ユーザーガイド

AWS CloudHSM

AWS CloudHSM: ユーザーガイド

Table of Contents

とは AWS CloudHSM

AWS CloudHSM は、 AWS クラウドの利点とハードウェアセキュリティモジュール (HSMs。ハードウェアセキュリティモジュール (HSM) は、暗号化オペレーションを処理し、暗号化キーの安全なストレージを提供するコンピューティングデバイスです。を使用すると AWS CloudHSM、AWS クラウド内の高可用性 HSMs を完全に制御し、低レイテンシーのアクセスと、HSM 管理 (バックアップ、プロビジョニング、設定、メンテナンスを含む) を自動化する安全な信頼ルートを利用できます。

AWS CloudHSM は、お客様にさまざまな利点を提供します。

FIPS クラスターと非 FIPS クラスターへのアクセス

AWS CloudHSM は、FIPS と非 FIPS の 2 つのモードでクラスターを提供します。FIPS モードでは、連邦情報処理規格 (FIPS) によって承認されたキーとアルゴリズムのみを使用できます。非 FIPS モードは、FIPS の承認に関係なく AWS CloudHSM、 でサポートされているすべてのキーとアルゴリズムを提供します。詳細については、「AWS CloudHSM クラスターモードと HSM タイプ」を参照してください。

HSMsは、汎用、単一テナント、および FIPS モードのクラスターで検証された FIPS 140-2 レベル 3 です。

AWS CloudHSM は、アプリケーションのアルゴリズムとキーの長さが事前に定義されたフルマネージド AWS サービスと比較して、柔軟性を高める汎用 HSMs を使用します。標準に準拠し、シングルテナントで、FIPS モードのクラスターに対して FIPS 140-2 レベル 3 検証済みの HSMs を提供しています。FIPS 140-2 レベル 3 検証の制限外のユースケースのお客様向けに、 は非 FIPS モードでクラスター AWS CloudHSM も提供します。詳細については、「AWS CloudHSM クラスター」を参照してください。

E2E 暗号化は AWS には表示されません。

データプレーンは end-to-end (E2E) 暗号化されており、AWS には表示されないため、独自のユーザー管理 (IAM ロール以外) を制御します。このコントロールのトレードオフは、マネージド型の AWS サービスを使用した場合よりも責任が大きくなることです。

キー、アルゴリズム、アプリケーション開発を完全に制御できます。

AWS CloudHSM を使用すると、使用するアルゴリズムとキーを完全に制御できます。暗号化キー (セッションキー、トークンキー、対称キー、非対称キーペアを含む) の生成、保存、インポート、エクスポート、管理、使用ができます。さらに、 AWS CloudHSM SDKsを使用すると、

アプリケーション開発、アプリケーション言語、スレッド、アプリケーションが物理的に存在する場所を完全に制御できます。

暗号化ワークロードをクラウドに移行します。

Public Key Cryptography Standards #11 (PKCS #11)、Java Cryptographic Extension (JCE)、Cryptography API: Next Generation (CNG)、または Key Storage Provider (KSP) を使用するパブリックキーインフラストラクチャを移行するお客様は、アプリケーションの変更を少なく AWS CloudHSM して に移行できます。

でできることの詳細については AWS CloudHSM、以下のトピックを参照してください。の使用を開始する準備ができたら AWS CloudHSM、「」を参照してください開始。

> **Note**
>
> データの暗号化キーを作成および管理するマネージド型サービスは欲しいが、独自の HSM を運用したくないまたは不要である場合、AWS Key Management Service の使用を検討してください。
> クラウド内の支払い処理アプリケーションの支払い HSM とキーを管理する柔軟性の高いサービスをお探しの場合は、AWS Payment Cryptography の使用を検討してください。

内容

AWS CloudHSM ユースケース

AWS CloudHSM は、さまざまな目標を達成するために使用できます。このトピックのコンテンツでは、 でできることの概要を説明します AWS CloudHSM。

規制の確実な順守

企業のセキュリティ標準に準拠する必要がある企業は、 AWS CloudHSM を使用して、機密性の高いデータを保護するプライベートキーを管理できます。が提供する HSMs AWS CloudHSM は FIPS 140-2 レベル 3 認定を受けており、PCI DSS に準拠しています。さらに、 AWS CloudHSM

は PCI PIN に準拠し、PCI-3DS に準拠しています。詳細については、「コンプライアンス」を参照してください。

データの暗号化と復号

を使用して AWS CloudHSM 、機密性の高いデータ、転送中の暗号化、保管中の暗号化を保護するプライベートキーを管理します。さらに、 は、複数の暗号化 SDKsとの標準準拠の統合 AWS CloudHSM を提供します。

文書へのプライベートキーとパブリックキーを使用した署名と検証

暗号化では、プライベートキーを使用して文書に署名すると、受信者はパブリックキーを使用して、他の誰でもないお客様が実際に文書を送信したことを検証できます。を使用して AWS CloudHSM 、この目的のために特別に設計された非対称パブリックキーとプライベートキーのペアを作成します。

HMAC と CMAC を使用したメッセージの認証

暗号化では、Cipher Message Authentication Codes (CMACs) と Hash-based Message Authentication Code (HMAC) を使用して、安全でないネットワークを介して送信されるメッセージを認証し、整合性を確保します。を使用すると AWS CloudHSM、HMACs と CMACs。

AWS CloudHSM と の利点を活用する AWS Key Management Service

お客様は AWS CloudHSM と を組み合わせて、FIPS 140-2 Level 3 認定のシングルテナント環境にキーマテリアルAWS KMSを保存できると同時に、 のキー管理、スケーリング、クラウド統合のメリットも享受できます AWS KMS。その方法について詳しくは、「AWS Key Management Service 開発者ガイド」の「AWS CloudHSM キーストア」を参照してください。

ウェブサーバーの SSL/TLS 処理のオフロード

インターネット経由でデータを安全に送信するために、ウェブサーバーでは、パブリック/プライベートのキーペアと SSL/TLS パブリックキー証明書を使用して、HTTPS セッションを確立します。このプロセスにはウェブサーバーに対する多くの計算が含まれますが、この一部を AWS CloudHSM クラスターにオフロードすることで、セキュリティを強化しながら、計算の負担を軽減できます。で SSL/TLS オフロードを設定する方法については AWS CloudHSM、「」を参照してくださいSSL/TLS のオフロード。

透過的なデータ暗号化 (TDE) の有効化

透過的なデータ暗号化 (TDE) を使用して、データベースファイルを暗号化します。TDE を使用すると、データベースソフトウェアはデータをディスクに保存する前に暗号化します。TDE マスター暗号化キーを AWS CloudHSMの HSM に保存すると、セキュリティを強化するのに役立ちます。で Oracle TDE を設定する方法については AWS CloudHSM、「」を参照してくださいOracle Database 暗号化。

発行認証機関 (CA) のプライベートキーの管理

認証機関 (CA) は、パブリックキーを ID (個人または組織) にバインドするデジタル証明書を発行する信頼されたエンティティです。CA を操作するには、CA によって発行された証明書に署名するプライベートキーを保護して、信頼関係を維持する必要があります。このようなプライベートキーを AWS CloudHSM クラスターに保存し、HSMsを使用して暗号化署名オペレーションを実行できます。

乱数の生成

暗号化キーを作成するための乱数の生成は、オンラインセキュリティの中核です。 AWS CloudHSM は、ユーザーが制御する HSMs で乱数を安全に生成するために使用でき、ユーザーのみに表示されます。

AWS CloudHSM の仕組み

このトピックでは、データを安全に暗号化し、独自の Amazon Virtual Private Cloud (VPC) で HSMs. AWS CloudHSM operates で暗号化オペレーションを実行するために使用する基本概念とアーキテクチャの概要を説明します。を使用する前に AWS CloudHSM、まずクラスターを作成し、そのクラスターに HSMs を追加し、ユーザーとキーを作成してから、クライアント SDKs を使用して HSMs をアプリケーションと統合します。これが完了したら、クライアント SDK ログ AWS CloudTrail、監査ログ、Amazon を使用して CloudWatch をモニタリング AWS CloudHSMします。

AWS CloudHSMの基本概念と、それらが連携してデータを保護する方法について説明します。

トピック

- AWS CloudHSM クラスター
- HSM ユーザー
- HSM キー

- Client SDK
- AWS CloudHSM クラスターバックアップ
- リージョン

AWS CloudHSM クラスター

個々の HSMs を同期され、冗長で、可用性の高い方法で連携させること AWS CloudHSM は難しい場合がありますが、クラスター にハードウェアセキュリティモジュール (HSMsを提供することで、面倒な作業が軽減されます。 クラスターは、同期 AWS CloudHSM を維持する個々の HSMsのコレクションです。クラスター内にある HSM でタスクまたはオペレーションを行うと、そのクラスター内の他の HSM は、自動的に最新の状態に維持されます。

AWS CloudHSM は、FIPS と非 FIPS の 2 つのモードでクラスターを提供します。FIPS モードでは、連邦情報処理規格 (FIPS) によって承認されたキーとアルゴリズムのみを使用できます。非 FIPS モードは、FIPS の承認に関係なく AWS CloudHSM、 でサポートされているすべてのキーとアルゴリズムを提供します。 は、hsm1.medium と hsm2m.medium の HSMs AWS CloudHSM も提供します。各 HSM タイプとクラスターモードの違いの詳細については、「」を参照してくださいAWS CloudHSM クラスターモードと HSM タイプ。

可用性、耐久性、スケーラビリティの目標を達成するには、複数のアベイラビリティーゾーンにまたがるクラスター内の HSM の数を設定します。1～28 HSMs を持つクラスターを作成できます (デフォルトの制限は、リージョン ごとに AWS アカウント 1 つにつき 6 個の HSMs AWSです)。HSMsの異なるアベイラビリティーゾーンに配置することができます。 AWS クラスターに HSM を追加すると、高いパフォーマンスを実現できます。複数のアベイラビリティーゾーンにクラスターを分散すると、冗長性と高可用性を実現します。

クラスターの詳細については、「AWS CloudHSM クラスターの管理」を参照してください。

クラスターを作成するには、「開始」を参照してください。

HSM ユーザー

ほとんどの AWS サービスやリソースとは異なり、 AWS Identity and Access Management (IAM) ユーザーや IAM ポリシーを使用してクラスター内のリソースにアクセスすることはできません。代わりに、 AWS CloudHSM クラスター内の HSM で HSM ユーザーを直接使用します。 HSMs

HSM ユーザーは IAM ユーザーとは異なります。正しい認証情報を持つ IAM ユーザーは、AWS API を介してリソースを操作することで HSM を作成できます。E2E 暗号化は AWS には表示されないた

め、認証情報は HSM 上で直接行われるため、HSM でのオペレーションを認証するには HSM ユーザー認証情報を使用する必要があります。HSMは、定義および管理する認証情報を使用して、各HSMユーザーを認証します。各 HSM ユーザーには、HSM でユーザーとして実行できるオペレーションを判断する タイプ があります。各HSMは、CloudHSMCLI を使用して定義する認証情報を使用して、各HSMユーザーを認証します。

以前の SDK バージョンシリーズ を使用している場合は、CloudHSM 管理ユーティリティ (CMU) を使用します。

HSM キー

AWS CloudHSM では、AWS CloudHSM クラスターにあるシングルテナント HSM で暗号化キーを安全に生成、保存、管理できます。キーは対称でも非対称でもよく、単一セッションではセッションキー (エフェメラルキー)、長期使用ではトークンキー (永続キー) にでき、AWS CloudHSM からのエクスポートと AWS CloudHSM へのインポートが可能です。キーは一般的な暗号化タスクや機能を実行するためにも使用できます。

- 対称暗号化アルゴリズムと非対称暗号化アルゴリズムの両方を使用して、暗号データ署名と署名検証を行います。
- ハッシュ関数を使用して、メッセージダイジェストと Hash-based Message Authentication Code (HMAC) を計算します。
- 他のキーをラップして保護します。
- 暗号化された安全なランダムデータにアクセスします。

クラスターが保持できる最大キーは、クラスター内の HSMsのタイプによって異なります。例えば、hsm2m.medium は hsm1,medium よりも多くのキーを保存します。これらの比較については、「AWS CloudHSM クォータ」を参照してください。

さらに、 AWS CloudHSM には、キーの使用と管理に関するいくつかの基本原則があります。

多くのキータイプとアルゴリズムから選択可能

独自のソリューションをカスタマイズできるように、 は、選択する多くのキータイプとアルゴリズム AWS CloudHSM を提供します。 アルゴリズムは、さまざまなキーサイズをサポートします。詳細については、それぞれ AWS CloudHSM クライアント SDKs の属性とメカニズムのページを参照してください。

キーの管理方法

AWS CloudHSM キーは SDKsとコマンドラインツールを使用して管理されます。これらのツールを使用してキーを管理する方法については、「でのキーの管理 AWS CloudHSM」と「のベストプラクティス AWS CloudHSM」を参照してください。

キーの所有者は誰か

では AWS CloudHSM、キーを作成する暗号ユーザー (CU) がキーを所有します。所有者は key share と key unshare コマンドを使用してキーを他の CU と共有または共有解除することができます。詳細については、「CloudHSM CLI を使用してキーを共有または共有解除します」を参照してください。

属性ベースの暗号化によりアクセスと使用を制御可能

AWS CloudHSM では、属性ベースの暗号化を使用できます。これは、キー属性を使用して、ポリシーに基づいてデータを復号できるユーザーを制御できる暗号化の一形式です。

Client SDK

を使用する場合 AWS CloudHSM、AWS CloudHSM クライアントソフトウェア開発キット (SDKsを使用して暗号化オペレーションを実行します。 AWS CloudHSM クライアント SDKsには以下が含まれます。

- 公開鍵暗号規格 #11 (PKCS #11)
- JCE プロバイダー
- OpenSSL Dynamic Engine
- 暗号化 API: Microsoft Windows 用の Next Generation (CNG) とキーストレージプロバイダー (KSP)

