider di chiavi master non consentono di specificare le chiavi di wrapping, quindi non è possibile utilizzarli per la crittografia. Durante la decrittografia, possono utilizzare qualsiasi chiave di wrapping che crittografi una chiave di dati. Ma a differenza dei provider di chiavi master legacy, che si comportano allo stesso modo, li crei esplicitamente in modalità discovery. Quando si utilizzano provider di chiavi master in modalità discovery, è possibile limitare le chiavi di wrapping che possono essere utilizzate a quelle in particolareAccount AWS. Questo filtro di rilevamento è facoltativo, ma è una procedura consigliata. Per informazioni suAWS partizioni e account, consulta [Amazon Resource Names](#) nel Riferimenti generali di AWS.

Gli esempi seguenti creano un provider di chiaviAWS KMS master in modalità rigorosa per la crittografia e un provider di chiaviAWS KMS master in modalità discovery per la decrittografia. Il provider di chiavi master in modalità discovery utilizza un filtro di rilevamento per limitare le chiavi di wrapping utilizzate per la decrittografia allaaws partizione e a un esempio particolareAccount AWS. Sebbene il filtro dell'account non sia necessario in questo esempio molto semplice, è una buona pratica molto utile quando un'applicazione crittografa i dati e un'altra applicazione li decrittografa.

## Java

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.7. x o versione successiva delSDK di crittografia AWS per Java. Per un esempio completo, vedere [DiscoveryDecryptionExample.java](#).

Per creare un'istanza di un provider di chiavi master in modalità rigorosa per la crittografia, questo esempio utilizza ilBuilder.buildStrict() metodo. Per creare un'istanza di un provider di chiavi master in modalità discovery per la decrittografia, utilizza ilBuilder.buildDiscovery() metodo. IlBuilder.buildDiscovery() metodo accetta un valoreDiscoveryFilter che limita l'AWS KMS keysaccessoAWS Encryption SDK allaAWS partizione e agli account specificati.

```
// Create a master key provider in strict mode for encrypting
// Replace the example alias ARN with a valid one from your Account 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

Questo esempio rappresenta il codice di un'applicazione che utilizza la versione 1.7. x o versione successiva del SDK di crittografia AWS per Python. Per un esempio completo, vedere [discovery\\_kms\\_provider.py](#).

Per creare un provider di chiavi master in modalità rigorosa per la crittografia, questo esempio utilizza `StrictAwsKmsMasterKeyProvider`. Per creare un provider di chiavi master in modalità discovery per la decrittografia, utilizza `DiscoveryAwsKmsMasterKeyProvider` un `DiscoveryFilter` che limita il numero di AWS KMS keys nella partizione e negli account specificati.

```
# Create a master key provider in strict mode
# Replace the example key ARN and alias ARNs with valid values from your Account
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

Questo esempio mostra come crittografare e decrittografare utilizzando la AWS Encryption CLI versione 1.7. x o versioni successive. A partire dalla versione 1.7. x, il `--wrapping-keys` parametro è necessario per la crittografia e la decrittografia. Il `--wrapping-keys` parametro supporta la modalità rigorosa e la modalità di rilevamento. Per esempi completi, vedere [the section called “Esempi”](#).

Durante la crittografia, questo esempio specifica una chiave di wrapping, che è obbligatoria. Durante la decrittografia, sceglie esplicitamente la modalità di rilevamento utilizzando l'attributo `discovery` del `--wrapping-keys` parametro con un valore di `true`.

Per limitare le chiavi di wrapping che AWS Encryption SDK possono essere utilizzate in modalità `discovery` a quelle in particolare Account AWS, questo esempio utilizza gli attributi `discovery-account` e `discovery-partition` and del `--wrapping-keys` parametro. Questi attributi opzionali sono validi solo quando l'attributo `discovery` è impostato su `true`. È necessario utilizzare `discovery-account` e `discovery-partition` insieme; nessuno dei due è valido da solo.

```

\\ 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 .

```

## Aggiornamento in corsoAWS KMSportachiavi

IlAWS KMSportachiavi in[SDK di crittografia AWS per C](#), il[AWS Encryption SDKper .NET](#)e il[SDK di crittografia AWS per JavaScript](#)supportare[best practice](#)permettendo di specificare le chiavi di wrapping durante la crittografia e la decrittografia. Se crei un[AWS KMSportachiavi discovery](#), lo fai esplicitamente.

### Note

La prima versione diAWS Encryption SDKfor .NET è versione 3.0.x. Tutte le versioni diAWS Encryption SDKper .NET supporta le migliori pratiche di sicurezza introdotte nella versione 2.0.xdelAWS Encryption SDK. Puoi passare in sicurezza alla versione più recente senza modifiche al codice o ai dati.

Quando esegui l'aggiornamento alla versione più recente 1.xversione delAWS Encryption SDK, puoi usare un[filtro di individuazione](#)per limitare le chiavi di involucro che un[AWS KMSportachiavi discovery](#)o[AWS KMSportachiavi di individuazione regionale](#)utilizza durante la decrittografia per quelli in particolareAccount AWS. Il filtraggio di un portachiavi di scoperta è unAWS Encryption SDK [best practice](#).

Gli esempi in questa sezione mostrano come aggiungere il filtro di scoperta a unAWS KMSportachiavi Regional Discovery.

Ulteriori informazioni sulla migrazione

Per tuttiAWS Encryption SDKutenti, scopri come impostare la tua politica di impegno in[the section called "Impostazione della politica di impegno"](#).

PerSDK di crittografia AWS per Java,SDK di crittografia AWS per PythoneAWSUtenti di Encryption CLI, scoprite l'aggiornamento richiesto per i provider di chiavi master in[the section called "Aggiornamento dei provider di chiaviAWS KMS principali"](#).

Potresti avere un codice simile al seguente nella tua applicazione. Questo esempio crea unAWS KMSportachiavi di individuazione regionale che può utilizzare solo chiavi di terminazione di terminata nella regione Stati Uniti occidentali (Oregon) (us-west-2). Questo esempio rappresenta il codice inAWS Encryption SDKversioni precedenti alla 1.7.x. Tuttavia, è ancora valido nelle versioni 1.7.xe dopo.

## 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 partire dalla versione 1.7.x, puoi aggiungere un filtro di scoperta a qualsiasi AWS KMS sporta chiavi Discovery. Questo filtro di scoperta limita il AWS KMS keys che il AWS Encryption SDK può essere utilizzato per la decrittografia di quelli presenti nella partizione e negli account specificati. Prima di utilizzare questo codice, modificate la partizione, se necessario, e sostituite gli ID account di esempio con quelli validi.

## C

Per un esempio, consulta [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

Per un esempio, consulta [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' }
})
```

## Impostazione della politica di impegno

[L'impegno chiave](#) garantisce che i dati crittografati vengano sempre decrittografati con lo stesso testo in chiaro. Per fornire questa proprietà di sicurezza, a partire dalla versione 1.7. x, AWS Encryption SDK utilizza nuove [suite di algoritmi](#) con impegno chiave. Per determinare se i dati sono crittografati e decrittografati con Key Commitment, utilizza l'impostazione di configurazione della [politica di impegno](#). La crittografia e la decrittografia dei dati con impegno chiave è una [procedura AWS Encryption SDK consigliata](#).

La definizione di una politica di impegno è una parte importante della seconda fase del processo di migrazione: la migrazione dalla più recente 1. x versioni delle AWS Encryption SDK due versioni 2.0. x e versioni successive. Dopo aver impostato e modificato la politica di impegno, assicurati di testare accuratamente l'applicazione prima di distribuirla in produzione. Per indicazioni sulla migrazione, consulta [Come migrare e implementare il AWS Encryption SDK](#).

L'impostazione della politica di impegno ha tre valori validi nelle versioni 2.0. x e versioni successive. Al più recente 1. versioni x (a partire dalla versione 1.7. x), solo `ForbidEncryptAllowDecrypt` è valido.

- `ForbidEncryptAllowDecrypt`—AWS Encryption SDK Non possono cifrare con un impegno chiave. Può decrittografare i testi cifrati con o senza impegno chiave.

Al più recente 1. x versioni, questo è l'unico valore valido. Garantisce che non si esegua la crittografia con un impegno chiave finché non si è completamente preparati a decifrare con un impegno chiave. L'impostazione del valore impedisce esplicitamente che la politica di impegno venga modificata automaticamente `require-encrypt-require-decrypt` quando si esegue l'aggiornamento alla versione 2.0. x o versioni successive. Puoi invece [migrare la tua politica di impegno](#) in più fasi.

- `RequireEncryptAllowDecrypt`— Crittografia AWS Encryption SDK sempre con impegno chiave. Può decrittografare i testi cifrati con o senza impegno chiave. Questo valore viene aggiunto nella versione 2.0. x.
- `RequireEncryptRequireDecrypt`— Crittografia e decrittografia AWS Encryption SDK sempre con impegno chiave. Questo valore viene aggiunto nella versione 2.0. x. È il valore predefinito nelle versioni 2.0. x e versioni successive.

Al più recente 1. x versioni, l'unico valore valido della politica di impegno è `ForbidEncryptAllowDecrypt`. Dopo aver effettuato la migrazione alla versione 2.0. x o successivamente, puoi [modificare la tua politica di impegno gradualmente](#) man mano che sei pronto. Non aggiornate la vostra politica di impegno `RequireEncryptRequireDecrypt` fino a quando non siete certi di non avere messaggi criptati senza impegno chiave.

Questi esempi mostrano come impostare la politica di impegno nella versione più recente 1. versioni x e nelle versioni 2.0. x e versioni successive. La tecnica dipende dal linguaggio di programmazione utilizzato.

Ulteriori informazioni sulla migrazione

Per SDK di crittografia AWS per Java SDK di crittografia AWS per Python, e per la CLI di AWS crittografia, scopri le modifiche necessarie ai provider di chiavi master in [the section called "Aggiornamento dei provider di chiavi AWS KMS principali"](#).

Per SDK di crittografia AWS per C e SDK di crittografia AWS per JavaScript, scopri un aggiornamento opzionale dei portachiavi in [Aggiornamento in corso AWS KMS portachiavi](#).

## Come impostare la tua politica di impegno

La tecnica utilizzata per impostare la politica di impegno varia leggermente a seconda dell'implementazione linguistica. In questi esempi viene mostrato come fare. Prima di modificare la politica di impegno, rivedi l'approccio in più fasi in [Come migrare e implementare](#).

### C

A partire dalla versione 1.7. x di esseSDK di crittografia AWS per C, si utilizza `aws_cryptosdk_session_set_commitment_policy` funzione per impostare la politica di impegno per le sessioni di crittografia e decrittografia. La politica di impegno impostata si applica a tutte le operazioni di crittografia e decrittografia eseguite in quella sessione.

`aws_cryptosdk_session_new_from_cmm` e `aws_cryptosdk_session_new_from_keyring` sono obsolete nella versione 1.7. x e rimosse nella versione 2.0. x. Queste funzioni vengono sostituite da `aws_cryptosdk_session_new_from_cmm_2` e `aws_cryptosdk_session_new_from_keyring_2` che restituiscono una sessione.

Quando si utilizza `aws_cryptosdk_session_new_from_keyring_2` o `aws_cryptosdk_session_new_from_cmm_2` al più recente 1. x versioni, è necessario chiamare `aws_cryptosdk_session_set_commitment_policy` funzione con il valore della politica di `COMMITMENT_POLICY_FORBID_ENCRYPT_ALLOW_DECRYPT` impegno. Nelle versioni 2.0. x e versioni successive, la chiamata a questa funzione è facoltativa e accetta tutti i valori validi. La politica di impegno predefinita per le versioni 2.0. x e dopo lo è `COMMITMENT_POLICY_REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Per un esempio completo, vedi [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

Il `require-encrypt-require-decrypt` valore è la politica di impegno predefinita in tutte le versioni AWS Encryption SDK di for.NET. È possibile impostarla in modo esplicito come

best practice, ma non sia necessario. Tuttavia, se si utilizza for.NETAWS Encryption SDK per decrittografare il testo cifrato con un'implementazione in un'altra lingua dell'impegnoAWS Encryption SDK without key, è necessario modificare il valore della politica di impegno in `REQUIRE_ENCRYPT_ALLOW_DECRYPT` or `FORBID_ENCRYPT_ALLOW_DECRYPT`. In caso contrario, i tentativi di decrittografare il testo cifrato hanno esito negativo.

InAWS Encryption SDK for.NET, si imposta la politica di impegno su un'istanza diAWS Encryption SDK. Crea un'istanza di un `AwsEncryptionSdkConfig` oggetto con un `CommitmentPolicy` parametro e usa l'oggetto di configurazione per creare l'AWS Encryption SDKistanza. Quindi, chiama `Decrypt()` metodi `Encrypt()` and dell'AWS Encryption SDKistanza configurata.

Questo esempio definisce la politica di impegno per `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

Per impostare una politica di impegno nella CLI di AWS crittografia, utilizza il `--commitment-policy` parametro. Questo parametro è stato introdotto nella versione 1.8. x.

Al più recente 1. versione x, quando si utilizza il `--wrapping-keys` parametro in un `--decrypt` comando `--encrypt` or, è necessario un `--commitment-policy` parametro con il `forbid-encrypt-allow-decrypt` valore. Altrimenti, il `--commitment-policy` parametro non è valido.

Nelle versioni 2.1. x e versioni successive, il `--commitment-policy` parametro è facoltativo e il valore predefinito è il `require-encrypt-require-decrypt` valore, che non crittografa o decrittograferà alcun testo cifrato senza l'impegno della chiave. Si consiglia tuttavia di impostare la politica degli impegni in modo esplicito in tutte le chiamate di cifrare e decrittografare.

Questo esempio definisce la politica di impegno. Utilizza anche il `--wrapping-keys` parametro che sostituisce il `--master-keys` parametro a partire dalla versione 1.8. x. Per informazioni dettagliate, consultare [the section called “Aggiornamento dei provider di chiavi AWS KMS principali”](#). Per esempi completi, vedere [Esempi di AWS CLI di crittografia](#).