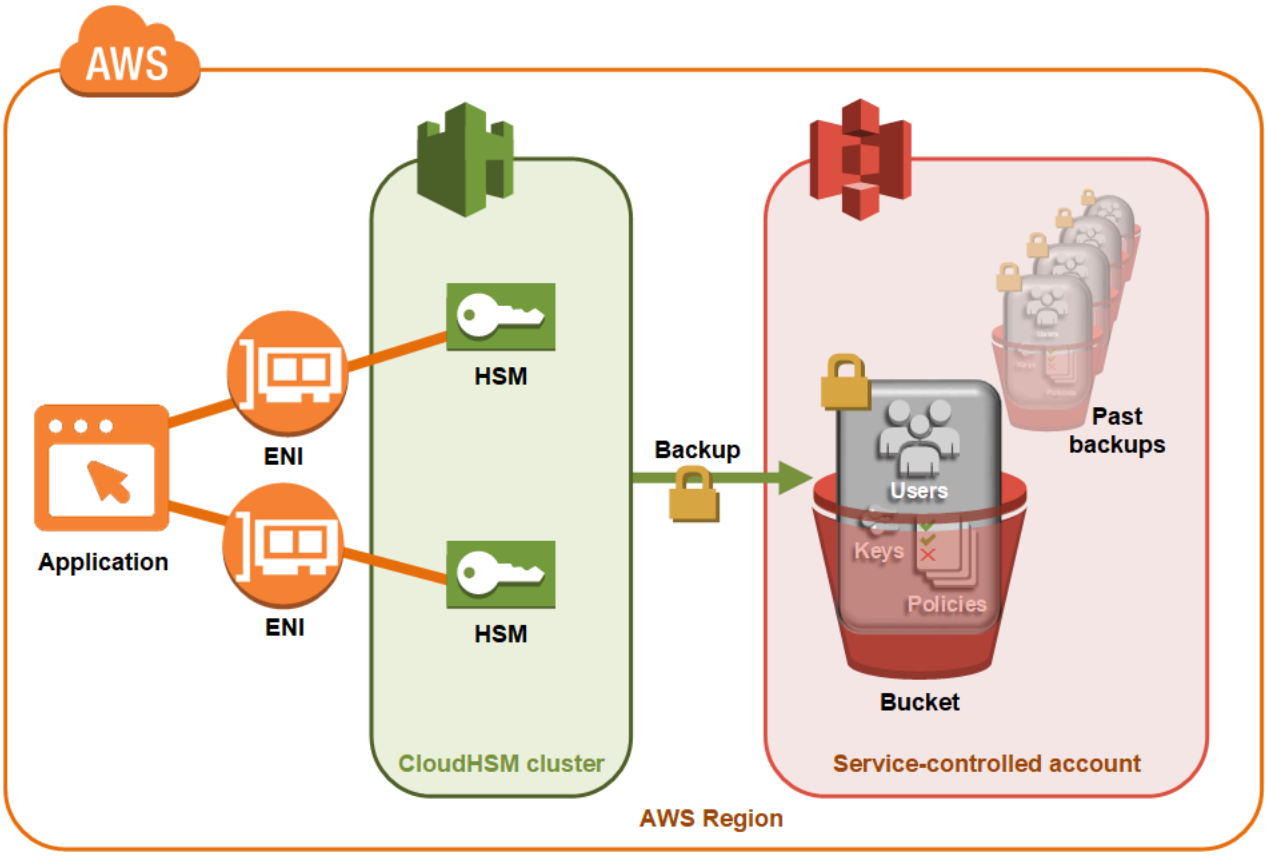
AWS CloudHSM クラスターでは、これらの SDK のいずれかまたはすべてを使用できます。これらの SDK を使用して HSM で暗号化操作を実行するアプリケーションコードを書き込みます。どのプラットフォームと HSM タイプが各 SDK をサポートしているかを確認するには、「」を参照してください。 Client SDK 5 がサポートするプラットフォーム

ユーティリティツールとコマンドラインツールは SDK を使用するだけでなく、アプリケーションの認証情報、ポリシー、設定を構成するためにも必要です。詳細については、「AWS CloudHSM コマンドラインツール」を参照してください。

クライアント SDK のインストールと使用、あるいはクライアント接続のセキュリティのさらなる詳細については、Client SDK と E nd-to-end 暗号化 を参照してください。

AWS CloudHSM クラスターバックアップ

AWS CloudHSM は、クラスター内のユーザー、キー、ポリシーを定期的にバックアップします。バックアップは安全で、耐久性が高くて、予測可能なスケジュールで更新されます。下の図は、バックアップとクラスターの関係を示しています。



ユーザーデータのさらなる操作方法の詳細については、バックアップの管理 を参照してください。

セキュリティ

AWS CloudHSM が HSM からバックアップを作成すると、HSM はすべてのデータを暗号化してから に送信します AWS CloudHSM。データがプレーンテキスト形式で HSM から外部に出ることはありません。さらに、 はバックアップの復号に使用されるキーにアクセス AWS できない

AWS ため、 によるバックアップの復号化はできません。詳細については、「クラスターバックアップのセキュリティ」を参照してください。

耐久性

AWS CloudHSM は、クラスターと同じリージョンのサービス制御の Amazon Simple Storage Service (Amazon S3) バケットにバックアップを保存します。バックアップの耐久レベルは 99.999999999% で、Amazon S3 に保存されているオブジェクトと同じです。

リージョン

でサポートされているリージョンの詳細については AWS CloudHSM、AWS CloudHSM 「」の「リージョンとエンドポイント」、または「リージョン表」を参照してください。 AWS 全般のリファレンス https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/

AWS CloudHSM は、特定のリージョンのすべてのアベイラビリティーゾーンで利用できるとは限りません。ただし、クラスター内のすべての HSMs 間で AWS CloudHSM 自動的に負荷分散されるため、パフォーマンスには影響しません。

ほとんどの AWS リソースと同様に、クラスターと HSMs はリージョンリソースです。クラスターをリージョン間で再利用したり、拡張したりすることはできません。「の開始方法 AWS CloudHSM」に一覧表示されている必要なステップをすべて実行し、新しいリージョンにクラスターを作成する必要があります。

ディザスタリカバリの目的で、 AWS CloudHSM では、 AWS CloudHSM クラスターのバックアップを 1 つのリージョンから別のリージョンにコピーできます。詳細については、「AWS CloudHSM クラスターバックアップ」を参照してください。

料金

では AWS CloudHSM、長期契約や前払いなしで時間単位で支払います。詳細については、 AWS ウェブサイトのAWS CloudHSM 「 の料金」を参照してください。

の開始方法 AWS CloudHSM

以下のトピックは、 AWS CloudHSM クラスターの作成、初期化、アクティブ化に役立ちます。これらの手順を完了したら、ユーザーやクラスターを管理できるほか、付属のソフトウェアライブラリを使用して、暗号化オペレーションを実行できるようになります。

内容

IAM 管理グループの作成

ベストプラクティスとして、 を使用して AWSを含む と AWS アカウントのルートユーザー やり取りしないでください AWS CloudHSM。代わりに、 AWS Identity and Access Management (IAM) を使用して IAM ユーザー、IAM ロール、またはフェデレーティッドユーザーを作成します。セクションの手順に従ってIAM ユーザーおよび管理者グループの作成管理者グループを作成し、AdministratorAccessポリシーをアタッチします。次に新しい管理者ユーザーを作成し、ユーザーをグループに追加します。必要に応じて、追加のユーザーをグループに追加します。追加した各ユーザーは、グループからAdministratorAccessポリシーを継承します。

もう 1 つのベストプラクティスは、 の実行に必要なアクセス許可のみを持つ AWS CloudHSM 管理者グループを作成することです AWS CloudHSM。必要に応じて個々のユーザーをこのグループに追

加します。 AWS へのフルアクセスではなく、グループにアタッチされた制限付きのアクセス許可が各ユーザーに継承されます。次のカスタマー管理ポリシー AWS CloudHSMセクションには、AWS CloudHSM 管理者グループにアタッチする必要があるポリシーが含まれています。

AWS CloudHSM は、 AWS アカウントのサービスにリンクされたロールを定義します。現在、サービスにリンクされたロールは、アカウントが AWS CloudHSM イベントをログ記録できるようにするアクセス許可を定義します。ロールは、ユーザーが自動 AWS CloudHSM または手動で作成できます。ロールを編集することはできませんが、削除することはできます。詳細については、「のサービスにリンクされたロール AWS CloudHSM」を参照してください。

IAM ユーザーおよび管理者グループの作成

IAM ユーザーと、その管理者グループの作成から開始します。

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインイン ユーザーガイド」の AWS 「 アクセスポータルへのサインイン」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

IAM ユーザーグループにアタッチ AWS CloudHSM できる のポリシーの例については、「」を参照してくださいの Identity and Access Management AWS CloudHSM。

仮想プライベートクラウド (VPC) の作成

まだ仮想プライベートクラウド (VPC) を持っていない場合は、このトピックのステップに従って、VPC を作成します。

> **Note**
>
> これらの手順に従うと、パブリックサブネットとプライベートサブネットが作成されます。

VPC を作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションバーで、リージョンセレクタを使用して、 AWSAWS CloudHSM が現在サポートされているリージョンのいずれかを選択します。
3. VPC を作成 ボタンを選択します。
4. [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. [名前タグの自動生成] に、「**`CloudHSM`**」などの識別可能な名前を入力します。
6. 他のすべてのオプションはデフォルト設定のままにします。
7. VPC を作成 を選択します。
8. VPC が作成されたら、VPC を表示 を選択して、先ほど作成した VPC を表示します。

クラスターを作成する

クラスターは、個々の HSMs。各クラスターの HSMs を AWS CloudHSM 同期して論理単位として機能します。 は、hsm1.medium と hsm2m.medium の HSMs AWS CloudHSM を提供します。クラスターを作成するときは、クラスターに含める 2 つの を選択します。各 HSM タイプとクラスターモードの違いの詳細については、「」を参照してくださいAWS CloudHSM クラスターモードとHSM タイプ。

クラスターを作成すると、 はユーザーに代わってクラスターのセキュリティグループ AWS CloudHSM を作成します。このセキュリティグループは、クラスター内の HSM へのネットワークアクセスを制御します。これにより、セキュリティグループの Amazon Elastic Compute Cloud (Amazon EC2) インスタンスからのみインバウンド接続ができます。デフォルトでは、セキュリティグループにインスタンスは一切含まれていません。後で、クライアントインスタンスを起動し、クラスターのセキュリティグループを設定して、HSM との接続および通信を可能にします。

> **Important**
>
> クラスターを作成すると、 は という名前のサービスにリンクされたロール AWS CloudHSM を作成します AWSServiceRoleForCloudHSM。がロールを作成 AWS CloudHSM できない場合、またはロールがまだ存在しない場合は、クラスターを作成できない可能性があります。詳細については、「クラスター作成エラーの解決」を参照してください。サービスにリンクされたロールの詳細については、「のサービスにリンクされたロール AWS CloudHSM」を参照してください。

AWS CloudHSM コンソール、AWS Command Line Interface (AWS CLI)、 AWS CloudHSM API からクラスターを作成できます。

> **Note**
>
> クラスター引数と APIs「」を参照してください。 create-cluster

クラスターを作成するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. ナビゲーションバーで、リージョンセレクタを使用して、 AWSAWS CloudHSM が現在サポートされているリージョンのいずれかを選択します。

3. [クラスターを作成] を選択します。
4. [Cluster configuration] セクションで、以下の操作を実行します。

   a. VPC では、仮想プライベートクラウド (VPC) の作成 で作成した VPC を選択します。

   b. Availability Zone(s) では、各アベイラビリティーゾーンの横にある、作成したプライベートサブネットを選択します。

   > **Note**
   >
   > AWS CloudHSM が特定のアベイラビリティーゾーンでサポートされていない場合でも、クラスター内のすべての HSMs 間で AWS CloudHSM 自動的に負荷分散されるため、パフォーマンスには影響しません。AWS CloudHSM のアベイラビリティーゾーンのサポートについては、「」の「リージョンとエンドポイント」を参照してください AWS CloudHSM。 AWS 全般のリファレンス

   c. HSM タイプ では、クラスターで作成できる HSM タイプと、クラスターの目的のモードを選択します。各リージョンでサポートされている HSM タイプについては、AWS CloudHSM 料金計算ツール をご覧ください。

   > **Important**
   >
   > クラスターの作成後、HSM タイプとクラスターモードを変更することはできません。ユースケースに適したタイプとモードについては、「」を参照してくださいAWS CloudHSM クラスターモードと HSM タイプ。

   d. クラスターソース で、新しいクラスターを作成するか、既存のバックアップから復元するかを指定します。

   - 非 FIPS モードのクラスターのバックアップは、非 FIPS モードのクラスターの復元にのみ使用できます。
   - FIPS モードのクラスターのバックアップは、FIPS モードのクラスターの復元にのみ使用できます。

5. [次へ] をクリックします。
6. サービスがバックアップを保持する期間を指定します。

> **Note**
>
> デフォルトの保存期間である 90 日を受け入れるか、7 ~ 379 日の間に新しい値を入力します。このサービスは、ここで指定した値よりも古いこのクラスター内のバックアップを自動的に削除します。これは後で変更できます。詳細については、「バックアップの保持を設定」を参照してください。

7. [Next] (次へ) を選択します。
8. (オプション) タグキーとオプションのタグ値を入力します。クラスターに複数のタグを追加するには、タグの追加 を選択します。
9. [Review] (レビュー) を選択します。
10. クラスター設定を確認し、[Create cluster (クラスターの作成)] を選択します。

クラスターを作成するには (AWS CLI)

- コマンドラインプロンプトで、create-cluster コマンドを実行します。HSM インスタンスタイプ、バックアップ保持期間、HSM を作成するサブネットのサブネット ID を指定します。作成したプライベートサブネットのサブネット ID を使用します。サブネットは、アベイラビリティーゾーンごとに 1 つだけ指定できます。

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm1.medium \
      --backup-retention-policy Type=DAYS,Value=<number of days> \
      --subnet-ids <subnet ID>

{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
         },
        "VpcId": "vpc-50ae0636",
        "SubnetMapping": {
            "us-west-2b": "subnet-49a1bc00",
            "us-west-2c": "subnet-6f950334",
            "us-west-2a": "subnet-fd54af9b"
        },
        "SecurityGroup": "sg-6cb2c216",
```

```
        "HsmType": "hsm1.medium",
        "Certificates": {},
        "State": "CREATE_IN_PROGRESS",
        "Hsms": [],
        "ClusterId": "cluster-igklspoyj5v",
        "ClusterMode": "FIPS",
        "CreateTimestamp": 1502423370.069
    }
}
```

> **Note**
>
> `ClusterMode` が指定されていない場合、 は FIPS モードにデフォルト設定されます。非 FIPS クラスターを作成するには、パラメータ を含める必要があります`--mode`。
>
> ```
> $ aws cloudhsmv2 create-cluster --hsm-type hsm2m.medium \
>       --backup-retention-policy Type=DAYS,Value=<number of days> \
>       --subnet-ids <subnet ID> \
>     --mode NON_FIPS
> ```

クラスターを作成するには (AWS CloudHSM API)

- CreateCluster リクエストを送信します。HSM インスタンスタイプ、バックアップ保持ポリシー、HSM を作成するサブネットのサブネット ID を指定します。作成したプライベートサブネットのサブネット ID を使用します。サブネットは、アベイラビリティーゾーンごとに 1 つだけ指定できます。

クラスターの作成に失敗した場合は、 AWS CloudHSM のサービスにリンクされたロールの問題に関連している可能性があります。障害を解決するためのヘルプについては、「クラスター作成エラーの解決」を参照してください。

クラスターセキュリティグループの確認

クラスターを作成すると、 は という名前のセキュリティグループ AWS CloudHSM を作成しますcloudhsm-cluster-*clusterID*-sg。このセキュリティグループには、ポート 2223～2225 経由のインバウンド通信とアウトバウンド通信を許可する事前に設定された TCP ルールが含まれます。この SG により、EC2 インスタンスは VPC を使用してクラスター内の HSM と通信できます。

Warning

- 事前設定された TCP ルールを削除または変更しないでください。このルールは、クラスターセキュリティグループに追加されています。このルールによって、接続の問題と HSM への不正アクセスを防ぐことができます。
- クラスターのセキュリティグループに追加することで、HSM への不正アクセスを防ぐことができます。セキュリティグループ内のインスタンスにアクセスできるユーザーはいずれも、HSM にアクセスできます。ほとんどのオペレーションでは、ユーザーは HSM にログインする必要があります。ただし、認証せずに HSM をゼロ化することもできます。ゼロ化すると、キーマテリアル、証明書などのデータは破棄されます。ゼロ化した場合、最後にバックアップしてから作成または変更したデータは失われ、復旧することはできません。不正アクセスを防ぐために、デフォルトのセキュリティグループのインスタンスの変更またはアクセスは、信頼されている管理者のみ行うことができることを確認します。

次のステップでは、Amazon EC2 インスタンスを起動 し、そのインスタンスに クラスターセキュリティグループを接続 して、HSM に接続します。

Amazon EC2 クライアントインスタンスの起動

AWS CloudHSM クラスターと HSM インスタンスを操作して管理するには、HSM の Elastic Network Interface HSMs と通信できる必要があります。これを行う最も簡単な方法として、同じ VPC 内の EC2 インスタンスをクラスターとして使用できます。以下の AWS リソースを使用してクラスターに接続することもできます。

- Amazon VPC ピアリング
- AWS Direct Connect
- VPN 接続

Note

このガイドでは、EC2 インスタンスを AWS CloudHSM クラスターに接続する方法の簡単な例を示します。安全なネットワーク設定に関するベストプラクティスについては、「」を参照してくださいクラスターへの安全なアクセス。

AWS CloudHSM ドキュメントでは、通常、クラスターを作成するのと同じ VPC とアベイラビリティーゾーン (AZ) で EC2 インスタンスを使用していることを前提としています。

EC2 インスタンスを作成するには

1. https://console.aws.amazon.com/ec2/ で EC2 ダッシュボード を開きます。
2. [Launch instance] (インスタンスを起動) を選択します。ドロップダウンメニューから、[インスタンスの起動] を選択します。
3. [名前] フィールドに、EC2 インスタンスの名前を入力します。
4. [アプリケーションおよび OS イメージ (Amazon マシンイメージ)] セクションで、CloudHSM がサポートするプラットフォームに対応する Amazon マシンイメージ (AMI) を選択します。詳細については、「Client SDK 5 がサポートするプラットフォーム」を参照してください。
5. インスタンスタイプ] セクションで、インスタンスタイプを選択します。
6. [キーペア] セクションで、既存のキーペアを使用するか、[新しいキーペアの作成] を選択して次の手順を実行します。
   a. [キーペア名] に、新しいキーペアの名前を入力します。
   b. [キーペアタイプ] で、キーペアタイプを選択します。
   c. [プライベートキーファイル形式] で、プライベートキーファイルの形式を選択します。
   d. [Create key pair] (キーペアを作成) を選択します。
   e. プライベートキーファイルをダウンロードして保存します。

   > **Important**
   > このタイミングでのみ、プライベートキーファイルを保存できます。ダウンロードしたファイルを安全な場所に保存します。インスタンスの起動時に、キーペアの名前を指定する必要があります。さらに、インスタンスに接続するたびに対応するプライベートキーを提供し、セットアップ時に作成したキーペアを選択する必要があります。
7. [ネットワーク設定] で [編集] を選択します。
8. VPC には、クラスター用に以前に作成した VPC を選択します。
9. [サブネット] で、VPC で以前に作成したパブリックサブネットを選択します。
10. [Auto-assign Public IP] (パブリック IP の自動割当て) で、[Enable] (有効化) を選択します。
11. [Select an existing security group (既存のセキュリティグループの選択)] を選択します。

12. [共通セキュリティグループ] では、ドロップダウンメニューからデフォルトのセキュリティグループを選択します。
13. ストレージの設定 では、ドロップダウンメニューを使用してストレージ設定を選択します。
14. 「概要」ウィンドウで「インスタンスを起動」を選択します。

> **Note**
>
> このステップを完了すると、EC2 インスタンスの作成プロセスが開始されます。

Linux Amazon EC2 クライアントを作成する方法の詳細については、Amazon EC2 Linux インスタンスの開始方法 を参照してください。実行中のクライアントへの接続については、次のトピックを参照してください。

- SSH を使用した Linux インスタンスへの接続
- PuTTY を使用した Windows から Linux インスタンスへの接続

Amazon EC2 ユーザーガイドは、Amazon EC2 インスタンスを設定および使用するための詳細な手順が含まれます。次のリストは、Linux と Windows Amazon EC2 クライアントで使用できるドキュメントの概要を示します。

- Linux Amazon EC2 クライアントを作成するには、Amazon EC2 Linux インスタンスの開始方法 を参照してください。

  実行中のクライアントへの接続については、次のトピックを参照してください。

  - SSH を使用した Linux インスタンスへの接続
  - PuTTY を使用した Windows から Linux インスタンスへの接続
- Windows Amazon EC2 クライアントを作成するには、Amazon EC2 Windows インスタンスの開始方法 を参照してください。Windows クライアントに接続する方法については、「Windows インスタンスへの接続」を参照してください。

> **Note**
>
> EC2 インスタンスは、このガイドに含まれるすべての AWS CLI コマンドを実行できます。AWS CLI がインストールされていない場合は、「AWS Command Line Interface」からダウンロードすることができます。Windows を使用している場合は、64 ビットまたは 32 ビット

の Windows インストーラをダウンロードして実行できます。Linux または macOS を使用している場合は、pip を使用して CLI をインストールできます。

クライアントの Amazon EC2 インスタンスのセキュリティグループを設定する

Amazon EC2 インスタンスを起動した際、デフォルトの Amazon VPC セキュリティグループに関連付けました。このトピックでは、クラスターセキュリティグループを EC2 インスタンスに関連付ける方法について説明します。この関連付けにより、EC2 インスタンスで実行されている AWS CloudHSM クライアントが HSMsと通信できるようになります。EC2 インスタンスを AWS CloudHSM クラスターに接続するには、VPC のデフォルトのセキュリティグループを適切に設定し、クラスターのセキュリティグループをインスタンスに関連付ける必要があります。

デフォルトのセキュリティグループの変更

クライアントソフトウェアをダウンロードしてインストールし、HSM と通信できるように、SSH 接続または RDP 接続が許可されるようにデフォルトのセキュリティグループを変更する必要があります。

デフォルトのセキュリティグループを変更するには

1. https://console.aws.amazon.com/ec2/ で EC2 ダッシュボード を開きます。
2. インスタンス (実行中) を選択し、 AWS CloudHSM クライアントをインストールする EC2 インスタンスの横にあるチェックボックスをオンにします。
3. [セキュリティ] タブで、[デフォルト] という名前のセキュリティグループを選択します。
4. ページの一番上で、[アクション]、[インバウンドのルールの編集] の順に選択します。
5. [Add rule (ルールの追加)] を選択します。
6. [タイプ] で、以下のいずれかを実行します。
   - Windows Server の Amazon EC2 インスタンスで、RDP を選択します。ポート 3389 は自動的に追加されています。
   - Linux Amazon EC2 インスタンスの場合は、SSH を選択します。ポート範囲 22 は自動的に追加されています。
7. いずれのオプションでも、[ソース] を [My IP] に設定すると、Amazon EC2 インスタンスと通信できるようになります。

> **Important**
>
> 誰もがインスタンスにアクセスできないようにするには、CIDR 範囲として 0.0.0.0/0 を指定しないでください。

8. [保存] を選択します。

Amazon EC2 インスタンスを AWS CloudHSM クラスターに接続する

EC2 インスタンスがクラスター内の HSM と通信できるように、クラスターのセキュリティグループを EC2 インスタンスに接続する必要があります。クラスターのセキュリティグループには、ポート 2223〜2225 経由のインバウンド通信を許可する事前に設定されたルールが含まれます。

EC2 インスタンスを AWS CloudHSM クラスターに接続するには

1. https://console.aws.amazon.com/ec2/ で EC2 ダッシュボード を開きます。
2. インスタンス (実行中) を選択し、 AWS CloudHSM クライアントをインストールする EC2 インスタンスのチェックボックスをオンにします。
3. ページの一番上で、アクション、セキュリティ、セキュリティグループの変更 を選択します。
4. クラスターの ID と一致するグループ名のセキュリティグループ (例: `cloudhsm-cluster-`*`clusterID`*`-sg`) を選択します。
5. [セキュリティグループを追加] を選択します。
6. [Save] を選択します。

> **Note**
>
> 最大 5 つのセキュリティグループを 1 つの Amazon EC2 インスタンスに割り当てることができます。上限に達した場合は、Amazon EC2 のインスタンスのデフォルトのセキュリティグループとクラスターのセキュリティグループを変更する必要があります。
>
> デフォルトのセキュリティグループで、以下の操作を行います。
>
> - クラスターセキュリティグループからポート 2223-2225 経由で TCP プロトコルを使用したトラフィックを許可するアウトバウンドルールを追加します。

クラスターのセキュリティグループで、以下の操作を行います。

- デフォルトのセキュリティグループからポート 2223-2225 経由で TCP プロトコルを使用したトラフィックを許可するインバウンドルールを追加します。

HSM を作成する

クラスターの作成後、HSM を作成することができます。ただし、クラスター内に HSM を作成する前に、クラスターを初期化されていない状態にする必要があります。クラスターの状態を確認するには、コンソール で AWS CloudHSM クラスターページを表示するか、 を使用して describe-clusters コマンド AWS CLI を実行するか、 AWS CloudHSM API でDescribeClustersリクエストを送信します。HSM は、AWS CloudHSM コンソール、AWS CLI、あるいは AWS CloudHSM API で作成できます。

HSM を作成するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. HSM を作成するクラスターの ID の横のラジオボタンを選択します。
3. アクション を選択します。ドロップダウンメニューから 初期化 を選択します。
4. 作成中の HSM のアベイラビリティーゾーン (AZ) を選択します。
5. [作成] を選択します。

HSM を作成するには (AWS CLI)

- コマンドラインプロンプトで、create-hsm コマンドを実行します。以前に作成したクラスターのクラスター ID と、HSM のアベイラビリティーゾーンを指定します。アベイラビリティーゾーンは、`us-west-2a`、`us-west-2b` などの形式で指定します。

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-
zone <Availability Zone>

{
    "Hsm": {
        "HsmId": "hsm-ted36yp5b2x",
        "EniIp": "10.0.1.12",
```

```
        "AvailabilityZone": "us-west-2a",
        "ClusterId": "cluster-igklspoyj5v",
        "EniId": "eni-5d7ade72",
        "SubnetId": "subnet-fd54af9b",
        "State": "CREATE_IN_PROGRESS"
    }
}
```

HSM (AWS CloudHSM API) を作成するには

- CreateHsm リクエストを送信します。以前に作成したクラスターのクラスター ID と、HSM のアベイラビリティーゾーンを指定します。

クラスターと HSM を作成した後、オプションで HSM のアイデンティティを確認するか、直接「クラスターの初期化」に進むことができます。

クラスターの HSM のアイデンティティと正当性の確認 (オプション)

クラスターを初期化するには、クラスターの最初の HSM によって生成された証明書署名リクエスト (CSR) に署名する必要があります。その前に、HSM のアイデンティティと正当性を確認することをお勧めします。

> **Note**
>
> このプロセスはオプションです。ただし、使用できるのはクラスターが初期化される前に限られます。クラスターの初期化後は、このプロセスを使用して証明書の取得や HSM の検証を行うことはできません。