```

\\ 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 partire dalla versione 1.7. x del SDK di crittografia AWS per Java, hai impostato la politica di impegno sulla tua istanza dell'AwsCrypto oggetto che rappresenta il AWS Encryption SDK cliente. Questa impostazione della politica di impegno si applica a tutte le operazioni di crittografia e decrittografia eseguite su quel client.

Il `AwsCrypto()` costruttore è obsoleto al più recente 1. x della SDK di crittografia AWS per Java e vengono rimosse nella versione 2.0. x. Viene sostituito da una nuova `Builder` classe, un `Builder.withCommitmentPolicy()` metodo e il tipo `CommitmentPolicy` enumerato.

Al più recente 1. x versioni, la `Builder` classe richiede il `Builder.withCommitmentPolicy()` metodo e l'`CommitmentPolicy.ForbidEncryptAllowDecrypt` argomento. A partire dalla versione 2.0. x, il `Builder.withCommitmentPolicy()` metodo è facoltativo; il valore predefinito è `CommitmentPolicy.RequireEncryptRequireDecrypt`.

Per un esempio completo, [SetCommitmentPolicyExamplevedi.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 partire dalla versione 1.7. x di essiSDK di crittografia AWS per JavaScript, puoi impostare la politica di impegno quando chiami la `new buildClient` funzione che istanzia un AWS Encryption SDK client. `buildClient` funzione assume un valore enumerato che rappresenta la tua politica di impegno. Restituisce `encrypt` e `decrypt` funziona in modo da far rispettare la politica di impegno durante la crittografia e la decrittografia.

Al più recente 1. x versioni, `buildClient` funzione richiede l'`CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` argomento. A partire dalla versione 2.0. x, l'argomento della politica di impegno è facoltativo e il valore predefinito è `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Il codice per Node.js e il browser sono identici per questo scopo, tranne per il fatto che il browser necessita di una dichiarazione per impostare le credenziali.

L'esempio seguente crittografa i dati con un AWS KMS portachiavi. La `new buildClient` funzione imposta la politica di impegno su `FORBID_ENCRYPT_ALLOW_DECRYPT`, il valore predefinito più recente 1. 2 versioni. Gli aggiornamenti `encrypt` e `decrypt` funzioni che `buildClient` restituisci applicano la politica di impegno che hai impostato.

```
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 partire dalla versione 1.7. x del SDK di crittografia AWS per Python, hai impostato la politica di impegno sulla tua istanza di `EncryptionSDKClient`, un nuovo oggetto che rappresenta il AWS Encryption SDK cliente. La politica di impegno impostata si applica a tutte le chiamate `encrypt` e `decrypt` che utilizzano quell'istanza del client.

Al più recente 1. x versioni, il costruttore `EncryptionSDKClient` richiede il valore `CommitmentPolicy.FORBID_ENCRYPT_ALLOW_DECRYPT` enumerato. A partire dalla versione 2.0. x, l'argomento della politica di impegno è facoltativo e il valore predefinito è `CommitmentPolicy.REQUIRE_ENCRYPT_REQUIRE_DECRYPT`.

Questo esempio utilizza il nuovo costruttore `EncryptionSDKClient` e imposta la politica di impegno sulla 1.7. x valore predefinito. Il costruttore crea un client che rappresenta il AWS Encryption SDK. Quando chiami i `encrypt` o `decrypt` su questo client, questi applicano la politica di impegno che hai impostato. Questo esempio utilizza anche il nuovo costruttore per la `StrictAwsKmsMasterKeyProvider` classe, che specifica AWS KMS keys quando crittografare e decrittografare.

Per un esempio completo, vedere [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
)
```

## Risoluzione dei problemi relativi alla migrazione alle versioni più recenti

Prima di aggiornare l'applicazione alla versione 2.0.xo più tardi delAWS Encryption SDK, aggiornamento alla versione più recente 1.xversione delAWS Encryption SDKe distribuiscilo completamente. Questo ti aiuterà a evitare la maggior parte degli errori che potresti riscontrare durante l'aggiornamento alle versioni 2.0.xe dopo. Per una guida dettagliata, inclusi esempi, vedere[Migrazione della tuaAWS Encryption SDK](#).

### Important

Verifica che il tuo ultimo 1.xla versione è 1.7.xo più tardi delAWS Encryption SDK.

### Note

AWSCriptazione CLI: Riferimenti di questa guida alla versione 1.7.xdelAWS Encryption SDKsi applica alla versione 1.8.xdelAWSCrittografia CLI. Riferimenti di questa guida alla versione 2.0.xdelAWS Encryption SDKsi applica al 2.1.xdelAWSCrittografia CLI. Le nuove funzionalità di sicurezza sono state inizialmente rilasciate inAWSEncryption CLI versioni 1.7.xe 2.0.x. Tuttavia,AWSCrittografia CLI versione 1.8.xsostituisce la versione 1.7.xeAWSCrittografia CLI 2.1.xsostituisce 2.0.x. Per informazioni dettagliate, consultare la sezione pertinente[consulenza sulla sicurezzanelaws-encryption-sdk-clirepository](#) su GitHub.

Questo argomento è stato progettato per aiutarti a riconoscere e risolvere gli errori più comuni che potresti riscontrare.

## Argomenti

- [Oggetti obsoleti o rimossi](#)
- [Conflitto di configurazione Politica di impegno e suite di algoritmi](#)
- [Conflitto di configurazione Politica di impegno e testo cifrato](#)
- [Convalida dell'impegno chiave non riuscita](#)
- [Altri errori di crittografia](#)
- [Altri errori di decrittografia](#)
- [Considerazioni sul rollback](#)

## Oggetti obsoleti o rimossi

Versione 2.0.x include diverse modifiche importanti, inclusa la rimozione di costruttori, metodi, funzioni e classi obsoleti che erano obsoleti nella versione 1.7.x. Per evitare errori del compilatore, errori di importazione, errori di sintassi e errori relativi ai simboli non trovati (a seconda del linguaggio di programmazione in uso), esegui prima l'aggiornamento alla versione più recente 1.x versione del AWS Encryption SDK per il tuo linguaggio di programmazione. (Deve essere la versione 1.7.x o più tardi.) Durante l'utilizzo della versione più recente 1.x versione, puoi iniziare a utilizzare gli elementi sostitutivi prima che i simboli originali vengano rimossi.

Se è necessario eseguire l'aggiornamento alla versione 2.0.x subito dopo, [consulta il changelog](#) per il tuo linguaggio di programmazione e sostituisci i simboli precedenti con i simboli consigliati dal changelog.

## Conflitto di configurazione Politica di impegno e suite di algoritmi

Se specifichi una suite di algoritmi che è in conflitto con [politica di impegno](#), la chiamata alla crittografia non riesce e restituisce unconfitto di configurazione `Errore`.

Per evitare questo tipo di errore, non specificate una suite di algoritmi. Per impostazione predefinita, il AWS Encryption SDK sceglie l'algoritmo più sicuro compatibile con la tua politica di impegno. Tuttavia, se devi specificare una suite di algoritmi, ad esempio una suite senza firma, assicurati di scegliere una suite di algoritmi compatibile con la tua politica di impegno.

Politica di impegno	Suite di algoritmi compatibili
ForbidEncryptAllowDecrypt	<p>Qualsiasi suite di algoritmi senza impegno chiave, come ad esempio:</p> <p>AES_256_GCM_IV12_TAG16_HKDF_SHA384_ECDSA_P384 (<a href="#">03 78</a>) (con firma)</p> <p>AES_256_GCM_IV12_TAG16_HKDF_SHA256 (<a href="#">01 78</a>) (senza firmare)</p>
RequireEncryptAllowDecrypt RequireEncryptRequireDecrypt	<p>Qualsiasi suite di algoritmi con impegno chiave, come ad esempio:</p> <p>AES_256_GCM_HKDF_SHA512_COM_MIT_KEY_ECDSA_P384 (<a href="#">05 78</a>) (con firma)</p> <p>AES_256_GCM_HKDF_SHA512_COM_MIT_KEY (<a href="#">04 78</a>) (senza firmare)</p>

Se si verifica questo errore quando non è stata specificata una suite di algoritmi, la suite di algoritmi in conflitto potrebbe essere stata scelta dal [gestore di materiali crittografici](#) (CM). La CMM predefinita non selezionerà una suite di algoritmi in conflitto, ma una CMM personalizzata sì. Per assistenza, consulta la documentazione della tua CMM personalizzata.

## Conflitto di configurazione Politica di impegno e testo cifrato

Il `RequireEncryptRequireDecrypt` [politica di impegno](#) non consente il `AWS Encryption SDK` per decifrare un messaggio crittografato senza [Scadenza impegno](#). Se chiedi al `AWS Encryption SDK` per decifrare un messaggio senza impegno chiave, restituisce un conflitto di configurazione `Errore`.

Per evitare questo errore, prima di impostare il `RequireEncryptRequireDecrypt` politica di impegno, assicurati che tutti i testi cifrati crittografati senza impegno chiave vengano decrittografati e ricrittografati con l'impegno della chiave o gestiti da un'altra applicazione. Se si verifica questo errore, è possibile restituire un errore relativo al testo cifrato in conflitto o modificare temporaneamente la politica di impegno a `RequireEncryptAllowDecrypt`.

Se riscontri questo errore perché hai effettuato l'aggiornamento alla versione 2.0.xo versione successiva da una versione precedente alla 1.7.x senza prima effettuare l'aggiornamento alla versione più recente 1.x versione 1.7.xo più tardi), considera [rollback](#) alla più recente 1.x versione e

distribuzione di tale versione su tutti gli host prima dell'aggiornamento alla versione 2.0.xo più tardi. Per assistenza, consulta [Come migrare e implementare ilAWS Encryption SDK](#).

## Convalida dell'impegno chiave non riuscita

Quando decifri i messaggi crittografati con l'impegno della chiave, potresti ottenere unConvalida dell'impegno chiave non riuscita messaggio di errore. Ciò indica che la chiamata di decrittografia non è riuscita a causa di una chiave di dati in un [messaggio criptato](#) non è identica alla chiave dati univoca del messaggio. Convalidando la chiave dei dati durante la decrittografia, [Scadenza impegno](#) ti protegge dalla decrittografia di un messaggio che potrebbe generare più di un testo in chiaro.

Questo errore indica che il messaggio crittografato che stavi cercando di decrittografare non è stato restituito dalAWS Encryption SDK. Potrebbe essere un messaggio creato manualmente o il risultato di un danneggiamento dei dati. Se si verifica questo errore, l'applicazione può rifiutare il messaggio e continuare o interrompere l'elaborazione di nuovi messaggi.

## Altri errori di crittografia

La crittografia può fallire per diversi motivi. Non puoi utilizzare un [AWS KMSportachiavi Discovery](#) o un [provider di chiavi master in modalità discovery](#) per crittografare un messaggio.

Assicurati di specificare un portachiavi o un fornitore di chiavi master con le chiavi di avvolgimento che possiedi [autorizzazione all'uso](#) per la crittografia. Per assistenza con le autorizzazioni suAWS KMS keys, vedi [Visualizzazione di una policy di chiave](#) e [Determinazione dell'accesso a unaAWS KMS key](#) nelAWS Key Management ServiceGuida per gli sviluppatori.

## Altri errori di decrittografia

Se il tentativo di decifrare un messaggio crittografato fallisce, significa cheAWS Encryption SDKnon ha potuto (o non vuole) decrittografare nessuna delle chiavi di dati crittografate nel messaggio.

Se hai usato un portachiavi o un provider di chiavi master che specifica le chiavi di impacchettamento,AWS Encryption SDKutilizza solo le chiavi di avvolgimento specificate. Verificare di utilizzare le chiavi di imballaggio che si intende utilizzare e di disporre `kms:Decrypt` autorizzazione su almeno una delle chiavi di imballaggio. Se stai utilizzandoAWS KMS keys, come riserva, puoi provare a decrittografare il messaggio con un [AWS KMSportachiavi Discovery](#) o un [provider di chiavi master in modalità discovery](#). Se l'operazione ha esito positivo, prima di restituire il testo in chiaro, verifica che la chiave utilizzata per decrittografare il messaggio sia attendibile.

## Considerazioni sul rollback

Se l'applicazione non riesce a crittografare o decrittografare i dati, in genere è possibile risolvere il problema aggiornando i simboli del codice, i portachiavi, i fornitori di chiavi master o [politica di impegno](#). Tuttavia, in alcuni casi, potresti decidere che è meglio ripristinare l'applicazione a una versione precedente di AWS Encryption SDK.

Se è necessario eseguire il rollback, farlo con cautela. Versioni di AWS Encryption SDK precedenti alla 1.7.x non può decifrare il testo cifrato crittografato con [Scadenza impegno](#).

- Tornando indietro rispetto all'ultima versione di una versione precedente di AWS Encryption SDK è generalmente sicuro. Potrebbe essere necessario annullare le modifiche apportate al codice per utilizzare simboli e oggetti non supportati nelle versioni precedenti.
- Dopo aver iniziato la crittografia con l'impegno chiave (impostazione della politica di impegno su `RequireEncryptAllowDecrypt`) nella versione 2.0.x o versione successiva, è possibile ripristinare la versione 1.7.x, ma non in nessuna versione precedente. Versioni di AWS Encryption SDK precedenti alla 1.7.x non può decifrare il testo cifrato crittografato con [Scadenza impegno](#).

Se abiliti accidentalmente la crittografia con l'impegno delle chiavi prima che tutti gli host possano decifrarla con l'impegno della chiave, potrebbe essere meglio continuare con l'implementazione anziché eseguire il rollback. Se i messaggi sono transitori o possono essere eliminati in modo sicuro, potresti prendere in considerazione un rollback con perdita di messaggi. Se è necessario un rollback, potresti prendere in considerazione la possibilità di scrivere uno strumento che decifri e cripti nuovamente tutti i messaggi.

## Domande frequenti

- [In che modo il AWS Encryption SDK è diverso dagli SDK AWS?](#)
- [Com'èAWS Encryption SDKdiverso dal client di crittografia Amazon S3?](#)
- [Quali algoritmi di crittografia sono supportati dal AWS Encryption SDK e qual è l'impostazione predefinita?](#)
- [In che modo viene generato il vettore di inizializzazione \(IV\) e dove viene memorizzato?](#)
- [Come viene generata, crittografata e decrittografata ciascuna chiave di dati?](#)
- [Come posso tenere traccia dei dati delle chiavi utilizzati per crittografare i miei dati?](#)
- [In che modo AWS Encryption SDK memorizza le chiavi di dati crittografati con i dati crittografati?](#)
- [Quale sovraccarico aggiunge il formato di messaggi del AWS Encryption SDK ai miei dati crittografati?](#)
- [È possibile usare il proprio provider di chiavi master?](#)
- [È possibile crittografare i dati in più di una chiave di wrapping?](#)
- [Quali tipi di dati è possibile crittografare con il AWS Encryption SDK?](#)
- [In che modo il AWS Encryption SDK crittografa e decrittografa i flussi di input/output \(I/O\)?](#)

In che modo il AWS Encryption SDK è diverso dagli SDK AWS?

La [AWS SDK](#) fornisce librerie per l'interazione con Amazon Web Services (AWS), incluso AWS Key Management Service (AWS KMS). Alcune delle implementazioni linguistiche del AWS Encryption SDK, come ad esempio il [AWS Encryption SDK per .NET](#), richiede sempre il [AWS SDK](#) nello stesso linguaggio di programmazione. Le implementazioni di altre lingue richiedono il corrispondente [AWS SDK](#) solo quando si utilizza AWS KMS chiavi nei tuoi portachiavi o master key provider. Per ulteriori informazioni, consulta l'argomento relativo al tuo linguaggio di programmazione in [Linguaggi di programmazione AWS Encryption SDK](#).

Puoi utilizzare il plugin [AWS SDK](#) con cui interagire AWS KMS, inclusa la crittografia e la decrittografia di piccole quantità di dati (fino a 4.096 byte con una chiave di crittografia simmetrica) e la generazione di chiavi dati per la crittografia lato client. Tuttavia, quando si genera una chiave dati, è necessario gestire l'intero processo di crittografia e decrittografia, inclusa la crittografia dei dati con la chiave dati esterna a AWS KMS, scartando in modo sicuro la chiave dati in testo chiaro, memorizzando la chiave dati crittografata e quindi decrittografando la chiave dati e decrittografando i tuoi dati. La [AWS Encryption SDK](#) gestisce questo processo per te.

LaAWS Encryption SDKfornisce una libreria che crittografa e decrittazione dei dati utilizzando standard e best practice del settore. Genera la chiave dati, la crittografa sotto le chiavi di wrapping specificate e restituisce un messaggio crittografato un oggetto di dati portatile che include i dati crittografati e le chiavi di dati crittografate necessarie per decrittare. Quando è il momento di decrittografare, si passa il messaggio crittografato e almeno una delle chiavi di wrapping (facoltative) eAWS Encryption SDKrestituisce i dati di testo normale.

È possibile utilizzareAWS KMS keyscome avvolgimento di chiavi nelAWS Encryption SDKma non è obbligatorio. È possibile utilizzare le chiavi di crittografia generate e quelle dal gestore di chiavi o dal modulo di sicurezza hardware locale. Puoi utilizzare il pluginAWS Encryption SDKanche se non disponi di unAWSconto.

Com'èAWS Encryption SDKdiverso dal client di crittografia Amazon S3?

La[Client di crittografia Amazon S3](#)nellaAWSGli SDK di forniscono crittografia e decrittografia per i dati archiviati in Amazon Simple Storage Service (Amazon S3). Questi client sono strettamente accoppiati per Amazon S3 e sono destinati a essere utilizzati solo con i dati ivi memorizzati.

Il AWS Encryption SDK fornisce la crittografia e la decrittazione per i dati che puoi memorizzare in qualsiasi luogo. LaAWS Encryption SDKe il client di crittografia Amazon S3 non sono compatibili perché producono testi cifrati con diversi formati di dati.

Quali algoritmi di crittografia sono supportati dal AWS Encryption SDK e qual è l'impostazione predefinita?

LaAWS Encryption SDKutilizza l'algoritmo simmetrico Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM), noto come AES-GCM, per crittografare i dati. Consente di scegliere tra diversi algoritmi simmetrici e asimmetrici per crittografare le chiavi dati che crittografano i dati.

Per AES-GCM, la suite di algoritmi predefinita è AES-GCM con una chiave a 256 bit, derivazione della chiave (HKDF),[firme digitali](#), [impegno chiave](#).AWS Encryption SDKsupporta inoltre chiavi di crittografia a 192 bit e 128 bit e algoritmi di crittografia senza firme digitali e impegno chiave.

In tutti i casi, la lunghezza del vettore di inizializzazione (IV) è pari a 12 byte; la lunghezza del tag di autenticazione è pari a 16 byte. Per impostazione predefinita, il kit SDK utilizza la chiave di dati come input per l' HMAC extract-and-expand funzione di derivazione della chiave (HKDF) per ottenere la chiave di crittografia AES-GCM e inoltre aggiungere una firma ECDSA (Elliptic Curve Digital Signature Algorithm).

Per ulteriori informazioni sulla scelta degli algoritmi da utilizzare, consulta [Suite di algoritmi supportate](#).

Per dettagli relativi all'implementazione degli algoritmi supportati, consulta [Riferimenti agli algoritmi](#).

In che modo viene generato il vettore di inizializzazione (IV) e dove viene memorizzato?

LaAWS Encryption SDKutilizza un metodo deterministico per creare un valore IV diverso per ciascun frame. Questa procedura garantisce che le IVs non vengano mai ripetute all'interno di un messaggio. (Prima della versione 1.3.0 delSDK di crittografia AWS per Javae laSDK di crittografia AWS per Python, ilAWS Encryption SDKgenerava casualmente un valore IV univoco per ciascun frame.)

Il IV è memorizzato nel messaggio crittografatoAWS Encryption SDKrestituisce. Per ulteriori informazioni, consultare la [Riferimenti a formati di messaggi AWS Encryption SDK](#).

Come viene generata, crittografata e decrittografata ciascuna chiave di dati?

Il metodo dipende dal keyring o dal fornitore di chiavi master utilizzate.