トピック

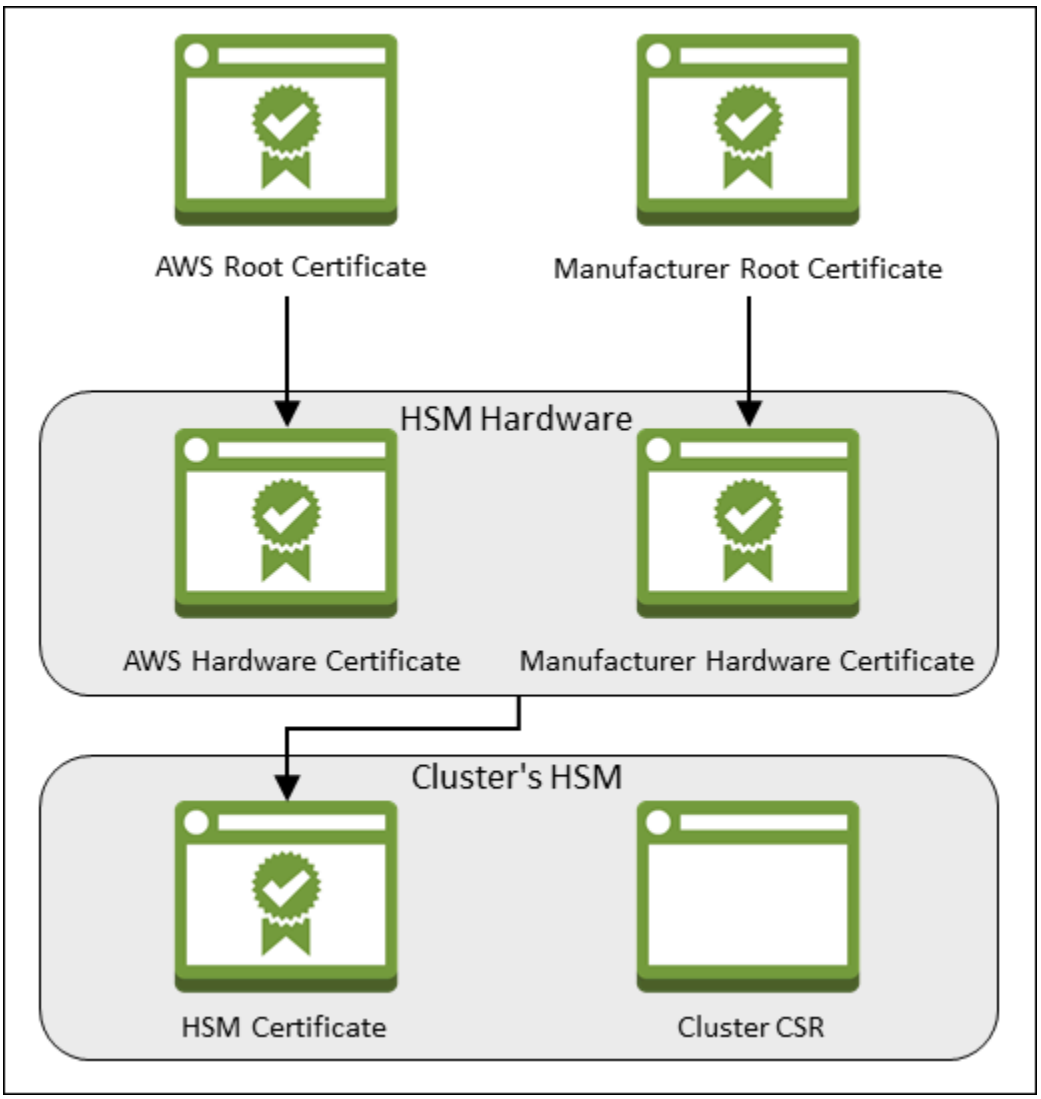
- 概要
- HSM からの証明書の取得
- ルート証明書の取得
- 証明書チェーンの確認
- パブリックキーの抽出と比較

概要

クラスターの最初の HSM のアイデンティティを確認するには、次のステップを実行します。

1. 証明書と CSR の取得 - このステップでは、3 つの証明書と CSR を HSM から取得します。また、2 つのルート証明書も取得 AWS CloudHSM します。1 つは から、もう 1 つは HSM ハードウェアメーカーから取得します。
2. 証明書チェーンの検証 – このステップでは、ルート証明書と製造元 AWS CloudHSM ルート証明書の 2 つの証明書チェーンを作成します。次に、これらの証明書チェーンで HSM 証明書を検証して、 AWS CloudHSM とハードウェア製造元の両方が HSM のアイデンティティと信頼性を証明しているかどうかを確認します。
3. パブリックキーの比較 - このステップでは、HSM 証明書とクラスター CSR のパブリックキーを抽出および比較し、それらが同じであることを確認します。これにより、CSR は認証され、信頼された HSM によって生成されていることを確信できます。

次の図は、CSR、証明書、およびその相互関係を示しています。以下のリストでは、各証明書を定義します。



AWS ルート証明書

これは AWS CloudHSMのルート証明書です。

製造元のルート証明書

これは、ハードウェア製造元のルート証明書です。

AWS ハードウェア証明書

AWS CloudHSM HSM ハードウェアがフリートに追加されると、 はこの証明書を作成しました。この証明書は、ハードウェア AWS CloudHSM を所有する をアサートします。

製造元のハードウェア証明書

この証明書は、HSM ハードウェア製造元が HSM ハードウェアを製造したときに作成したものです。この証明書は、製造元がハードウェアを作成したことを主張しています。

HSM 証明書

クラスターの最初の HSM を作成すると、FIPS 検証済みハードウェアが HSM 証明書を生成します。この証明書は、HSM ハードウェアが HSM の作成元であることを証明します。

クラスター CSR

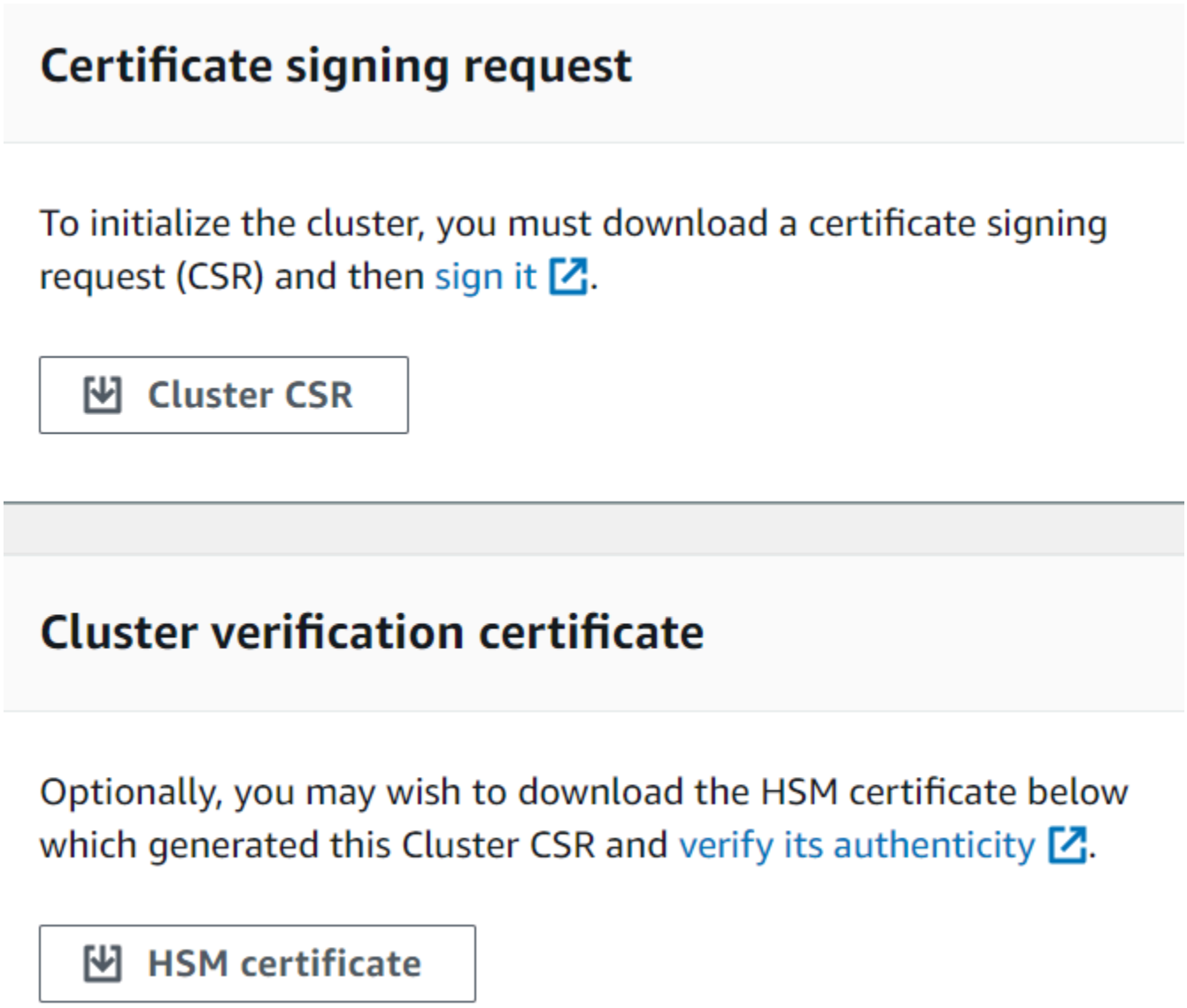
最初の HSM によってクラスター CSR が作成されます。クラスター CSR に署名する と、クラスターがクレームされます。次に、署名した CSR を使用して クラスターを初期化 できます。

HSM からの証明書の取得

HSM のアイデンティティと正当性を確認するには、最初に CSR と 5 つの証明書を取得します。HSM から 3 つの証明書を取得します。これは、AWS CloudHSM コンソール 、 ()、または API で実行できます。 AWS Command Line InterfaceAWS CLI AWS CloudHSM

CSR および HSM 証明書を取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. 検証する HSM のクラスター ID の横にあるラジオボタンをオンにします。
3. アクション を選択します。ドロップダウンメニューから 初期化 を選択します。
4. HSM を作成する 前のステップ を完了していない場合は、作成する HSM のアベイラビリティーゾーン (AZ) を選択します。次に、作成 を選択します。
5. 証明書と CSR の準備ができると、それらをダウンロードするためのリンクが表示されます。



6. 各リンクを選択し、CSR と証明書をダウンロードして保存します。以降のステップを簡素化するために、すべてのファイルを同じディレクトリに保存し、デフォルトのファイル名を使用します。

CSR および HSM 証明書を取得するには (AWS CLI)

- コマンドプロンプトで、describe-clusters コマンドを 4 回実行し、毎回 CSR と異なる証明書を抽出してファイルに保存します。

  a. 次のコマンドを発行してクラスター CSR を抽出します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query 'Clusters[].Certificates.ClusterCsr'
 \
                                   > <cluster ID>_ClusterCsr.csr
```

b. 次のコマンドを発行して HSM 証明書を抽出します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.HsmCertificate' \
                                   > <cluster ID>_HsmCertificate.crt
```

c. 次のコマンドを実行して、 AWS ハードウェア証明書を抽出します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.AwsHardwareCertificate' \
                                   > <cluster ID>_AwsHardwareCertificate.crt
```

d. 次のコマンドを発行して製造元のハードウェア証明書を抽出します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.ManufacturerHardwareCertificate' \
                                   > <cluster
 ID>_ManufacturerHardwareCertificate.crt
```

CSR 証明書と HSM 証明書を取得するには (AWS CloudHSM API)

- DescribeClusters リクエストを送信し、レスポンスから CSR と証明書を抽出して保存します。

ルート証明書の取得

AWS CloudHSM および製造元のルート証明書を取得するには、次の手順に従います。ルート証明書ファイルを、CSR と HSM 証明書ファイルが含まれているディレクトリに保存します。

AWS CloudHSM および製造元のルート証明書を取得するには

1. AWS CloudHSM ルート証明書をダウンロードします: AWS_CloudHSM _Root-G1.zip
2. HSM タイプに適した製造元のルート証明書をダウンロードします。
   - hsm1.medium 製造元ルート証明書: Liquid_security_certificate.zip
   - hsm2m.medium 製造元ルート証明書: Liquid_security_certificate.zip

   > **Note**
   >
   > ランディングページから各証明書をダウンロードするには、次のリンクを使用します。
   >
   > - hsm1.medium の製造元ルート証明書のランディングページ
   > - hsm2m.medium の製造元ルート証明書のランディングページ
   >
   > [Download Certificate] リンクを右クリックしてから、[Save Link As...] を選択して証明書ファイルを保存することが必要になる場合があります。
3. ファイルをダウンロードした後、内容を抽出 (解凍) します。

証明書チェーンの確認

このステップでは、2 つの証明書チェーンを構築します。1 つは AWS CloudHSM ルート証明書、もう 1 つは製造元ルート証明書です。次に、OpenSSL を使用して各証明書チェーンの HSM 証明書を検証します。

証明書チェーンを作成するには、Linux シェルを開きます。OpenSSL が必要です (ほとんどの Linux シェルにあります)。さらに、ダウンロードしたルート証明書と HSM 証明書ファイルが必要です。ただし、このステップ AWS CLI では は必要なく、シェルを AWS アカウントに関連付ける必要はありません。

HSM 証明書を AWS CloudHSM ルート証明書で検証するには

1. ダウンロードしたルート証明書と HSM 証明書ファイルの保存先のディレクトリに移動します。以下のコマンドでは、すべての証明書が現在のディレクトリにあり、デフォルトのファイル名を使用しているものとします。

   次のコマンドを使用して、ハードウェア証明書と AWS CloudHSM ルート証明書をその順序で含む AWS 証明書チェーンを作成します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

   ```
   $ cat <cluster ID>_AwsHardwareCertificate.crt \
         AWS_CloudHSM_Root-G1.crt \
         > <cluster ID>_AWS_chain.crt
   ```

2. AWS 証明書チェーンで HSM 証明書を検証するには、次の OpenSSL コマンドを使用します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

   ```
   $ openssl verify -CAfile <cluster ID>_AWS_chain.crt <cluster ID>_HsmCertificate.crt
   <cluster ID>_HsmCertificate.crt: OK
   ```

製造元のルート証明書で HSM 証明書を検証するには

1. 次のコマンドを使用して、製造元のハードウェア証明書と、製造元のルート証明書が含まれている証明書チェーンを、その順番で作成します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

   ```
   $ cat <cluster ID>_ManufacturerHardwareCertificate.crt \
         liquid_security_certificate.crt \
         > <cluster ID>_manufacturer_chain.crt
   ```

2. 製造元の証明書チェーンで HSM 証明書を検証するには、次の OpenSSL コマンドを使用します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

   ```
   $ openssl verify -CAfile <cluster ID>_manufacturer_chain.crt <cluster
    ID>_HsmCertificate.crt
   <cluster ID>_HsmCertificate.crt: OK
   ```

パブリックキーの抽出と比較

OpenSSL を使用して HSM 証明書とクラスター CSR のパブリックキーを抽出および比較して、それらが同じであることを確認します。

パブリックキーを比較するには、Linux シェルを使用します。ほとんどの Linux シェルで使用できる OpenSSL が必要ですが、このステップ AWS CLI では は必要ありません。シェルを AWS アカウントに関連付ける必要はありません。

パブリックキーを抽出して比較するには

1. 次のコマンドを使用して、HSM 証明書からパブリックキーを抽出します。

```
$ openssl x509 -in <cluster ID>_HsmCertificate.crt -pubkey -noout > <cluster
 ID>_HsmCertificate.pub
```

2. 次のコマンドを使用して、クラスター CSR からパブリックキーを抽出します。

```
$ openssl req -in <cluster ID>_ClusterCsr.csr -pubkey -noout > <cluster
 ID>_ClusterCsr.pub
```

3. 次のコマンドを使用してパブリックキーを比較します。パブリックキーが同じである場合、次のコマンドによる出力はありません。

```
$ diff <cluster ID>_HsmCertificate.pub <cluster ID>_ClusterCsr.pub
```

HSM のアイデンティティと正当性を確認したら、「クラスターの初期化」に進みます。

クラスターの初期化

AWS CloudHSM クラスターを初期化するには、以下のトピックのステップを実行します。

> **Note**
>
> クラスターを初期化する前に、HSM のアイデンティティと正当性を検証するプロセスを確認します。このプロセスはオプションですが、使用期間はクラスターが初期化されるまでの間に限ります。クラスターの初期化後は、このプロセスを使用して証明書の取得や HSM の検証を行うことはできません。

トピック

- クラスター CSR の取得
- CSR の署名
- クラスターの初期化

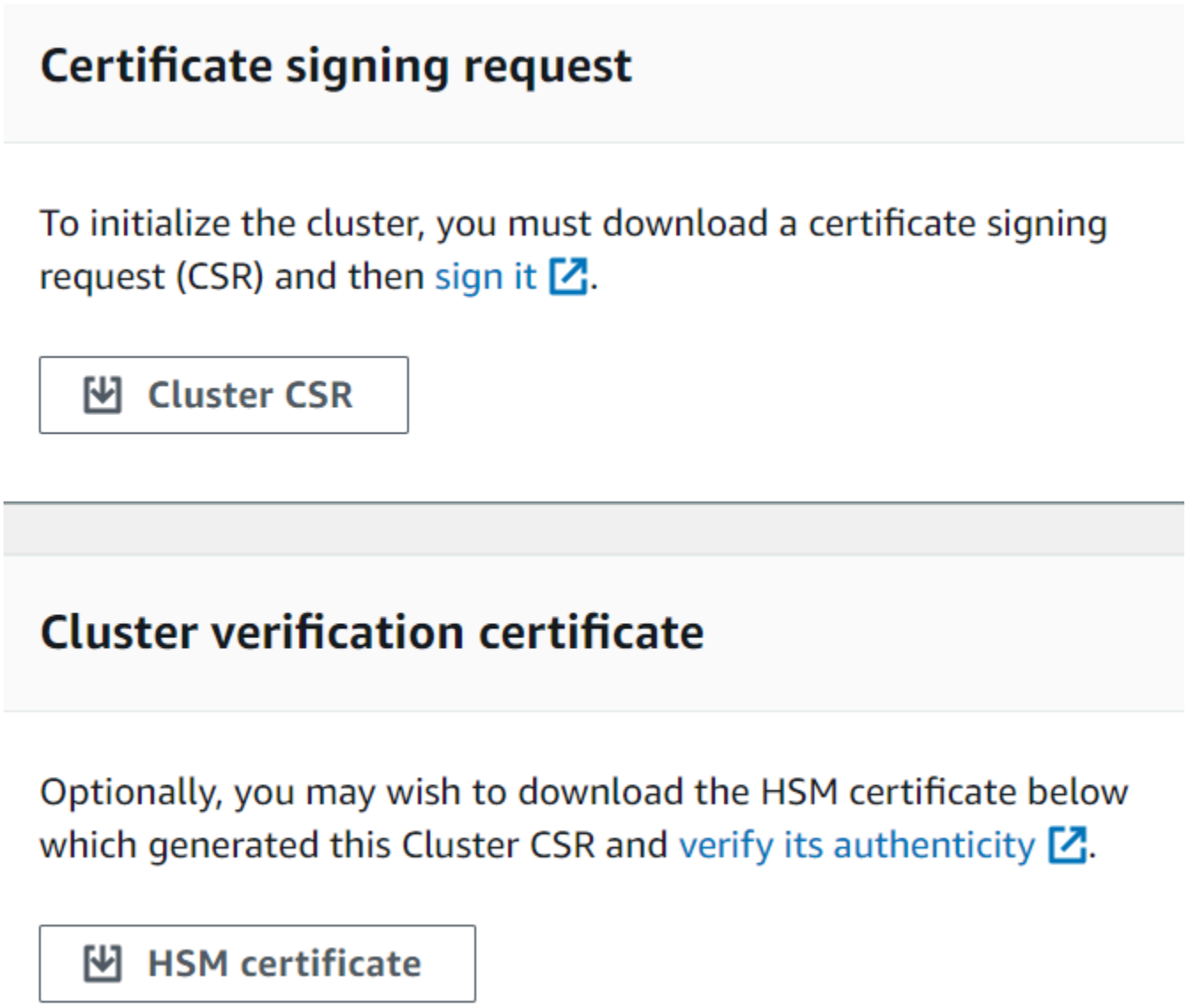
クラスター CSR の取得

クラスターを初期化する前に、クラスターの最初の HSM によって生成された証明書署名リクエスト (CSR) をダウンロードして署名する必要があります。「クラスターの HSM のアイデンティティの確認」のステップに従っていれば、既に CSR があり、「CSR の署名」に進むことができます。それ以外の場合は、AWS CloudHSM コンソール 、 ()、または API を使用して CSR を取得します。AWS Command Line InterfaceAWS CLI AWS CloudHSM

> **Important**
>
> クラスターを初期化するには、トラストアンカーが RFC 5280 に準拠し、次の要件を満たしている必要があります。
>
> - X509v3 エクステンションを使用する場合は、X509v3 基本制約エクステンションが必要です。
> - トラストアンカーは自己署名証明書でなければなりません。
> - エクステンションの値は互いに矛盾してはいけません。

CSR を取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. 検証する HSM のクラスター ID の横にあるラジオボタンをオンにします。
3. アクション を選択します。ドロップダウンメニューから 初期化 を選択します。
4. HSM を作成する 前のステップ を完了していない場合は、作成する HSM のアベイラビリティーゾーン (AZ) を選択します。次に、作成 を選択します。
5. CSR の準備ができると、CSR をダウンロードするためのリンクが表示されます。



6. [Cluster CSR] を選択して CSR をダウンロードし、保存します。

CSR を取得するには (AWS CLI)

- コマンドプロンプトで、次の describe-clusters コマンドを実行し、CSR を抽出してファイルに保存します。*<cluster ID>* を、前に作成したクラスターの ID に置き換えます。

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query 'Clusters[].Certificates.ClusterCsr' \
                                   > <cluster ID>_ClusterCsr.csr
```

CSR (AWS CloudHSM API) を取得するには

1. DescribeClusters リクエストを送信します。
2. レスポンスから CSR を抽出して保存します。

CSR の署名

現在、自己署名入りの署名証明書を作成し、これを使用してクラスターの CSR に署名する必要があります。このステップ AWS CLI では は必要なく、シェルを AWS アカウントに関連付ける必要はありません。CSR に署名するには、以下を行う必要があります。

1. 前のセクションを完了してください (「クラスター CSR の取得」を参照)。
2. プライベートキーを作成します。
3. プライベートキー を使用して、署名証明書を作成します。
4. クラスター CSR の署名。

プライベートキーを作成する

> **Note**
>
> 実稼働用クラスターでは、作成しようとしているキーはランダム性の信頼できるソースを使用して安全な方法で作成されている必要があります。安全なオフサイトあるいはオフライン HSM またはその同等を使用することが推奨されます。キーを安全に保存します。このキーによってクラスターの ID が確立され、クラスターに含まれる HSM をユーザーが単独で制御できるようになります。
> 開発とテストには、クラスター証明書の作成と署名に任意のツール (OpenSSL) などを使用できます。以下の例では、 キーを作成する方法を示します。キーを使用して自己署名証明書 (以下を参照) を作成したら、安全な方法でキーを保存する必要があります。 AWS CloudHSM インスタンスにサインインするには、証明書が存在する必要がありますが、プライベートキーは存在しません。

次のコマンドを使用して、プライベートキーを作成します。 AWS CloudHSM クラスターを初期化するときは、RSA 2048 証明書または RSA 4096 証明書を使用する必要があります。

```
$ openssl genrsa -aes256 -out customerCA.key 2048
```

```
Generating RSA private key, 2048 bit long modulus
........+++
............+++
e is 65537 (0x10001)
Enter pass phrase for customerCA.key:
Verifying - Enter pass phrase for customerCA.key:
```

プライベートキーを使用して、自己署名証明書を作成します。

本稼働のクラスター用のプライベートキーの作成に使用する信頼できるハードウェアも、このキーを使用した自己署名証明書を生成するソフトウェアツールを提供していることが必要です。次の例では、OpenSSL および署名証明書を作成するために前のステップで作成したプライベートキーを使用しています。証明書の有効期間は 10 年 (3652 日) です。画面の指示を読み、プロンプトに従います。

```
$ openssl req -new -x509 -days 3652 -key customerCA.key -out customerCA.crt
Enter pass phrase for customerCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

このコマンドは customerCA.crt という名前の証明書を作成します。この証明書は、 AWS CloudHSM クラスターに接続するすべてのホストに配置します。ファイルに別の名前を付与した場合、またはホストのルート以外のパスにファイルを保存した場合には、それぞれに応じてクライアント設定ファイルを編集する必要があります。作成したばかりの証明書およびプライベートキーを使用して、次のステップでクラスター証明書署名リクエスト (CSR) に署名します。

クラスターの CSR に署名する

本稼働のクラスター用のプライベートキーの作成に使用する信頼できるハードウェアも、このキーを使用して CSR に署名するツールを提供している必要があります。次の例では、OpenSSL を使用してクラスターの CSR に署名します。この例では、プライベートキーと前のステップで作成した自己署名証明書を使用します。

```
$ openssl x509 -req -days 3652 -in <cluster ID>_ClusterCsr.csr \
                               -CA customerCA.crt \
                               -CAkey customerCA.key \
                               -CAcreateserial \
                               -out <cluster ID>_CustomerHsmCertificate.crt
Signature ok
subject=/C=US/ST=CA/O=Cavium/OU=N3FIPS/L=SanJose/CN=HSM:<HSM
 identifer>:PARTN:<partition number>, for FIPS mode
Getting CA Private Key
Enter pass phrase for customerCA.key:
```

完了すると、このコマンドは *<cluster ID>*_CustomerHsmCertificate.crt という名前のファイルを作成します。クラスターを初期化する際は、これを署名済み証明書として使用します。

クラスターの初期化

署名済み HSM 証明書と署名証明書を使用して、クラスターを初期化します。AWS CloudHSM コンソール 、、AWS CLIまたは AWS CloudHSM API を使用できます。

クラスターを初期化するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. 検証する HSM のクラスター ID の横にあるラジオボタンをオンにします。
3. アクション を選択します。ドロップダウンメニューから 初期化 を選択します。
4. HSM を作成する 前のステップ を完了していない場合は、作成する HSM のアベイラビリティーゾーン (AZ) を選択します。次に、作成 を選択します。
5. [Download certificate signing request] ページで、[Next] を選択します。[Next] が利用できない場合は、最初にいずれかの CSR または証明書のリンクを選択します。次いで、[次へ] を選択します。
6. [Sign certificate signing request (CSR)] ページで、[Next] を選択します。
7. [Upload the certificates] ページで、以下の作業を行います。

a. [Cluster certificate (クラスター証明書)] の横にある [Upload file (ファイルのアップロード)] を選択します。先に署名した HSM 証明書を探し選択します。前のセクションのステップを完了したら、*<cluster ID>*_CustomerHsmCertificate.crt という名前のファイルを選択します。

b. [証明書の発行] の横にある [ファイルのアップロード] を選択します。次に、署名証明書を選択します。前のセクションのステップを完了したら、customerCA.crt という名前のファイルを選択します。

c. [Upload and initialize] を選択します。

クラスターを初期化するには (AWS CLI)

- コマンドラインプロンプトで、initialize-cluster コマンドを実行します。以下を指定します。
  - 前に作成したクラスターの ID。
  - 前に署名した HSM 証明書。前のセクションのステップを完了すると、*<cluster ID>*_CustomerHsmCertificate.crt というファイル名で保存されています。
  - 署名用証明書。前のセクションのステップを完了すると、署名証明書は「customerCA.crt」というファイル名で保存されています。

```
$ aws cloudhsmv2 initialize-cluster --cluster-id <cluster ID> \
                                    --signed-cert file://<cluster
 ID>_CustomerHsmCertificate.crt \
                                    --trust-anchor file://customerCA.crt
{
    "State": "INITIALIZE_IN_PROGRESS",
    "StateMessage": "Cluster is initializing. State will change to INITIALIZED upon
 completion."
}
```

クラスターを初期化するには (AWS CloudHSM API)

- 以下を使用して InitializeCluster リクエストを送信します。
  - 前に作成したクラスターの ID。
  - 前に署名した HSM 証明書。

- 署名用証明書。

CloudHSM CLI をインストールして設定する

AWS CloudHSM クラスター内の HSM を操作するには、CloudHSM CLI が必要です。

タスク

- AWS CloudHSM コマンドラインツールをインストールする

AWS CloudHSM コマンドラインツールをインストールする

クライアントインスタンスに接続し、次のコマンドを実行して、 AWS CloudHSM コマンドラインツールをダウンロードしてインストールします。詳細については、「Amazon EC2 クライアントインスタンスの起動」を参照してください。

次のコマンドを使用して、CloudHSM CLI をダウンロードしてインストールします。

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

ARM64 の Amazon Linux 2 アーキテクチャ:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.aarch64.rpm
```

Amazon Linux 2023

x86_64 アーキテクチャの Amazon Linux 2023:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

ARM64 アーキテクチャの Amazon Linux 2023:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-
latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

RHEL 7 (7.8+)

x86_64 アーキテクチャの RHEL 7:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-
latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

x86_64 アーキテクチャの RHEL 8:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-cli-
latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el8.x86_64.rpm
```

RHEL 9 (9.2+)

x86_64 アーキテクチャの RHEL 9:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.x86_64.rpm
```

ARM64 アーキテクチャの RHEL 9:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.aarch64.rpm
```

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/cloudhsm-cli_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u20.04_amd64.deb
```

Ubuntu 22.04 LTS

x86_64 アーキテクチャの Ubuntu 22.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_amd64.deb
```

ARM64 アーキテクチャの Ubuntu 22.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_arm64.deb
```

Windows Server 2016

x86_64 アーキテクチャの Windows Server 2016 の場合は、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2019

x86_64 アーキテクチャの Windows Server 2019 の場合は、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

次のコマンドを使用して CloudHSM CLI を設定します。

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IP addresses of the HSMs>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI IP addresses
 of the HSMs>
```