LaAWS KMSportachiavi e master key provider nelAWS Encryption SDKusa il linkAWS KMS [GenerateDataKey](#)Operazione API per generare ogni chiave dati e crittografarla sotto la sua chiave di wrapping. Per crittografare le copie della chiave dati con chiavi KMS aggiuntive, utilizzano ilAWS KMS [Encrypt](#)operazione. Per decrittografare le chiavi dati, utilizzano ilAWS KMS [Decrypt](#)operazione. Per informazioni dettagliate, consulta [.AWS KMSKeyring](#)nellaAWS Encryption SDKSpecifiche in GitHub.

Altri portachiavi generano la chiave dati, crittografano e decifrano utilizzando metodi di best practice per ciascun linguaggio di programmazione. Per ulteriori informazioni, vedere le specifiche del portachiavi o del fornitore di chiavi master nella[Sezione del framework](#)delAWS Encryption SDKSpecifiche in GitHub.

Come posso tenere traccia dei dati delle chiavi utilizzati per crittografare i miei dati?

AWS Encryption SDK lo fa per te Quando crittografi i dati, il kit SDK crittografa la chiave dei dati e archivia la chiave crittografata con i dati crittografati nel [messaggio crittografato](#) che restituisce. Quando esegui la decrittazione dei dati, il AWS Encryption SDK estrae la chiave di dati crittografata dal messaggio crittografato, la decrittata, quindi la utilizza per decrittare i dati.

In che modo AWS Encryption SDK memorizza le chiavi di dati crittografati con i dati crittografati?

Le operazioni di crittografia in AWS Encryption SDK restituiscono un [messaggio crittografato](#), una struttura di dati singola che contiene i dati crittografati e le chiavi dei dati crittografati. Il formato del messaggio è costituito da almeno due parti: un'intestazione e un corpo. L'intestazione del messaggio contiene le chiavi di dati crittografati e le informazioni su come è formato il corpo del

messaggio. Il corpo del messaggio contiene i dati crittografati. Se la suite di algoritmi include un [firma digitale](#) il formato del messaggio include un piè di pagina che contiene la firma. Per ulteriori informazioni, consulta la pagina [Riferimenti a formati di messaggi AWS Encryption SDK](#).

Quale sovraccarico aggiunge il formato di messaggi del AWS Encryption SDK ai miei dati crittografati?

La quantità di sovraccarico aggiunta dal AWS Encryption SDK dipende da diversi fattori, tra cui:

- La dimensione dei dati di testo non crittografato
- Quale degli algoritmi supportati viene utilizzato
- Se vengono forniti dati autenticati aggiuntivi (AAD) e la lunghezza di tale AAD
- Il numero e tipo di chiavi di wrapping o chiavi master
- Le dimensioni del frame (quando vengono utilizzati i [dati framed](#))

Quando utilizzi il file AWS Encryption SDK con la sua configurazione predefinita (una AWS KMS key in quanto chiave di wrapping (o chiave master), nessun AAD, dati non framed e algoritmi di crittografia con firma), il sovraccarico è di circa 600 byte. In generale, puoi ragionevolmente presumere che il AWS Encryption SDK aggiunga un sovraccarico di 1 KB al massimo, senza includere gli AAD forniti. Per ulteriori informazioni, consulta la pagina [Riferimenti a formati di messaggi AWS Encryption SDK](#).

È possibile usare il proprio provider di chiavi master?

Sì. I dettagli di implementazione variano a seconda di quale dei [linguaggi di programmazione supportati](#) utilizzi. Tuttavia, tutte le lingue supportate consentono di definire personalizzati [gestori di materiali crittografici \(CMM\)](#), master key provider, portachiavi, chiavi master e chiavi di avvolgimento.

È possibile crittografare i dati in più di una chiave di wrapping?

Sì. È possibile crittografare la chiave di dati con le aggiuntive chiavi di wrapping (o chiavi master) per aggiungere ridondanza quando la chiave è in un'altra regione o non è disponibile per la decrittografia.

Per crittografare i dati in più chiavi di wrapping, crea un keyring o fornitore di chiavi master con più chiavi di wrapping. Quando utilizzi i keyring puoi creare un [singolo keyring con più chiavi di wrapping](#) o un [keyring multiplo](#).

Quando si crittografano i dati con più chiavi di wrapping, AWS Encryption SDK utilizza un tasto wrapping per generare una chiave dati in testo normale. La chiave dati è unica e

matematicamente estranea alla chiave di avvolgimento. L'operazione restituisce la chiave di dati di testo normale e una copia della chiave di dati crittografata dalla chiave di wrapping. Quindi, il metodo di crittografia, crittografa la chiave di dati con le altre chiavi di wrapping. Il risultato [messaggio crittografato](#) include i dati crittografati e una chiave di dati crittografati per ciascuna chiave di wrapping.

Il messaggio crittografato può essere decrittografato utilizzando una qualsiasi delle chiavi di wrapping utilizzate nell'operazione di crittografia. La AWS Encryption SDK utilizza una chiave di wrapping per decodificare una chiave di dati crittografati. Quindi, utilizza la chiave di dati di testo normale per decrittare i dati.

Quali tipi di dati è possibile crittografare con il AWS Encryption SDK?

La maggior parte delle implementazioni del linguaggio di programmazione AWS Encryption SDK può crittografare byte non elaborati (array di byte), flussi I/O (flussi di byte) e stringhe. La AWS Encryption SDK per .NET non supporta flussi I/O. Forniamo il codice di esempio per ciascuno dei [linguaggi di programmazione supportati](#).

In che modo il AWS Encryption SDK crittografa e decrittifica i flussi di input/output (I/O)?

AWS Encryption SDK crea un flusso di crittografia o decrittazione che racchiude un flusso di I/O sottostante. Il flusso di crittografia e decrittografia esegue un'operazione di crittografia su una chiamata di lettura o di scrittura. Ad esempio, può leggere dati di testo non crittografati nel flusso sottostante e crittografarlo prima di restituire il risultato. In alternativa, può leggere testo cifrato da un flusso sottostante e decifrarlo prima di restituire il risultato. Forniamo il codice di esempio per crittografare e decrittografare flussi per ciascuno dei [linguaggi di programmazione supportati](#) che supporta lo streaming.

La AWS Encryption SDK per .NET non supporta flussi I/O.

# Riferimento AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

Per le specifiche che definiscono gli elementi di una corretta AWS Encryption SDK implementazione, vedere la [AWS Encryption SDK Specificazione](#) in GitHub.

AWS Encryption SDK utilizza [algoritmi supportati](#) per restituire una struttura di dati singola o un messaggio che contiene dati crittografati e chiavi dei dati crittografati corrispondenti. I seguenti argomenti spiegano gli algoritmi e la struttura dei dati. Utilizza queste informazioni per creare librerie che sono in grado di leggere e scrivere testi cifrati compatibili con questo SDK.

## Argomenti

- [Riferimenti a formati di messaggi AWS Encryption SDK](#)
- [Esempi di formati di messaggi AWS Encryption SDK](#)
- [Riferimento ai dati autenticati aggiuntivi \(AAD\) del corpo per AWS Encryption SDK](#)
- [Riferimenti agli algoritmi AWS Encryption SDK](#)
- [Riferimento al vettore di inizializzazione AWS Encryption SDK](#)
- [AWS KMS Dettagli tecnici del portachiavi gerarchico](#)

## Riferimenti a formati di messaggi AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

Per le specifiche che definiscono gli elementi di una corretta AWS Encryption SDK implementazione, vedere la [AWS Encryption SDK Specificazione](#) in GitHub.

Le operazioni di crittografia incluse AWS Encryption SDK restituiscono una singola struttura di dati o un [messaggio crittografato](#) che contiene i dati crittografati (testo cifrato) e tutte le chiavi di dati crittografate. Per comprendere questa struttura di dati, oppure per creare librerie per le operazioni di lettura e scrittura, hai bisogno di comprendere il formato del messaggio.

Il formato del messaggio è costituito da almeno due parti: un'intestazione e un corpo. In alcuni casi, il formato del messaggio è composto da una terza parte nota come piè di pagina. Il formato del messaggio definisce una sequenza ordinata di byte nell'ordine dei byte di rete, definito anche formato big-endian. Il formato del messaggio inizia con l'intestazione, seguita dal corpo, seguita dal piè di pagina (quando disponibile).

Le [suite di algoritmi](#) supportate da AWS Encryption SDK utilizzano una delle due versioni di formato di messaggio. Le suite di algoritmi senza [impegno chiave](#) utilizzano il formato dei messaggi versione 1. Le suite di algoritmi con impegno chiave utilizzano il formato dei messaggi versione 2.

## Argomenti

- [Struttura dell'intestazione](#)
- [Struttura corpo](#)
- [Struttura piè di pagina](#)

## Struttura dell'intestazione

L'intestazione del messaggio contiene la chiave di dati crittografati e le informazioni su come è formato il corpo del messaggio. La tabella seguente descrive i campi che costituiscono l'intestazione nelle versioni 1 e 2 dei formati di messaggio. I byte vengono aggiunti nell'ordine mostrato.

Il valore Non presente indica che il campo non esiste in quella versione del formato del messaggio. Il testo in grassetto indica valori diversi in ogni versione.

### Note

Potrebbe essere necessario scorrere orizzontalmente o verticalmente per visualizzare tutti i dati di questa tabella.

## Struttura dell'intestazione

Campo	Formato del messaggio (versione 1)	Formato dei messaggi versione 2
	Lunghezza (byte)	Lunghezza (byte)
<a href="#">Versione</a>	1	1
<a href="#">Type</a>	1	Non presente
<a href="#">ID algoritmo</a>	2	2
<a href="#">ID messaggio</a>	16	32
<a href="#">Lunghezza AAD</a>	2	2
	Quando il <a href="#">contesto di crittografia</a> è vuoto, il valore del campo Lunghezza AAD a 2 byte è 0.	Quando il <a href="#">contesto di crittografia</a> è vuoto, il valore del campo Lunghezza AAD a 2 byte è 0.
<a href="#">AAD</a>	Variabile. La lunghezza di questo campo viene visualizzata nei 2 byte precedenti (campo Lunghezza AAD).  Quando il <a href="#">contesto di crittografia</a> è vuoto, non vi è alcun campo AAD nell'intestazione.	Variabile. La lunghezza di questo campo viene visualizzata nei 2 byte precedenti (campo AAD Length).  Quando il <a href="#">contesto di crittografia</a> è vuoto, non vi è alcun campo AAD nell'intestazione.
<a href="#">Conteggio chiave dati crittografati</a>	2	2
<a href="#">Chiavi di dati crittografati</a>	Variabile. Determinato dal numero di chiavi di dati crittografati e dalla lunghezza di ciascuna.	Variabile. Determinato dal numero di chiavi di dati crittografati e dalla lunghezza di ciascuna.
<a href="#">Content Type</a>	1	1
<a href="#">Riservata</a>	4	Non presente

Campo	Formato del messaggio (versione 1)	Formato dei messaggi versione 2
	Lunghezza (byte)	Lunghezza (byte)
<a href="#">Lunghezza IV</a>	1	Non presente
<a href="#">Lunghezza del frame</a>	4	4
<a href="#">Algorithm Suite Data</a>	Non presente	Variabile. Determinato dall' <a href="#">algoritmo</a> che ha generato il messaggio.
<a href="#">Autenticazione dell'intestazione</a>	Variabile. Determinato dall' <a href="#">algoritmo</a> che ha generato il messaggio.	Variabile. Determinato dall' <a href="#">algoritmo</a> che ha generato il messaggio.