クラスターのアクティブ化

AWS CloudHSM クラスターをアクティブ化すると、クラスターの状態が初期化からアクティブに変わります。その後、ハードウェアセキュリティモジュール (HSM) のユーザーを管理 し、HSM を使用します。

Important

クラスターをアクティブ化する前に、クラスターに接続する各 EC2 インスタンス上のプラットフォームのために、デフォルトの場所に発行証明書をコピーする必要があります (クラスターを初期化するときに、発行証明書を作成します)。

Linux

```
/opt/cloudhsm/etc/customerCA.crt
```

Windows

```
C:\ProgramData\Amazon\CloudHSM\customerCA.crt
```

発行証明書を配置したら、CloudHSM CLI をインストールし、最初の HSM で cluster activate コマンドを実行します。クラスター内の最初の HSM の管理者アカウントに unactivated-admin ロールが割り当てられているはずです。これはクラスターがアクティブ化される前にのみ存在する一時的なロールです。クラスターをアクティブ化すると、非アクティブ化された管理者のロールは管理者に変わります。

クラスターをアクティブ化する

1. 以前に起動したクライアントインスタンスに接続します。詳細については、「Amazon EC2 クライアントインスタンスの起動」を参照してください。Linux インスタンスまたは Windows Server を起動できます。

2. CloudHSM CLI をインタラクティブモードで実行します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

3. (オプション) user list コマンドを使用して、既存のユーザーを表示します。

   ```
   aws-cloudhsm > user list
   {
     "error_code": 0,
     "data": {
       "users": [
         {
           "username": "admin",
           "role": "unactivated-admin",
           "locked": "false",
           "mfa": [],
           "cluster-coverage": "full"
         },
         {
           "username": "app_user",
           "role": "internal(APPLIANCE_USER)",
           "locked": "false",
           "mfa": [],
           "cluster-coverage": "full"
         }
       ]
     }
   }
   ```

4. cluster activate コマンドを使用して初期管理者パスワードを設定します。

   ```
   aws-cloudhsm > cluster activate
   Enter
    password:<NewPassword>
   Confirm password:<NewPassword>
   ```

```
{
  "error_code": 0,
  "data": "Cluster activation successful"
}
```

新しいパスワードをパスワードワークシートに書き留めておくことをお勧めします。ワークシートを紛失しないでください。パスワードワークシートのコピーを印刷することをお勧めします。このコピーに重要な HSM のパスワードをメモし、安全な場所に保存してください。また、このワークシートのコピーをオフサイトの安全なストレージに保存することをお勧めします。

5. (オプション) user list コマンドを使用して、ユーザーのタイプが admin/CO に変更されていることを確認します。

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
       {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

6. quit コマンドを使用して、CloudHSM CLI ツールを停止します。

```
aws-cloudhsm > quit
```

CMU または CloudHSM CLI のオペレーションの詳細については、HSM ユーザーについて と CMU での HSM ユーザー管理について を参照してください。

新しい証明書とプライベートキーで SSL を再設定する (オプション)

AWS CloudHSM は SSL 証明書を使用して HSM への接続を確立します。クライアントをインストールすると、デフォルトのキーと SSL 証明書が含まれます。しかし、自分で作成して使用することはできます。クラスターの初期化*customerCA.crt*時に作成した自己署名証明書 (???) が必要になります。

高レベルでは、これは 2 ステップのプロセスです。

1. 最初にプライベートキーを使用し、そのキーを使って証明書署名リクエスト (CSR) を作成します。CSR に署名するには、発行証明書 (クラスターの初期化時に作成した証明書) を使用します。
2. 次に構成ツールを使用して、キーと証明書を適切なディレクトリにコピーします。

キー、CSR を作成してから、CSR に署名します

手順はクライアント SDK 3 またはクライアント SDK 5 と同じです。

新しい証明書とプライベートキーで SSL を再設定するには

1. 次の OpenSSL コマンドを使用してプライベートキーを作成します。

```
openssl genrsa -out ssl-client.key 2048
Generating RSA private key, 2048 bit long modulus
........+++
............+++
e is 65537 (0x10001)
```

2. 次の OpenSSL コマンドを使用して、証明書署名リクエスト (CSR) を作成します。ユーザーの証明書について一連の質問をされます。

```
openssl req -new -sha256 -key ssl-client.key -out ssl-client.csr
Enter pass phrase for ssl-client.key:
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:
State or Province Name (full name) []:
Locality Name (eg, city) [Default City]:
Organization Name (eg, company) [Default Company Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. クラスターを初期化したときに作成した *customerCA.crt* 証明書で CSR に署名します。

```
openssl x509 -req -days 3652 -in ssl-client.csr \
        -CA customerCA.crt \
        -CAkey customerCA.key \
        -CAcreateserial \
        -out ssl-client.crt
Signature ok
subject=/C=US/ST=WA/L=Seattle/O=Example Company/OU=sales
Getting CA Private Key
```

のカスタム SSL を有効にする AWS CloudHSM

クライアント SDK 3 またはクライアント SDK 5 では、手順が異なります。configureコマンドラインツールを用いての作業のさらなる詳細については、??? を参照してください。

トピック

- クライアント SDK 3 のためのカスタム SSL
- クライアント SDK 5 のためのカスタム SSL

クライアント SDK 3 のためのカスタム SSL

クライアント SDK 3 の設定ツールを使用して、カスタム SSL を有効にします。クライアント SDK 3 のツールの構成のさらなる詳細については、??? を参照してください。。

Linux 上のクライアント SDK 3 を用いて TLS クライアント/サーバーの相互認証のためにカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 設定ツールを使用して、`ssl-client.crt` と `ssl-client.key` を指定します。

```
sudo /opt/cloudhsm/bin/configure --ssl \
 --pkey /opt/cloudhsm/etc/ssl-client.key \
 --cert /opt/cloudhsm/etc/ssl-client.crt
```

3. `customerCA.crt` 証明書を信頼ストアに追加します。証明書のサブジェクト名のハッシュを作成します。これにより、その名前で証明書を検索できるようにするインデックスが作成されます。

```
openssl x509 -in /opt/cloudhsm/etc/customerCA.crt -hash | head -n 1
1234abcd
```

   ディレクトリを作成します。

```
mkdir /opt/cloudhsm/etc/certs
```

   ハッシュ名の証明書を含むファイルを作成します。

```
sudo cp /opt/cloudhsm/etc/customerCA.crt /opt/cloudhsm/etc/certs/1234abcd.0
```

クライアント SDK 5 のためのカスタム SSL

クライアント SDK 5 の設定ツールを使用して、カスタム SSL を有効にします。クライアント SDK 5 のツールの設定のさらなる詳細については、??? を参照してください。

PKCS #11 library

Linux 上のクライアント SDK 5 を用いて TLS クライアント/サーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" `
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

OpenSSL Dynamic Engine

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

JCE provider

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" `
```

```
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

CloudHSM CLI

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" `
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

アプリケーションを構築します

を使用してアプリケーションを構築し、キーを操作します AWS CloudHSM。

新しいクラスターでキーの作成と使用を開始するには、まず CloudHSM 管理ユーティリティ (CMU) を使用してハードウェアセキュリティモジュール (HSM) ユーザーを作成する必要があります。さらなる詳細については、HSM ユーザー管理タスクについて、AWS CloudHSM コマンドラインインターフェイス (CLI) の開始方法、および HSM ユーザーの管理方法 を参照してください。

> **Note**
>
> Client SDK 3 を使用している場合は、CloudHSM CLI の代わりに CloudHSM 管理ユーティリティ (CMU) を使用してください。

HSM ユーザーが配置されている状態で、HSM にログインし、次のいずれかのオプションを使用してキーを作成および使用できます。

- key management utility、コマンドラインツール を使用
- PKCS #11 library を使用して C アプリケーションを構築する
- JCE provider を使用して Java アプリケーションを構築する
- OpenSSL Dynamic Engine directly from the command line を使用
- NGINX and Apache web servers を用いて、TLS オフロードのために [OpenSSL Dynamic Engine] を使用
- CNG および KSP プロバイダーを使用して Microsoft Windows Server Certificate Authority (CA) AWS CloudHSM で使用する
- CNG プロバイダーと KSP プロバイダーを使用して Microsoft Sign Tool AWS CloudHSM で使用する
- TLS オフロード用の CNG および KSP プロバイダーを、Internet Information Server (IIS) web server と共に使用する

のベストプラクティス AWS CloudHSM

AWS CloudHSMを効果的に使用するには、このトピックのベストプラクティスを実行してください。

内容

クラスターの管理

AWS CloudHSM クラスターを作成、アクセス、管理するときは、このセクションのベストプラクティスに従ってください。

クラスターをスケールしてピークトラフィックを処理する

クラスターが処理できる最大スループットには、クライアントインスタンスのサイズ、クラスターサイズ、ネットワークトポグラフィー、ユースケースに必要な暗号化オペレーションなど、いくつかの要因が影響する可能性があります。

手始めに、一般的なクラスターのサイズと構成に関するパフォーマンスの見積もりに関するトピック AWS CloudHSM パフォーマンス を参照してください。現在のアーキテクチャが耐障害性に優れていて、適切な規模になっているかを判断するために、予想されるピーク負荷でクラスターの負荷テストを行うことを推奨します。

高可用性対応のクラスターをアーキテクチャー

メンテナンスを考慮して冗長性を追加します。スケジュールされたメンテナンスや問題を検出した場合は、HSM を AWS 置き換えることができます。原則として、クラスターサイズには少なくとも +1 の冗長性が必要です。たとえば、ピーク時にサービスを稼働させるために HSM が 2 つ必要であれば、理想的なクラスターサイズは 3 つになります。可用性に関するベストプラクティスに従えば、HSM を置き換えてもサービスに影響はないはずです。ただし、交換した HSM で進行中のオペレーションは失敗する可能性があるため、再試行する必要があります。

HSM を複数のアベイラビリティーゾーンに分散させる: アベイラビリティーゾーンが停止した際に、サービスをどのように運用できるかを検討してください。 AWS では、HSM をできるだけ多くのアベイラビリティーゾーンに分散することを推奨しています。HSM が 3 つあるクラスターでは、HSM を 3 つのアベイラビリティーゾーンに分散する必要があります。システムによっては、追加の冗長性が必要な場合があります。

新しく生成されたキーの耐久性を確保するために、HSM を少なくとも 3 つ用意してください

新しく生成されたキーの耐久性が必要なアプリケーションの場合は、リージョン内の異なるアベイラビリティーゾーンに少なくとも 3 つの HSM を分散させることを推奨します。

クラスターへの安全なアクセス

プライベートサブネットを使用してインスタンスへのアクセスを制限する: HSM とクライアントインスタンスは、VPC のプライベートサブネットで起動します。これにより、外部からの HSM へのアクセスが制限されます。

VPC エンドポイントを使用して APIs にアクセスする: AWS CloudHSM データプレーンは、インターネットや AWS APIs にアクセスすることなく動作するように設計されています。クライアントインスタンスが AWS CloudHSM API にアクセスする必要がある場合は、クライアントインスタンスでインターネットアクセスを必要とせずに、VPC エンドポイントを使用して API にアクセスできます。詳細については、「AWS CloudHSM および VPC エンドポイント」を参照してください。

SSL を再構成してクライアント/サーバー通信 :TLS AWS CloudHSM を使用して HSM への接続を確立します。クラスターを初期化したら、外部 TLS 接続の確立に使用したデフォルトの TLS 証明書とキーを置き換えることができます。詳細については、「での SSL/TLS オフロードによるウェブサーバーセキュリティの向上 AWS CloudHSM」を参照してください。

ニーズに合わせてスケールすることでコストを削減

AWS CloudHSMの使用には初期費用はかかりません。HSM を終了するまで、HSM を起動するたびに 1 時間あたりの料金をお支払いいただきます。サービスで の継続的な使用が必要ない場合は AWS CloudHSM、HSM が不要になったときにゼロ HSMs にスケールダウン (削除) することでコストを削減できます。HSM が再び必要になった場合は、HSM をバックアップから復元できます。たとえば、月に一度、具体的にはその月の最終日にコードに署名する必要があるワークロードがある場合は、その前にクラスターをスケールアップし、作業完了後に HSM を削除してスケールダウンし、翌月末にクラスターを復元して署名オペレーションを再実行することができます。

AWS CloudHSM は、クラスター内の HSMs定期的なバックアップを自動的に作成します。後日新しい HSM を追加すると、 AWS CloudHSM は最新のバックアップを新しい HSM に復元し、残したのと同じ場所から使用を再開できるようにします。 AWS CloudHSM アーキテクチャコストを計算するには、「 のAWS CloudHSM 料金」を参照してください。

関連リソース:

- バックアップの一般的な概要
- バックアップの保持ポリシー
- AWS リージョン間でのバックアップのコピー

HSM ユーザー管理

このセクションのベストプラクティスに従って、 AWS CloudHSM クラスター内のユーザーを効果的に管理します。HSM ユーザーは IAM ユーザーとは異なります。適切な権限を持つ ID ベースのポリシーを持つ IAM ユーザーとエンティティは、AWS API を介してリソースを操作することで HSM を作成できます。HSM を作成したら、HSM ユーザー認証情報を使用して HSM でのオペレーションを認証する必要があります。HSM ユーザーの詳細なガイドについては、「での HSM ユーザーの管理 AWS CloudHSM」を参照してください。

HSM ユーザーの認証情報の保護

HSM ユーザーは HSM にアクセスして暗号化オペレーションや管理オペレーションを実行できるため、HSM ユーザーの認証情報を安全に保護することが不可欠です。 AWS CloudHSM は HSM ユーザー認証情報にアクセスできないため、認証情報にアクセスできなくなった場合はサポートできません。

ロックアウトを防ぐため、少なくとも 2 人の管理者を配置します

クラスターからロックアウトされないように、1 つの管理者パスワードを紛失した場合に備えて、少なくとも 2 人の管理者を配置することをお勧めします。このような場合は、もう一方の管理者にパスワードをリセットしてもらうことができます。

> **Note**
>
> Client SDK 5 の管理者は、クライアント SDK 3 の Crypto Officer (CO) と同義です。

すべてのユーザー管理オペレーションでクォーラムを有効にします

クォーラムでは、ユーザー管理オペレーションを承認しないと実行できない管理者の最小数を設定できます。管理者には権限があるため、すべてのユーザー管理オペレーションでクォーラムを有効にすることをお勧めします。これにより、管理者パスワードのいずれかが侵害された場合の影響を抑えることができます。詳細については、Managing Quorum を参照してください。

各自の権限を制限した複数の暗号ユーザーを作成する

暗号ユーザーの責任を分離することで、1 人のユーザーがシステム全体を完全に制御できなくなります。このため、複数の暗号ユーザーを作成し、それぞれの権限を制限するようお勧めします。通常、これは異なる暗号ユーザーにそれぞれ異なる責任と実行するアクションを明確に与えることによって行われます (たとえば、1 人の暗号ユーザーがキーを生成して他の暗号ユーザーと共有し、その暗号ユーザーをアプリケーションで利用させるなど)。

関連リソース:

- キーシェア
- キー共有解除

HSM キー管理

AWS CloudHSMでキーを管理する場合は、このセクションのベストプラクティスに従います。

適切なキーのタイプを選択する

セッションキーを使用する場合、1 秒あたりのトランザクション数 (TPS) は、そのキーが存在する 1 つの HSM に制限されます。クラスターに HSM を追加しても、そのキーのリクエストのスループットは向上しません。同じアプリケーションにトークンキーを使用すると、リクエストはクラスター内の利用可能なすべての HSM に負荷分散されます。詳細については、「でのキー同期と耐久性の設定 AWS CloudHSM」を参照してください。

キーのストレージ制限を管理する

HSM には、HSM に一度に保存できるトークンとセッションキーの最大数に制限があります。キーストレージの制限に関する詳細は、「AWS CloudHSM クォータ」を参照してください。アプリケーションが制限を超えるものを必要とする場合は、次の 1 つまたは複数の方法を使用してキーを効果的に管理できます。

トラステッドラッピングを使用してキーを外部データストアに保存する:トラステッドキーラッピングを使用すると、ラップされたすべてのキーを外部データストア内に保存することで、キーストレージの制限を回避できます。このキーを使用する必要がある場合は、キーをセッションキーとして HSM にアンラップし、そのキーを必要なオペレーションに使用してからセッションキーを破棄できます。元のキーデータはデータストアに安全に保存され、必要なときにいつでも使用できます。そのために信頼できるキーを使用することで、最大限の保護が可能になります。

キーをクラスターに分散する: キーストレージの制限を克服するためのもう 1 つの方法は、キーを複数のクラスターに保存することです。このアプローチでは、各クラスターに保存されているキーのマッピングを維持します。このマッピングを使用して、必要なキーを使用してクライアントリクエストをクラスターにルーティングします。同じクライアントアプリケーションから複数のクラスターに接続する方法の詳細については、以下のトピックを参照してください。

- JCE プロバイダーによる複数のクラスターへの接続
- PKCS #11 を使用した複数のスロットへの接続

キーラッピングの管理と保護

キーには、EXTRACTABLE 属性によって抽出可能または抽出不可のマークが付けられます。デフォルトでは、HSM キーは抽出可能とマークされています。

抽出可能なキーとは、キーラッピングによって HSM からエクスポートできるキーのことです。ラップされたキーは暗号化されるため、使用する前に同じラップキーを使用してラップを解除する必要があります。抽出不可能なキーは、いかなる状況においても HSM からエクスポートすることはできません。抽出不可能なキーを抽出可能にする方法はありません。このため、キーを抽出可能にする必要があるかどうかを検討し、それに応じて対応するキー属性を設定することが重要です。

アプリケーションでキーラッピングが必要な場合は、信頼できるキーラッピングを利用して、管理者によって明示的に信頼できるとマークされたキーのみを HSM ユーザーがラップ/アンラップできるように制限する必要があります。詳細については、でのキーの管理 AWS CloudHSM の「信頼できるキーラッピングに関するトピック」を参照してください。

関連リソース

- ラップ関数とアンラップ関数
- JCE の暗号関数
- サポートされている Java キー属性

- CloudHSM CLI のキー属性

アプリケーション統合

このセクションのベストプラクティスに従って、アプリケーションと AWS CloudHSM クラスターの統合方法を最適化します。

Client SDK のブートストラップ

Client SDK をクラスターに接続する前に、ブートストラップする必要があります。IP アドレスをクラスターにブートストラップするときは、可能であれば `--cluster-id` パラメーターを使用することをおすすめします。この方法では、個々のアドレスを追跡しなくても、クラスター内のすべての HSM IP アドレスが設定に入力されます。これにより、HSM がメンテナンス中であったり、アベイラビリティーゾーンが停止した場合でも、アプリケーションの初期化の回復力がさらに高まります。詳細については、「クライアント SDK をブートストラップする」を参照してください。

認証してオペレーションを実行

では AWS CloudHSM、暗号化オペレーションなどのほとんどのオペレーションを実行する前に、クラスターに対して認証する必要があります。

CloudHSM CLI による認証: CloudHSM CLI による認証は、シングルコマンドモード またはインタラクティブモード のいずれかを使用して行うことができます。login (ログイン) コマンドを使用してインタラクティブモードで認証します。シングルコマンドモードで認証するには、環境変数 `CLOUDHSM_ROLE` および `CLOUDHSM_PIN` を設定する必要があります。その方法の詳細については、「シングルコマンドモード」を参照してください。 AWS CloudHSM では、アプリケーションで使用されていないときは、HSM 資格情報を安全に保管することをお勧めします。

PKCS #11 で認証する: PKCS #11 では、C_ を使用してセッションを開いた後に C_Login API を使用してログインしますOpenSession。C_Login はスロット (クラスター) ごとに 1 つだけ実行する必要があります。正常にログインしたら、追加のログイン操作を実行するOpenSession ことなく、C_ を使用して追加のセッションを開くことができます。PKCS #11 への認証の例については、「PKCS #11 ライブラリのコードサンプル」を参照してください。

JCE による認証: AWS CloudHSM JCE プロバイダーは、暗黙的なログインと明示的なログインの両方をサポートします。適切な方法は、ユースケースによって異なります。アプリケーションがクラスターから切断され、再認証が必要になった場合、SDK が自動的に認証を処理するため、可能な場合は Implicit Login を使用することをお勧めします。暗黙的ログインを使用すると、アプリケーションコードを制御できないインテグレーションを使用する場合でも、アプリケーションに認証情報を提供

できます。ログイン方法の詳細については、「JCE プロバイダーに認証情報を提供する」を参照してください。

OpenSSL による認証: OpenSSL 動的エンジンでは、環境変数を使用して認証情報を指定します。AWS CloudHSM では、HSM 認証情報は、アプリケーションで使用しないときは安全に保存することをお勧めします。可能であれば、手動で入力しなくてもこれらの環境変数を体系的に取得して設定するように環境を設定する必要があります。OpenSSL による認証の詳細については、「OpenSSL Dynamic Engine のインストール」を参照してください。

アプリケーションのキーを効果的に管理する