## Versione

La versione di questo formato di messaggio. La versione è codificata 1 o 2 come byte 01 o 02 in notazione esadecimale

## Type

Il tipo di questo formato di messaggio. Il tipo indica il tipo di struttura. L'unico tipo supportato viene descritto come dati crittografati autenticati dal cliente. Il valore del tipo è 128, codificato come byte 80 in notazione esadecimale.

Questo campo non è presente nella versione 2 del formato di messaggio.

## ID algoritmo

Identificativo dell'algoritmo utilizzato. È un valore a 2 byte interpretato come un numero intero senza segno a 16 bit. Per ulteriori informazioni sugli algoritmi, consulta [Riferimenti agli algoritmi AWS Encryption SDK](#).

## ID messaggio

Un valore generato casualmente che identifica il messaggio. L'ID del messaggio:

- Identifica in modo univoco il messaggio crittografato.
- Associa debolmente l'intestazione del messaggio al corpo del messaggio.

- Fornisce un meccanismo per riutilizzare una chiave di dati in modo sicuro con più messaggi crittografati.
- Protegge da un riutilizzo accidentale della chiave di dati o dall'esaurimento delle chiavi nella AWS Encryption SDK.

Questo valore è 128 bit nella versione 1 del formato di messaggio e 256 bit nella versione 2.

## Lunghezza AAD

La durata dei dati autenticati aggiuntivi (AAD). Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono l'AAD.

Quando il [contesto di crittografia](#) è vuoto, il valore del campo AAD Length è 0.

## AAD

Dati autenticati aggiuntivi. L'AAD è una codifica del [contesto di crittografia](#), un array di coppie chiave-valore in cui ciascuna chiave e il valore sono una stringa di caratteri con codifica UTF-8. Il contesto di crittografia viene trasformato in una sequenza di byte e utilizzato per il valore AAD. Quando il contesto di crittografia è vuoto, non vi è alcun campo AAD nell'intestazione.

Quando vengono utilizzati gli [algoritmi con firma](#), il contesto di crittografia deve contenere la coppia chiave-valore {'aws-crypto-public-key', Qtxt}. Qtxt rappresenta la curva ellittica punto Q compressa in base alla [SEC 1 versione 2.0](#) e con codifica base64. Il contesto di crittografia può contenere valori aggiuntivi, ma la durata massima dell'AAD costruito è di  $2^{16} - 1$  byte.

La tabella seguente descrive i campi che costituiscono l'AAD. Le coppie chiave-valore sono ordinate per chiave, in ordine crescente in base al codice di caratteri UTF-8. I byte vengono aggiunti nell'ordine mostrato.

## Struttura AAD

Campo	Lunghezza (byte)
<a href="#">Conteggio coppia chiave-valore</a>	2
<a href="#">Lunghezza chiave</a>	2
<a href="#">Chiave</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza chiave).

Campo	Lunghezza (byte)
<a href="#">Lunghezza valore</a>	2
<a href="#">Valore</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza valore).

### Conteggio coppia chiave-valore

Il numero di coppie chiave-valore nell'AAD. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di coppie chiave-valore nell'AAD. Il numero massimo di coppie chiave-valore nell'AAD è di  $2^{16} - 1$ .

Quando non vi è alcun contesto di crittografia o il contesto di crittografia è vuoto, questo campo non è presente nella struttura AAD.

### Lunghezza chiave

La lunghezza della chiave per la coppia chiave-valore. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono la chiave.

### Chiave

La chiave per la coppia chiave-valore. Si tratta di una sequenza di byte con codifica UTF-8.

### Lunghezza valore

La lunghezza del valore per la coppia chiave-valore. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono il valore.

### Valore

Il valore per la coppia chiave-valore. Si tratta di una sequenza di byte con codifica UTF-8.

### Conteggio chiave dati crittografati

Il numero di chiavi di dati crittografati. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di chiavi di dati crittografati. Il numero massimo di chiavi dati crittografate in ogni messaggio è  $65.535 (2^{16} - 1)$ .

## Chiavi di dati crittografati

Sequenza di chiavi di dati crittografati. La lunghezza della sequenza è determinata dal numero di chiavi di dati crittografati e dalla lunghezza di ciascuna. La sequenza contiene almeno una chiave di dati crittografati.

La tabella seguente descrive i campi che costituiscono ogni chiave di dati crittografati. I byte vengono aggiunti nell'ordine mostrato.

### Struttura chiave dati crittografati

Campo	Lunghezza (byte)
<a href="#">Lunghezza ID provider chiave</a>	2
<a href="#">ID provider chiave</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza ID provider chiave).
<a href="#">Lunghezza informazioni provider chiave</a>	2
<a href="#">Informazioni provider chiave</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza informazione provider chiave).
<a href="#">Lunghezza chiave dati crittografati</a>	2
<a href="#">Chiave di dati crittografati</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza chiave dati crittografati).

### Lunghezza ID provider chiave

Lunghezza dell'identificatore del provider della chiave. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono l'ID del provider della chiave.

### ID provider chiave

Identificatore del provider della chiave. Viene utilizzato per indicare il provider della chiave dei dati crittografati ed è destinato a essere ampliabile.

## Lunghezza informazioni provider chiave

Lunghezza delle informazioni del provider della chiave. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono le informazioni del provider della chiave.

## Informazioni provider chiave

Informazioni provider chiave. Dipende dal provider di chiavi.

Quando AWS KMS è il fornitore della chiave principale o utilizzi un AWS KMS portachiavi, questo valore contiene l'Amazon Resource Name (ARN) di AWS KMS key.

## Lunghezza chiave dati crittografati

La lunghezza della chiave di dati crittografati. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono la chiave di dati crittografati.

## Chiave di dati crittografati

Chiave di dati crittografati. È la chiave di crittografia dei dati crittografati dal provider di chiavi.

## Content Type

Il tipo di dati crittografati, senza frame o incorniciati.

### Note

Quando possibile, utilizza dati con frame. AWS Encryption SDK supporta dati senza frame solo per uso precedente. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza frame. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza frame.

I dati con frame sono suddivisi in parti di uguale lunghezza; ogni parte viene crittografata separatamente. Il contenuto framed è di tipo 2, codificato come byte 02 in notazione esadecimale.

I dati senza frame non vengono divisi; si tratta di un singolo blob crittografato. Il contenuto non-framed è di tipo 1, codificato come byte 01 in notazione esadecimale.

## Riservata

Una sequenza riservata di 4 byte. Questo valore deve essere 0. È codificato come byte 00 00 00 00 in notazione esadecimale (una sequenza a 4 byte di un valore intero a 32 bit pari a 0).

Questo campo non è presente nella versione 2 del formato dei messaggi.

## Lunghezza IV

La lunghezza del vettore di inizializzazione (IV). Si tratta di un valore di 1 byte interpretato come un numero intero senza segno a 8 bit che specifica il numero di byte che contengono l'IV. Tale valore viene determinato dal valore dell'IV in byte dell'[algoritmo](#) che ha generato il messaggio.

Questo campo non è presente nella versione 2 del formato di messaggio, che supporta solo suite di algoritmi che utilizzano valori IV deterministici nell'intestazione del messaggio.

## Lunghezza del frame

La lunghezza di ogni frame di dati incorniciati. È un valore a 4 byte interpretato come un numero intero senza segno a 32 bit che specifica il numero di byte in ogni frame. Quando i dati non sono inclusi in frame, ovvero quando il valore del Content Type campo è 1, questo valore deve essere 0.

### Note

Quando possibile, usa dati incorniciati. AWS Encryption SDK supporta dati senza frame solo per uso precedente. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza frame. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza frame.

## Algorithm Suite Data

Dati supplementari necessari all'[algoritmo](#) che ha generato il messaggio. La lunghezza e il contenuto sono determinati dall'algoritmo. La sua lunghezza potrebbe essere 0.

Questo campo non è presente nella versione 1 del formato di messaggio.

## Autenticazione dell'intestazione

L'autenticazione dell'intestazione viene determinata dall'[algoritmo](#) che ha generato il messaggio. L'autenticazione dell'intestazione viene calcolata in base all'intera intestazione. È costituito da un IV e un tag di autenticazione. I byte vengono aggiunti nell'ordine mostrato.

## Struttura autenticazione dell'intestazione

Campo	Lunghezza nella versione 1.0 (byte)	Lunghezza nella versione 2.0 (byte)
<a href="#">IV</a>	Variabile. Determinato dal valore dell'IV in byte dell' <a href="#">algoritmo</a> che ha generato il messaggio.	N/D
<a href="#">Tag di autenticazione</a>	Variabile. Determinato dal valore in byte del tag di autenticazione dell' <a href="#">algoritmo</a> che ha generato il messaggio.	Variabile. Determinato dal valore in byte del tag di autenticazione dell' <a href="#">algoritmo</a> che ha generato il messaggio.

### IV

Il vettore di inizializzazione (IV) utilizzato per calcolare il tag di autenticazione dell'intestazione.

Questo campo non è presente nell'intestazione del formato dei messaggi versione 2. La versione 2 del formato dei messaggi supporta solo suite di algoritmi che utilizzano valori IV deterministici nell'intestazione del messaggio.

### Tag di autenticazione

Il valore di autenticazione per l'intestazione. È utilizzato per autenticare l'intero contenuto dell'intestazione.

## Struttura corpo

Il corpo del messaggio contiene i dati crittografati, denominati testo codificato. La struttura del corpo dipende dal tipo di contenuto (non-framed o framed). Le seguenti sezioni descrivono il formato del corpo del messaggio per ogni tipo di contenuto. La struttura del corpo del messaggio è la stessa nelle versioni 1 e 2 del formato dei messaggi.

### Argomenti

- [Dati non-framed](#)
- [Dati framed](#)

## Dati non-framed

I dati non-framed sono crittografati in un singolo blob con un IV univoco e l'[AAD del corpo](#).

### Note

Quando possibile, usa dati con frame. AWS Encryption SDK supporta dati senza frame solo per uso precedente. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza frame. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza frame.

La tabella seguente descrive i campi che costituiscono i dati non-framed. I byte vengono aggiunti nell'ordine mostrato.

### Struttura del corpo non-framed

Campo	Lunghezza, in byte
<a href="#">IV</a>	Variabile. Pari al valore specificato nei <a href="#">Lunghezza IV</a> byte dell'intestazione.
<a href="#">Lunghezza contenuti crittografati</a>	8
<a href="#">Contenuto crittografato</a>	Variabile. Pari al valore specificato negli 8 byte precedenti (lunghezza contenuto crittografato).
<a href="#">Tag di autenticazione</a>	Variabile. Determinato dall' <a href="#">implementazione dell'algoritmo</a> utilizzato.

### IV

Il vettore di inizializzazione (IV) da usare con l'[algoritmo di crittografia](#).

### Lunghezza contenuti crittografati

La lunghezza dei contenuti crittografati, o il testo cifrato. Si tratta di un valore di 8 byte interpretato come un numero intero senza segno a 64 bit che specifica il numero di byte che contengono il contenuto crittografato.

Tecnicamente, il valore massimo consentito è di  $2^{63} - 1$  o 8 exbibyte EiB (8). Tuttavia, in pratica, il valore massimo è  $2^{36} - 32$  o 64 gibibytes (64 GiB), a causa delle limitazioni imposte dagli [algoritmi implementati](#).

#### Note

L'implementazione Java di questo SDK limita ulteriormente questo valore a  $2^{31} - 1$  o 2 gibibyte (2 GiB), a causa di limitazioni nel linguaggio.

## Contenuto crittografato

I contenuti crittografati (testo codificato) come sono restituiti dall'[algoritmo di crittografia](#).

## Tag di autenticazione

Il valore di autenticazione per il corpo. Viene utilizzato per autenticare il corpo del messaggio.

## Dati framed

Nei dati incorniciati, i dati di testo normale sono divisi in parti di uguale lunghezza denominate frame. AWS Encryption SDK [Crittografa ogni frame separatamente con un IV e un AAD body univoci](#).

#### Note

Quando possibile, utilizza dati con frame. AWS Encryption SDK supporta dati senza frame solo per uso precedente. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza frame. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza frame.

La [lunghezza del frame](#), ovvero la lunghezza del [contenuto crittografato](#) nel frame, può essere diversa per ogni messaggio. Il numero massimo di byte in un frame è  $2^{32} - 1$ . Il numero massimo di frame in un messaggio è  $2^{32} - 1$ .

Sono disponibili due tipi di frame: periodici e finali. Ogni messaggio deve essere costituito da o includere un frame finale.

Tutti i frame normali di un messaggio hanno la stessa lunghezza del frame. Il frame finale può avere una lunghezza del frame diversa.

La composizione dei frame nei dati framed varia a seconda della lunghezza del contenuto crittografato.

- Uguale alla lunghezza del frame: quando la lunghezza del contenuto crittografato è uguale alla lunghezza del frame dei frame normali, il messaggio può essere costituito da un frame normale che contiene i dati, seguito da un frame finale di lunghezza zero (0). In alternativa, il messaggio può essere costituito solo da un frame finale contenente i dati. In questo caso, il frame finale ha la stessa lunghezza del frame normale.
- Multiplo della lunghezza del frame: quando la lunghezza del contenuto crittografato è un multiplo esatto della lunghezza del frame dei frame normali, il messaggio può terminare in un frame normale che contiene i dati, seguito da un frame finale di lunghezza zero (0). In alternativa, il messaggio può terminare in un frame finale contenente i dati. In questo caso, il frame finale ha la stessa lunghezza del frame normale.
- Non è un multiplo della lunghezza del frame: quando la lunghezza del contenuto crittografato non è un multiplo esatto della lunghezza del frame dei frame normali, il frame finale contiene i dati rimanenti. La lunghezza del frame finale è inferiore alla lunghezza del frame dei frame normali.
- Lunghezza del frame inferiore alla lunghezza del frame: quando la lunghezza del contenuto crittografato è inferiore alla lunghezza del frame normale, il messaggio è costituito da un frame finale che contiene tutti i dati. La lunghezza del frame finale è inferiore alla lunghezza del frame dei frame normali.

Le tabelle seguenti descrivono i campi che costituiscono i frame. I byte vengono aggiunti nell'ordine mostrato.

Struttura corpo framed, frame periodico

Campo	Lunghezza, in byte
<a href="#">Numero sequenza</a>	4
<a href="#">IV</a>	Variabile. Pari al valore specificato nei <a href="#">Lunghezza IV</a> byte dell'intestazione.
<a href="#">Contenuto crittografato</a>	Variabile. Pari al valore specificato nei <a href="#">Lunghezza del frame</a> dell'intestazione.

Campo	Lunghezza, in byte
<a href="#">Tag di autenticazione</a>	Variabile. Stabilito dall'algorithmo utilizzato, come specificato nel <a href="#">ID algoritmo</a> dell'intestazione.

## Numero sequenza

Il numero di sequenza del frame. Si tratta di un numero di contatori incrementale per il frame. È un valore a 4 byte interpretato come un numero intero senza segno a 32 bit.

I dati framed devono iniziare al numero di sequenza 1. I frame successivi devono essere in ordine e devono contenere un incremento di 1 del frame precedente. In caso contrario, il processo di decrittografia arresta e segnala un errore.

## IV

Il vettore di inizializzazione (IV) per il frame. L'SDK utilizza un metodo deterministico per creare un altro IV per ogni frame nel messaggio. La lunghezza viene specificata dalla [suite dell'algorithmo](#) utilizzato.

## Contenuto crittografato

I contenuti crittografati (testo codificato) per il frame come sono restituiti dall'[algoritmo di crittografia](#).

## Tag di autenticazione

Il valore di autenticazione per il frame. Viene utilizzato per autenticare il frame completo.

## Struttura corpo con frame, frame finale

Campo	Lunghezza, in byte
<a href="#">Fine numero sequenza</a>	4
<a href="#">Numero sequenza</a>	4
<a href="#">IV</a>	Variabile. Pari al valore specificato nei <a href="#">Lunghezza IV</a> byte dell'intestazione.

Campo	Lunghezza, in byte
<a href="#">Lunghezza contenuti crittografati</a>	4
<a href="#">Contenuto crittografato</a>	Variabile. Pari al valore specificato negli 4 byte precedenti (lunghezza contenuto crittografato).
<a href="#">Tag di autenticazione</a>	Variabile. Stabilito dall'algoritmo utilizzato, come specificato nel <a href="#">ID algoritmo</a> dell'interazione.

## Fine numero sequenza

Un indicatore per il frame finale. Il valore è codificato come 4 byte FF FF FF FF in notazione esadecimale.

## Numero sequenza

Il numero di sequenza del frame. Si tratta di un numero di contatori incrementale per il frame. È un valore a 4 byte interpretato come un numero intero senza segno a 32 bit.

I dati framed devono iniziare al numero di sequenza 1. I frame successivi devono essere in ordine e devono contenere un incremento di 1 del frame precedente. In caso contrario, il processo di decrittografia arresta e segnala un errore.

## IV

Il vettore di inizializzazione (IV) per il frame. L'SDK utilizza un metodo deterministico per creare un altro IV per ogni frame nel messaggio. La lunghezza del IV viene specificata dalla [suite dell'algoritmo](#).

## Lunghezza contenuti crittografati

La lunghezza del contenuto crittografato. Si tratta di un valore di 4 byte interpretato come un numero intero senza segno a 32 bit che specifica il numero di byte che contengono il contenuto crittografato per il frame.

## Contenuto crittografato

I contenuti crittografati (testo codificato) per il frame come sono restituiti dall'[algoritmo di crittografia](#).

## Tag di autenticazione

Il valore di autenticazione per il frame. Viene utilizzato per autenticare il frame completo.

## Struttura piè di pagina

Quando vengono utilizzati gli [algoritmi con firma](#), il formato del messaggio contiene un piè di pagina. Il piè di pagina del messaggio contiene una [firma digitale](#) calcolata sull'intestazione e sul corpo del messaggio. La tabella seguente descrive i campi che costituiscono il piè di pagina. I byte vengono aggiunti nell'ordine mostrato. La struttura del piè di pagina del messaggio è la stessa nelle versioni 1 e 2 del formato dei messaggi.

### Struttura piè di pagina

Campo	Lunghezza, in byte
<a href="#">Lunghezza firma</a>	2
<a href="#">Firma</a>	Variabile. Pari al valore specificato nei 2 byte precedenti (lunghezza firma).

### Lunghezza firma

La lunghezza della firma. Si tratta di un valore di 2 byte interpretato come un numero intero senza segno a 16 bit che specifica il numero di byte che contengono la firma.

### Firma

La firma.

## Esempi di formati di messaggi AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

Per le specifiche che definiscono gli elementi di una corretta AWS Encryption SDK implementazione, si veda la [AWS Encryption SDK Specificazione](#) in GitHub.

I seguenti argomenti mostrano esempi di formati di messaggi AWS Encryption SDK. Ogni esempio illustra i byte raw, in notazione esadecimale, seguiti da una descrizione di ciò che questi byte rappresentano.

## Argomenti

- [Dati incorniciati \(formato messaggio versione 1\)](#)
- [Dati incorniciati \(formato messaggio versione 2\)](#)
- [Dati non inclusi in frame \(formato messaggio versione 1\)](#)

## Dati incorniciati (formato messaggio versione 1)

L'esempio seguente mostra il formato dei messaggi per i dati con frame nella [versione 1 del formato di messaggio](#).

```
+-----+
| Header |
+-----+
01          Version (1.0)
80          Type (128, customer authenticated encrypted
  data)
0378       Algorithm ID (see Riferimenti agli
  algoritmi)
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	0015	AAD Key-Value Pair 3, Value ("example")
6177732D 63727970 746F2D70 75626C69		AAD Key-Value Pair 4, Key Length (21)
632D6B65 79		AAD Key-Value Pair 4, Key ("aws-crypto-
0044		AAD Key-Value Pair 4, Value Length (68)
416A4173 7569326F 7430364C 4B77715A		AAD Key-Value Pair 4, Value
("AjAsui2ot06LKwqZXDJnU/Aqc2vD+00kp0Z1cc8Tg2qd7rs5aLTg7lvfUEW/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 ACBB08B2 1C785089
A90DB923 699A1495 C3B31B50 0A48A830
201E3AD9 1EA6DA14 7F6496DB 6BC104A4
DEB7F372 375ECB28 9BF84B6D 2863889F
CB80A167 9C361C4B 5EC07438 7A4822B4
A7D9D2CC 5150D414 AF75F509 FCE118BD
6D1E798B AEBA4CDB AD009E5F 1A571B77
0041BC78 3E5F2F41 8AF157FD 461E959A

```

BB732F27 D83DC36D CC9EBC05 00D87803	
57F2BB80 066971C2 DEEA062F 4F36255D	
E866C042 E1382369 12E9926B BA40E2FC	
A820055F FB47E428 41876F14 3B6261D9	
5262DB34 59F5D37E 76E46522 E8213640	
04EE3CC5 379732B5 F56751FA 8E5F26AD	Frame 1, Authentication Tag
00000002	Frame 2, Sequence Number (2)
F1140984 FF25F943 959BE514	Frame 2, IV
216C7C6A 2234F395 F0D2D9B9 304670BF	Frame 2, Encrypted Content
A1042608 8A8BCB3F B58CF384 D72EC004	
A41455B4 9A78BAC9 36E54E68 2709B7BD	
A884C1E1 705FF696 E540D297 446A8285	
23DFEE28 E74B225A 732F2C0C 27C6BDA2	
7597C901 65EF3502 546575D4 6D5EBF22	
1FF787AB 2E38FD77 125D129C 43D44B96	
778D7CEE 3C36625F FF3A985C 76F7D320	
ED70B1F3 79729B47 E7D9B5FC 02FCE9F5	
C8760D55 7779520A 81D54F9B EC45219D	
95941F7E 5CBAEAC8 CEC13B62 1464757D	
AC65B6EF 08262D74 44670624 A3657F7F	
2A57F1FD E7060503 AC37E197 2F297A84	
DF1172C2 FA63CF54 E6E2B9B6 A86F582B	
3B16F868 1BBC5E4D 0B6919B3 08D5ABCF	
FECDC4A4 8577F08B 99D766A1 E5545670	
A61F0A3B A3E45A84 4D151493 63ECA38F	Frame 2, Authentication Tag
FFFFFFFF	Final Frame, Sequence Number End
00000003	Final Frame, Sequence Number (3)
35F74F11 25410F01 DD9E04BF	Final Frame, IV
0000008E	Final Frame, Encrypted Content Length (142)
F7A53D37 2F467237 6FBD0B57 D1DFE830	Final Frame, Encrypted Content
B965AD1F A910AA5F 5EFFFFFF4 BC7D431C	
BA9FA7C4 B25AF82E 64A04E3A A0915526	
88859500 7096FABB 3ACAD32A 75CFED0C	
4A4E52A3 8E41484D 270B7A0F ED61810C	
3A043180 DF25E5C5 3676E449 0986557F	
C051AD55 A437F6BC 139E9E55 6199FD60	
6ADC017D BA41CDA4 C9F17A83 3823F9EC	
B66B6A5A 80FDB433 8A48D6A4 21CB	
811234FD 8D589683 51F6F39A 040B3E3B	Final Frame, Authentication Tag
+-----+	
Footer	
+-----+	
0066	Signature Length (102)
30640230 085C1D3C 63424E15 B2244448	Signature

```

639AED00 F7624854 F8CF2203 D7198A28
758B309F 5EFD9D5D 2E07AD0B 467B8317
5208B133 02301DF7 2DFC877A 66838028
3C6A7D5E 4F8B894E 83D98E7C E350F424
7E06808D 0FE79002 E24422B9 98A0D130
A13762FF 844D

```

## Dati incorniciati (formato messaggio versione 2)

L'esempio seguente mostra il formato dei messaggi per i dati con frame nella [versione 2 del formato di messaggio](#).

```

+-----+
| 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
("QXRnM3JwOEVBNTFhNnBmaTk3MUI1TnRk3NHp0Mn1ZWE5vSmtwRHFPc0dIYkVaVDRqME50M1FkRStmbTFVY01WdThnPT0=
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
```

## Dati non inclusi in frame (formato messaggio versione 1)

L'esempio seguente mostra il formato dei messaggi per i dati non-framed.

### Note

Quando possibile, usa dati incorniciati. AWS Encryption SDKSupporta dati senza frame solo per uso precedente. Alcune implementazioni linguistiche di AWS Encryption SDK possono ancora generare testo cifrato senza frame. Tutte le implementazioni linguistiche supportate possono decrittografare testo cifrato con e senza frame.

```
+-----+
| Header |
+-----+
01                               Version (1.0)
80                               Type (128, customer authenticated encrypted
  data)
0378                             Algorithm ID (see Riferimenti agli
  algoritmi)
B8929B01 753D4A45 C0217F39 404F70FF   Message ID (random 128-bit value)
008E                             AAD Length (142)
0004                             AAD Key-Value Pair Count (4)
0005                             AAD Key-Value Pair 1, Key Length (5)
30746869 73                       AAD Key-Value Pair 1, Key ("This")
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 ("lan")
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 public-key")	AAD Key-Value Pair 4, Key ("aws-crypto- public-key")
632D6B65 79	
0044	AAD Key-Value Pair 4, Value Length (68)
41734738 67473949 6E4C5075 3136594B	AAD Key-Value Pair 4, Value
("AsG8gG9InLPu16YK1qXTOD+nykG8YqHAHqecj8aXfD2e5B4gtVE73dZkyClA+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 DAEB44AF 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
+-----+
| Footer |
+-----+
0067
30650230 7229DDF5 B86A5B64 54E4D627
CBE194F1 1CC0F8CF D27B7F8B F50658C0
BE84B355 3CED1721 A0BE2A1B 8E3F449E
1BEB8281 023100B2 0CB323EF 58A4ACE3
1559963B 889F72C3 B15D1700 5FB26E61
331F3614 BC407CEE B86A66FA CBF74D9E

```

Authentication Tag

Signature Length (103)

Signature

34CB7E4B 363A38

## Riferimento ai dati autenticati aggiuntivi (AAD) del corpo per AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

[Per le specifiche che definiscono gli elementi di una corretta implementazione, si veda la Specificazione in. AWS Encryption SDK](#) [AWS Encryption SDK GitHub](#)

È necessario fornire dati autenticati aggiuntivi (AAD) all'[algoritmo AES-GCM](#) per ciascuna operazione di crittografia. Questo vale sia per i [dati del corpo](#) framed e non framed. Per ulteriori informazioni su AAD e su come viene utilizzato in Galois/Counter Mode (GCM), consulta l'articolo sulle [raccomandazioni per le modalità di cifratura a blocchi: Galois/Counter Mode \(GCM\) e GMAC](#).

La tabella seguente descrive i campi che costituiscono l'AAD del corpo. I byte vengono aggiunti nell'ordine mostrato.

### Struttura dell'AAD del corpo

Campo	Lunghezza, in byte
<a href="#">ID messaggio</a>	16
<a href="#">Contenuto dell'AAD del corpo</a>	Variabile. Vedi Contenuto dell'AAD del corpo nell'elenco qui di seguito.
<a href="#">Numero sequenza</a>	4
<a href="#">Lunghezza del contenuto</a>	8

## ID messaggio

Lo stesso valore [ID messaggio](#) impostato nell'intestazione del messaggio.

## Contenuto dell'AAD del corpo

Un valore codificato UTF-8, stabilito dal tipo di dati del corpo utilizzati.

Per i [dati non framed](#), utilizza il valore `AWSKMSEncryptionClient Single Block`.

Per i frame regolari nei [dati framed](#), utilizza il valore `AWSKMSEncryptionClient Frame`.

Per il frame finale nei [dati framed](#), utilizza il valore `AWSKMSEncryptionClient Final Frame`.

## Numero sequenza

È un valore a 4 byte interpretato come un numero intero senza segno a 32 bit.

Per i [dati framed](#), questo è il numero di sequenza di frame.

Per i [dati non framed](#), usa il valore 1, codificato come 4 byte `00 00 00 01` in notazione esadecimale.

## Lunghezza del contenuto

La lunghezza in byte dei dati di testo non crittografato forniti all'algoritmo per la crittografia. È un valore a 8 byte interpretato come un numero intero senza segno a 64 bit.

## Riferimenti agli algoritmi AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

Per le specifiche che definiscono gli elementi di una corretta AWS Encryption SDK implementazione, vedere la [AWS Encryption SDK Specificazione](#) in GitHub.

Se state creando una libreria personale in grado di leggere e scrivere testi cifrati compatibili con AWS Encryption SDK, dovrete capire come AWS Encryption SDK implementa le suite di algoritmi supportate per crittografare i dati grezzi.

AWS Encryption SDK supporta le seguenti suite di algoritmi. Tutte le suite di algoritmi AES-GCM hanno un [vettore di inizializzazione](#) a 12 byte e un tag di autenticazione AES-GCM a 16 byte. La suite di algoritmi predefinita varia a seconda della versione e della politica di impegno chiave selezionata. AWS Encryption SDK Per i dettagli, consulta la [politica di impegno e la suite di algoritmi](#).

### Suite di algoritmi AWS Encryption SDK

ID algoritmo	Versione del formato del messaggio	Algoritmo di crittografia	Lunghezza della chiave dati (bit)	Algoritmo di derivazione della chiave	Algoritmo di firma	Algoritmo di impegno chiave	Lunghezza dei dati della suite di algoritmi (byte)
05 78	0x02	AES-GCM	256	HKDF con SHA-512	ECDSA con P-384 e SHA-384	HKDF con SHA-512	32 (impegno chiave)
04 78	0x02	AES-GCM	256	HKDF con SHA-512	Nessuno	HKDF con SHA-512	32 (impegno chiave)
03 78	0x01	AES-GCM	256	HKDF con SHA-384	ECDSA con P-384 e SHA-384	Nessuno	N/D
03 46	0x01	AES-GCM	192	HKDF con SHA-384	ECDSA con P-384 e SHA-384	Nessuno	N/D

ID algoritmo	Versione del formato del messaggio	Algoritmo di crittografia	Lunghezza della chiave dati (bit)	Algoritmo di derivazione della chiave	Algoritmo di firma	Algoritmo di impegno chiave	Lunghezza dei dati della suite di algoritmi (byte)
02 14	0x01	AES-GCM	128	HKDF con SHA-256	ECDSA con P-256 e SHA-256	Nessuno	N/D
01 78	0x01	AES-GCM	256	HKDF con SHA-256	Nessuno	Nessuno	N/D
01 46	0x01	AES-GCM	192	HKDF con SHA-256	Nessuno	Nessuno	N/D
01 14	0x01	AES-GCM	128	HKDF con SHA-256	Nessuno	Nessuno	N/D
00 78	0x01	AES-GCM	256	Nessuno	Nessuna	Nessuno	N/D
00 46	0x01	AES-GCM	192	Nessuno	Nessuna	Nessuno	N/D
00 14	0x01	AES-GCM	128	Nessuno	Nessuna	Nessuno	N/D

## ID algoritmo

Un valore esadecimale a 2 byte che identifica in modo univoco l'implementazione di un algoritmo. [Questo valore viene memorizzato nell'intestazione del messaggio del testo cifrato.](#)

## Versione del formato del messaggio

La versione del formato del messaggio. Le suite di algoritmi con impegno chiave utilizzano il formato dei messaggi versione 2 (0x02). Le suite di algoritmi senza impegno chiave utilizzano il formato dei messaggi versione 1 (0x01).

## Lunghezza dei dati della suite di algoritmi

La lunghezza in byte dei dati specifici della suite di algoritmi. Questo campo è supportato solo nella versione 2 del formato di messaggio (0x02). Nella versione 2 del formato di messaggio (0x02), questi dati vengono visualizzati nel `Algorithm suite data` campo dell'intestazione del messaggio. Le suite di algoritmi che supportano [l'impegno delle chiavi](#) utilizzano 32 byte per la stringa di impegno chiave. Per ulteriori informazioni, consulta `Key commit algorithm` in questo elenco.

## Lunghezza chiave dati

La lunghezza della [chiave dati](#) in bit. AWS Encryption SDKSupporta chiavi a 256 bit, 192 bit e 128 bit. [La chiave dati viene generata da un portachiavi o da una chiave master.](#)

In alcune implementazioni, questa chiave dati viene utilizzata come input per una funzione di derivazione delle `extract-and-expand` chiavi basata su HMAC (HKDF). L'output dell'HKDF viene usato come chiave di crittografia dei dati nell'algoritmo di crittografia. Per ulteriori informazioni, consulta `Algoritmo di derivazione delle chiavi` in questo elenco.

## Algoritmo di crittografia

Il nome e la modalità dell'algoritmo di crittografia utilizzato. Le suite di algoritmi AWS Encryption SDK utilizzano l'algoritmo di crittografia `Advanced Encryption Standard (AES) con Galois/Counter Mode (GCM)`.

## Algoritmo di impegno chiave

L'algoritmo utilizzato per calcolare la stringa di impegno chiave. L'output viene memorizzato nel `Algorithm suite data` campo dell'intestazione del messaggio e viene utilizzato per convalidare la chiave di dati per l'impegno chiave.

Per una spiegazione tecnica dell'aggiunta dell'impegno chiave a una suite di algoritmi, vedi [Key Committing AEAD in Cryptology ePrint Archive](#).

## Algoritmo di derivazione della chiave

La funzione di derivazione delle extract-and-expand chiavi basata su HMAC (HKDF) utilizzata per derivare la chiave di crittografia dei dati. AWS Encryption SDK [Utilizza l'HKDF definito nella RFC 5869.](#)

Suite di algoritmi senza impegno chiave (ID dell'algoritmo —) 01xx 03xx

- La funzione hash utilizzata è SHA-384 o SHA-256, a seconda della suite di algoritmi.
- Per la fase di estrazione:
  - Non vengono utilizzati salt. Secondo la RFC, il sale è impostato su una stringa di zeri. La lunghezza della stringa è uguale alla lunghezza dell'output della funzione hash, che è di 48 byte per SHA-384 e 32 byte per SHA-256.
  - Il materiale di codifica di input è la chiave dati del portachiavi o del fornitore della chiave principale.
- Per la fase di espansione:
  - La chiave di input pseudo-casuale è l'output della fase di estrazione.
  - Le informazioni di input sono una concatenazione dell'ID dell'algoritmo e dell'ID del messaggio (in quest'ordine).
  - La lunghezza del materiale di codifica di output è la lunghezza della chiave Data. Questo output viene usato come chiave di crittografia dei dati nell'algoritmo di crittografia.

Suite di algoritmi con impegno chiave (ID dell'algoritmo 04xx e05xx)

- La funzione hash utilizzata è SHA-512.
- Per la fase di estrazione:
  - Il sale è un valore casuale crittografico a 256 bit. Nella [versione 2 del formato di messaggio](#) (0x02), questo valore viene memorizzato nel campo. MessageID
  - Il materiale di codifica iniziale è la chiave dati del portachiavi o del fornitore della chiave principale.
- Per la fase di espansione:
  - La chiave di input pseudo-casuale è l'output della fase di estrazione.
  - L'etichetta della chiave è costituita dai byte della stringa con codifica UTF-8 in ordine di byte big endian. DERIVEKEY
  - Le informazioni di input sono una concatenazione dell'ID dell'algoritmo e dell'etichetta della chiave (in quest'ordine).

- La lunghezza del materiale di codifica di output è la lunghezza della chiave Data. Questo output viene usato come chiave di crittografia dei dati nell'algoritmo di crittografia.

### Versione del formato del messaggio

La versione del formato dei messaggi utilizzata con la suite di algoritmi. Per informazioni dettagliate, vedi [Riferimenti a formati di messaggi](#).

### Algoritmo di firma

L'algoritmo di firma utilizzato per generare una [firma digitale](#) sull'intestazione e sul corpo del testo cifrato. AWS Encryption SDK utilizza l'Elliptic Curve Digital Signature Algorithm (ECDSA) con le seguenti specifiche:

- La curva ellittica utilizzata è la curva P-384 o P-256, come specificato dall'ID dell'algoritmo. Queste curve sono definite in [Digital Signature Standard \(DSS\) \(FIPS PUB 186-4\)](#).
- La funzione hash utilizzata è SHA-384 (con la curva P-384) o SHA-256 (con la curva P-256).

## Riferimento al vettore di inizializzazione AWS Encryption SDK

Le informazioni presenti su questa pagina sono un riferimento per la creazione della tua libreria di crittografia, compatibile con AWS Encryption SDK. Se non stai creando la tua libreria di crittografia compatibile, allora è probabile che queste informazioni non ti serviranno.

Per utilizzare AWS Encryption SDK in uno dei linguaggi di programmazione supportati, consulta [Linguaggi di programmazione](#).

[Per le specifiche che definiscono gli elementi di una corretta AWS Encryption SDK implementazione, vedere la Specificazione in. AWS Encryption SDK GitHub](#)

AWS Encryption SDK fornisce i [vettori di inizializzazione](#) (IV) richiesti da tutte le [suite di algoritmi](#) supportate. Il kit SDK utilizza i numeri di sequenza di frame per creare un IV in modo che due frame nello stesso messaggio non abbiano lo stesso IV.

Ogni IV a 96 bit (12 byte) è costituito da due array di byte big-endian concatenati nell'ordine seguente:

- 64 bit: 0 (riservato per uso futuro)

- 32 bit: numero di sequenza del frame. Per il tag di autenticazione dell'intestazione, questo valore è costituito da tutti zeri.

Prima dell'introduzione del [caching della chiave di dati](#), AWS Encryption SDK ha sempre utilizzato una nuova chiave di dati per crittografare ciascun messaggio e ha generato tutti gli IV in modo casuale. Gli IV generati in modo casuale erano sicuri dal punto di vista crittografico perché le chiavi di dati non venivano mai riutilizzate. Quando l'SDK ha introdotto il caching della chiave dei dati, che intenzionalmente riutilizza le chiavi dei dati, abbiamo modificato il modo in cui l'SDK genera gli IV.

L'utilizzo di IV deterministici che non possono essere ripetuti all'interno di un messaggio aumenta in modo significativo il numero di chiamate che possono essere eseguite in modo sicuro in un'unica chiave di dati. Inoltre, le chiavi di dati che vengono memorizzate nella cache utilizzano sempre una suite di algoritmi con una [funzione di derivazione della chiave](#). L'utilizzo di un IV deterministico con una funzione di derivazione chiave pseudo-casuale per derivare le chiavi di crittografia da una chiave dei dati consente a AWS Encryption SDK di crittografare  $2^{32}$  messaggi senza superare i limiti di crittografia.

## AWS KMS Dettagli tecnici del portachiavi gerarchico

Il [portachiavi AWS KMS Hierarchical](#) utilizza una chiave dati univoca per crittografare ogni campo e crittografa ogni chiave di dati con una chiave di avvolgimento unica derivata da una chiave branch attiva. Utilizza una [derivazione della chiave](#) in modalità contatore con una funzione pseudocasuale con HMAC SHA-256 per derivare la chiave di wrapping a 32 byte con i seguenti input.

- Un sale casuale da 16 byte
- La chiave branch attiva
- Il valore [codificato UTF-8](#) per l'identificatore del provider di chiavi "» aws-kms-hierarchy

Il portachiavi Hierarchical utilizza la chiave di wrapping derivata per crittografare una copia della chiave dati in chiaro utilizzando AES-GCM-256 con un tag di autenticazione a 16 byte e i seguenti input.

- La chiave di wrapping derivata viene utilizzata come chiave di crittografia AES-GCM
- La chiave dati viene utilizzata come messaggio AES-GCM
- Un vettore di inizializzazione casuale (IV) a 12 byte viene utilizzato come AES-GCM IV
- Dati autenticati aggiuntivi (AAD) contenenti i seguenti valori serializzati.

Valore	Lunghezza in byte	Interpretato come
"aws-kms-hierarchy"	17	codificato UTF-8
L'identificatore della chiave di filiale	Variabile	codificato UTF-8
La versione Branch Key	16	codificato UTF-8
Contesto di crittografia	Variabile	coppie chiave-valore codificate in UTF-8

# Cronologia dei documenti per la AWS Encryption SDK Developer Guide

Questo argomento descrive gli aggiornamenti importanti alla Guida per gli sviluppatori di AWS Encryption SDK.

## Argomenti

- [Aggiornamenti recenti](#)
- [Aggiornamenti precedenti](#)

## Aggiornamenti recenti

La tabella seguente descrive le modifiche significative apportate a questa documentazione dal Novembre 2017. Oltre alle modifiche maggiori elencate qui, aggiorniamo la documentazione di frequente per migliorare le descrizioni e gli esempi e per dar spazio al feedback inviatoci. Per ricevere una notifica sulle modifiche rilevanti, iscriversi al feed RSS.

Modifica	Descrizione	Data
<a href="#">SDK di crittografia AWS per Javaversione 3.x</a>	Si integra SDK di crittografia AWS per Java con la libreria dei fornitori di materiali . Aggiunge il supporto per i portachiavi e il contesto di crittografia richiesto CMM.	6 dicembre 2023
<a href="#">AWS Encryption SDKper.NET versione 4.x</a>	Aggiunge il supporto per il portachiavi AWS KMS Hierarchical, il contesto di crittografia richiesto CMM e i portachiavi RSA asimmetrici. AWS KMS	12 ottobre 2023

<a href="#">Disponibilità generale</a>	Presentazione del AWS Encryption SDK supporto per.NET.	17 maggio 2022
<a href="#">Modifica della documentazione</a>	Sostituisci il AWS Key Management Service termine Customer Master Key (CMK) con AWS KMS key«chiave KMS».	30 agosto 2021
<a href="#">Disponibilità generale</a>	È stato aggiunto il supporto perAWS Key Management Service. (AWS KMS) Chiavi multiregionali. Le chiavi multiregionali sono AWS KMS chiavi diverse Regioni AWS che possono essere utilizzat e in modo intercambiabile perché hanno lo stesso ID di chiave e lo stesso materiale chiave.	8 giugno 2021
<a href="#">Disponibilità generale</a>	Documentazione aggiunta e aggiornata sul processo di decrittografia dei messaggi migliorato.	11 maggio 2021
<a href="#">Disponibilità generale</a>	Documentazione aggiunta e aggiornata per la versione di disponibilità generale di AWS Encryption CLI versione 1.8. x per sostituire la versione 1.7 di AWS Encryption CLI. x e AWS Encryption CLI 2.1. x per sostituire AWS Encryption CLI 2.0. x.	27 ottobre 2020

<a href="#">Disponibilità generale</a>	Documentazione aggiunta e aggiornata per la versione a disponibilità generale delle AWS Encryption SDK versioni 1.7. x e 2.0. x, tra cui una <a href="#">guida alle migliori pratiche</a> , una <a href="#">guida alla migrazione</a> , <a href="#">concetti</a> aggiornati, <a href="#">argomenti aggiornati sui linguaggi di programmazione</a> , un <a href="#">riferimento aggiornato alle suite di algoritmi</a> , un <a href="#">riferimento aggiornato al formato dei messaggi</a> e un nuovo <a href="#">esempio di formato dei messaggi</a> .	24 settembre 2020
<a href="#">Disponibilità generale</a>	È stata aggiunta e aggiornata la documentazione della versione di disponibilità generale di <a href="#">SDK di crittografia AWS per JavaScript</a> .	1 ottobre 2019
<a href="#">Versione di anteprima</a>	È stata aggiunta e aggiornata la documentazione della versione beta pubblica di <a href="#">SDK di crittografia AWS per JavaScript</a> .	21 giugno 2019
<a href="#">Disponibilità generale</a>	È stata aggiunta e aggiornata la documentazione della versione di disponibilità generale di <a href="#">SDK di crittografia AWS per C</a> .	16 maggio 2019

<a href="#">Versione di anteprima</a>	È stata aggiunta e aggiornata la documentazione della versione di anteprima di <a href="#">SDK di crittografia AWS per C</a> .	5 febbraio 2019
<a href="#">Nuova versione</a>	Aggiunta documentazione dell' <a href="#">interfaccia a riga di comando</a> per l'AWS Encryption SDK.	20 novembre 2017

## Aggiornamenti precedenti

La tabella seguente descrive le modifiche significative alla AWS Encryption SDK Developer Guide prima di novembre 2017.

Modifica	Descrizione	Data
Nuovo rilascio	<p>Aggiunto il capitolo <a href="#">Caching della chiave dei dati</a> per la nuova funzionalità.</p> <p>Aggiunto l'argomento <a href="#">the section called "Riferimento al vettore di inizializzazione"</a> che spiega che l'SDK è stato modificato: dalla generazione di IV random è passato alla creazione di IV deterministici.</p> <p>È stato aggiunto l'<a href="#">the section called "Concetti"</a> argomento per spiegare i concetti, incluso il nuovo gestore dei materiali crittografici.</p>	31 luglio 2017
Aggiornamento	La documentazione <a href="#">Riferimenti a formati di messaggi</a> è stata	21 marzo 2017

Modifica	Descrizione	Data
	<p>estesa in una nuova sezione <a href="#">Riferimento AWS Encryption SDK</a>.</p> <p>È stata aggiunta una sezione su AWS Encryption SDK di <a href="#">Suite di algoritmi supportate</a>.</p>	
Nuovo rilascio	AWS Encryption SDK ora supporta il linguaggio di programmazione <a href="#">Python</a> , oltre a <a href="#">Java</a> .	21 marzo 2017
Versione iniziale	Versione iniziale di AWS Encryption SDK e di questa documentazione.	22 marzo 2016

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.