キー属性を使用してキーで実行できる操作を制御する: キーを生成するときは、キー属性を使用して、そのキーに対する特定の種類の操作を許可または拒否する一連の権限を定義します。キーは、タスクを完了するのに必要な最小限の属性で生成することをおすすめします。たとえば、暗号化に使用する AES キーを HSM からラップアウトすることも許可しないでください。詳細については、以下のClient SDK の属性ページを参照してください。

- PKCS #11 の主要属性
- JCE キーの属性

可能な場合は、キーオブジェクトをキャッシュしてレイテンシーを最小限に抑える: キー検索オペレーションはクラスター内のすべての HSM にクエリを実行します。このオペレーションはコストがかかり、クラスター内の HSM 数に合わせて拡張できません。

- PKCS #11 では、 `C_FindObjects` API を使用してキーを検索します。
- JCE では、 を使用してキーを検索します KeyStore。

最適なパフォーマンス AWS を得るには、アプリケーションの起動時にキー検索コマンド (findKeyや などキーリスト) を 1 回だけ使用し、返されたキーオブジェクトをアプリケーションメモリにキャッシュすることをお勧めします。後でこのキーオブジェクトが必要になった場合は、オペレーションのたびにこのオブジェクトをクエリするのではなく、キャッシュからオブジェクトを取得する必要があります。そうすると、パフォーマンスのオーバーヘッドが大幅に増加します。

マルチスレッドを使用してください

AWS CloudHSM はマルチスレッドアプリケーションをサポートしていますが、マルチスレッドアプリケーションには注意すべき点がいくつかあります。

PKCS #11 では、PKCS #11 ライブラリ (呼び出し側C_Initialize) は 1 回だけ初期化する必要があります。各スレッドには独自のセッションを割り当てる必要があります (C_OpenSession)。同じセッションを複数のスレッドで使用することはお勧めしません。

JCE では、 AWS CloudHSM プロバイダーは 1 回だけ初期化する必要があります。SPI オブジェクトのインスタンスをスレッド間で共有しないでください。例えば、暗号、署名、ダイジェスト、Mac、 KeyFactory または KeyGenerator オブジェクトは、独自のスレッドのコンテキストでのみ使用する必要があります。

スロットリングエラーの処理

以下の条件下では、HSM スロットリングエラーが発生する可能性があります。

- クラスターがピーク時のトラフィックを管理できるよう適切にスケーリングされていない。
- メンテナンスイベント中は、クラスターのサイズが +1 の冗長性を持たない。
- アベイラビリティーゾーンが停止すると、クラスターで使用可能な HSM の数が減少する。

このシナリオを最適に処理する方法については、「HSM スロットリング」を参照してください。

クラスターのサイズが適切でスロットリングされないように、 AWS では、予想されるピークトラフィックで環境で負荷テストを行うことをお勧めします。

クラスターオペレーションのリトライを統合してください

AWS は、運用上またはメンテナンス上の理由で HSM を置き換える場合があります。このような状況に対してアプリケーションに回復力を持たせるために、 では、クラスターにルーティングされるすべてのオペレーションにクライアント側の再試行ロジックを実装 AWS することをお勧めします。置換によって失敗したオペレーションを後で再試行しても、成功することが期待されます。

ディザスタリカバリ戦略の実装

イベントに対応して、トラフィックをクラスター全体またはリージョン全体から遠ざける必要がある場合があります。以下のセクションでは、そのための複数の戦略について説明します。

VPC ピアリングを使用して、別のアカウントまたはリージョンからクラスターにアクセスする: VPC ピアリングを使用して、別のアカウントまたはリージョンから AWS CloudHSM クラスターにアクセスできます。詳細については、VPC Peering Guideの「VPC ピア機能とは」を参照してください。ピアリング接続を確立し、セキュリティグループを適切に設定すると、通常と同じ方法で HSM IP アドレスと通信できます。

同じアプリケーションから複数のクラスターに接続する: クライアント SDK 5 の JCE プロバイダー、PKCS #11 ライブラリ、および CloudHSM CLI は、同じアプリケーションから複数のクラスターへの接続をサポートしています。たとえば、それぞれ異なるリージョンにある 2 つのアクティブなクラスターがある場合、アプリケーションは両方に一度に接続して、通常のオペレーションの一部としてこの 2 つのクラスター間の負荷を分散できます。アプリケーションが Client SDK 5 (最新の SDK) を使用していない場合、同じアプリケーションから複数のクラスターに接続することはできません。あるいは、別のクラスターを稼働させ続け、地域的な障害が発生した場合には、ダウンタイムを最小限に抑えるためにトラフィックを他のクラスターに移すこともできます。詳細については、それぞれのページを参照してください。

- PKCS #11 を使用した複数のスロットへの接続
- JCE プロバイダーによる複数のクラスターへの接続
- CloudHSM CLI を使用した複数のクラスターへの接続

バックアップからのクラスターの復元: 既存のクラスターのバックアップから新しいクラスターを作成できます。詳細については、「AWS CloudHSM バックアップの管理」を参照してください。

モニタリング

このセクションでは、クラスターとアプリケーションのモニタリングに使用できる複数のメカニズムについて説明します。モニタリングの詳細については、「モニタリング AWS CloudHSM」を参照してください。

クライアントログのモニタリング

すべてのClient SDK には、監視可能なログが書き込まれます。ログ記録の詳細については、「Client SDK ログの操作」を参照してください。

Amazon ECS や など、一時的なプラットフォームでは AWS Lambda、ファイルからクライアントログを収集するのが難しい場合があります。このような状況では、ログをコンソールに書き込むように Client SDK ロギングを設定するのがベストプラクティスです。ほとんどの のサービスでは、この出力が自動的に収集され、Amazon CloudWatch ログに公開され、保持および表示できます。

AWS CloudHSM Client SDK 上でサードパーティー統合を使用している場合は、そのソフトウェアパッケージが出力をコンソールにもログ記録するように設定する必要があります。 AWS CloudHSM Client SDK からの出力は、このパッケージによってキャプチャされ、それ以外の場合は独自のログファイルに書き込まれる場合があります。

アプリケーションのロギングオプションの設定方法については、「Client SDK 5 設定ツール」を参照してください。

監査ログのモニタリング

AWS CloudHSM は、Amazon CloudWatch アカウントに監査ログを発行します。監査ログは HSM から取得され、監査目的で特定のオペレーションを追跡します。

監査ログを使用して、HSM で呼び出された管理コマンドを追跡できます。たとえば、予期しない管理オペレーションが実行されていることに気付いたときにアラームをトリガーできます。

詳細については、「HSM 監査ログの記録の仕組み」を参照してください。

モニタリング AWS CloudTrail

AWS CloudHSM は、ユーザー AWS CloudTrail、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています AWS CloudHSM。 は、 のすべての API コールをイベント AWS CloudHSM として AWS CloudTrail キャプチャします。キャプチャされた呼び出しには、 AWS CloudHSM コンソールからの呼び出しと AWS CloudHSM API オペレーションへのコード呼び出しが含まれます。

を使用して AWS CloudTrail 、 AWS CloudHSM コントロールプレーンに対して行われた API コールを監査し、アカウントで望ましくないアクティビティが発生していないことを確認できます。

詳細については、「AWS CloudTrail および の使用 AWS CloudHSM」を参照してください。

Amazon CloudWatch メトリクスのモニタリング

Amazon CloudWatch メトリクスを使用して、 AWS CloudHSM クラスターをリアルタイムでモニタリングできます。メトリクスは、リージョン、クラスター ID、HSM ID、およびクラスター ID ごとにグループ化できます。

Amazon CloudWatch メトリクスを使用すると、サービスに影響を与える可能性のある潜在的な問題を警告するように Amazon CloudWatch アラームを設定できます。以下を監視するようにアラームを設定することをお勧めします。

- HSM のキー制限に近づく
- HSM の HSM セッション数の制限に近づいています
- HSM の HSM ユーザー数制限に近づいています。

- 同期の問題を特定するための HSM ユーザー数またはキー数の違い
- が AWS CloudHSM 問題を解決するまでクラスターをスケールアップする異常な HSMs

詳細については、「Amazon CloudWatch Logs と AWS CloudHSM 監査ログの使用」を参照してください。

AWS CloudHSM クラスターの管理

AWS CloudHSM クラスターは、 AWS CloudHSM コンソール、 AWS SDKs のいずれか、またはコマンドラインツール から管理できます。詳細については、以下のトピックを参照してください。

クラスターを作成するには、「開始」を参照してください。

クラスターアーキテクチャ

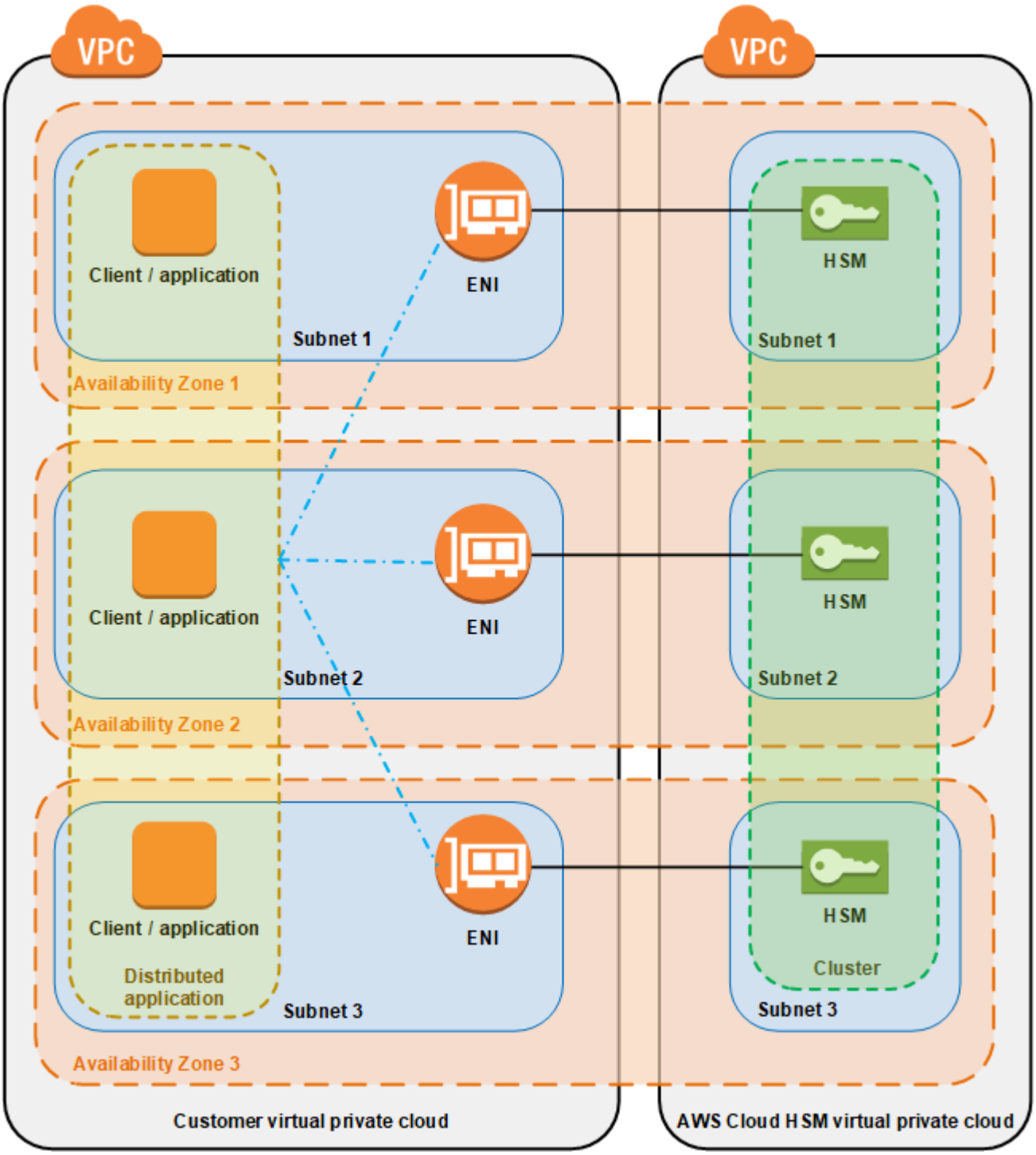
クラスターを作成するときは、 AWS アカウントに Amazon Virtual Private Cloud (VPC) を指定し、その VPC に 1 つ以上のサブネットを指定します。選択した AWS リージョンの各アベイラビリティーゾーン (AZ) に 1 つのサブネットを作成することをお勧めします。VPC を作成するときにプライベートサブネットを作成できます。詳細については、「仮想プライベートクラウド (VPC) の作成」を参照してください。

HSM を作成する度に、HSM のクラスターとアベイラビリティーゾーンを指定します。HSM を別々のアベイラビリティーゾーンに指定すると、いずれかのアベイラビリティーゾーンが使用できなくなった場合でも冗長性と高可用性を維持します。

HSM を作成すると、 は AWS アカウント内の指定されたサブネットに Elastic Network Interface (ENI) AWS CloudHSM を配置します。Elastic Network Interface は、HSM とやり取りするためのインターフェイスです。HSM は、 が所有する AWS アカウントの別の VPC にあります AWS CloudHSM。HSM と対応するネットワークインターフェイスは、同じアベイラビリティーゾーンに存在します。

クラスター内の HSMsを操作するには、 AWS CloudHSM クライアントソフトウェアが必要です。通常、次の図に示すように、HSM ENI と同じ VPC にある Amazon EC2 インスタンス (クライアントインスタンス) でクライアントをインストールします。ただし、これは技術的には必要ありません。HSM ENI に接続できる限り、互換性のある任意のコンピュータでクライアントをインストールできます。クライアントは ENI を通じてクラスター内の個々の HSM と通信します。

次の図は、それぞれ VPC 内の異なるアベイラビリティーゾーンにある 3 HSMs を持つ AWS CloudHSM クラスターを示しています。



クラスターの同期

AWS CloudHSM クラスターでは、 AWS CloudHSM は個々の HSMsのキーを同期させます。HSM 上でキーを同期するために必要な操作はありません。各 HSM のユーザーとポリシーを同期させるに

は、HSM ユーザー を管理する前に AWS CloudHSM クライアント設定ファイルを更新します。詳細については、「HSM ユーザーを同期する」を参照してください。

クラスターに新しい HSM を追加すると、 AWS CloudHSM は既存の HSM のすべてのキー、ユーザー、ポリシーのバックアップを作成します。次に、そのバックアップが新しい HSM に復元されます。これにより、2 つの HSM の同期が保たれます。

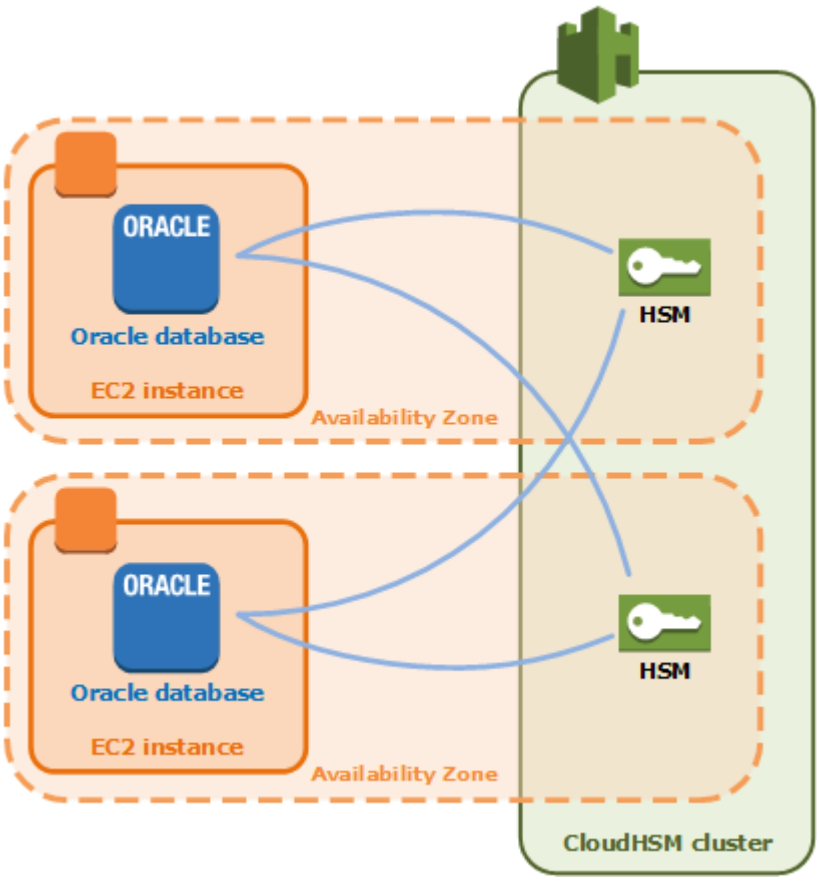
クラスター内の HSMs が同期に失敗した場合、 はそれら AWS CloudHSM を自動的に再同期します。これを有効にするには、アプライアンスユーザー の認証情報 AWS CloudHSM を使用します。このユーザーは、 が提供するすべての HSMsに存在し AWS CloudHSM 、アクセス許可が制限されています。HSM でオブジェクトのハッシュの取得と、マスク (暗号化) されたオブジェクトの抽出および挿入を行うことができます。 AWS は、ユーザーあるいはキーの表示や変更、およびこのキーを使用した一切の暗号化オペレーションを実行することはできません。

クラスターの高可用性とロードバランシング

複数の HSM を持つ AWS CloudHSM クラスターを作成すると、自動的に負荷分散が行われます。ロードバランシングは、追加の処理に対する HSM の容量に基づき、AWS CloudHSM クライアントによって、クラスター内のすべての HSM に暗号化オペレーションが分散されることを意味します。

異なる AWS アベイラビリティーゾーンに HSMs を作成すると、高可用性が自動的に取得されます。高可用性は、個々の HSM に単一障害点がないことにより、高い信頼性を取得できることを意味します。各クラスターには最低 2 つの HSMsがあり、各 HSM は AWS リージョン内の異なるアベイラビリティーゾーンにあることをお勧めします。

たとえば、次の図では、Oracle データベースアプリケーションが 2 つの異なるアベイラビリティーゾーンに分散されています。データベースインスタンスは、各アベイラビリティーゾーンの HSM を含むクラスターにマスターキーを保存します。 は、キーを両方の HSMs AWS CloudHSM に自動的に同期して、すぐにアクセスでき冗長になるようにします。



AWS CloudHSM クラスターモードと HSM タイプ

AWS CloudHSM には、FIPS と非 FIPS の 2 つのクラスターモードがあります。 には、hsm1.medium と hsm2m.medium の 2 つの HSM タイプ AWS CloudHSM もあります。ニーズに合ったクラスターモードと HSM タイプを決定する前に、このページの詳細を確認してください。

> **Note**
>
> 2024 年 6 月 10 日より前に作成されたすべてのクラスターは FIPS モードであり、HSM タイプは hsm1.medium です。

クラスターのモードと HSM タイプを確認するには、describe-clusters コマンドを使用します。

クラスターモード

AWS CloudHSM は、FIPS と非 FIPS の 2 つのモードでクラスターを提供します。FIPS モードでは、連邦情報処理規格 (FIPS) によって承認されたキーとアルゴリズムのみを使用できます。非 FIPS

モードは、FIPS の承認に関係なく AWS CloudHSM、 でサポートされているすべてのキーとアルゴリズムを提供します。

次の表に、各クラスターモードの主な違いを示します。

差別化機能	FIPS モード	非 FIPS モード
HSM タイプの互換性	hsm1.medium で使用できます。	hsm2m.medium で使用できます。
バックアップ互換性	FIPS モードでの復元クラスターのバックアップにのみ使用できます。	非 FIPS モードでの復元クラスターのバックアップにのみ使用できます。
キーの選択	FIPS 承認済みの AWS CloudHSM キーをサポートします[1]。	FIPS 承認の AWS CloudHSM キーと FIPS 承認のキーの両方をサポートします。
アルゴリズム	FIPS 承認済みの AWS CloudHSM アルゴリズムをサポートします[1]。	FIPS 承認と FIPS 承認の両方の AWS CloudHSM アルゴリズムをサポートします。
証明書	FIPS 140-2、PCI PIN、PCI-3 DS に準拠しています。	

[1] 詳細については、「Deprecation notifications」を参照してください。

クラスターモードを選択する前に、クラスターのモード (FIPS または非 FIPS) は作成後に変更できないことに注意してください。そのため、ニーズに適したモードを選択してください。

HSM タイプ

クラスターモードに加えて、 は hsm1.medium と hsm2m.medium の 2 つの HSM タイプ AWS CloudHSM を提供します。HSM の種類ごとに使用するハードウェアは異なり、各クラスターには 1 種類の HSM しか含めることができません。次の表はこの 2 つの主な相違点の一覧です。

差別化機能	hsm1.medium	hsm2m.medium
クラスターモードの互換性	FIPS モードでクラスターで使用できます。	現在、FIPS 以外のモードのクラスターで使用できます。
バックアップ互換性	hsm1.medium クラスターへの復元のバックアップにのみ使用できます。	hsm2m.medium クラスターのバックアップにのみ使用できます。
キー容量	クラスターあたり 3,300。	合計 16,666 個のキー、非対称キーはクラスターあたり最大 3,333 個です。
Client SDK	すべての Client SDK をサポートします。	CNG プロバイダーSDKs をサポートします。 ???
Client SDK のバージョン	SDK バージョン 3.1.0 以降と互換性があります。	Client SDK バージョン 5.12.0 以降と互換性があります。
利用可能なリージョン	CloudHSM が利用可能なすべてのリージョンで使用できます。	今後追加でサポートされるリージョンを含む、限られた数のリージョンで利用できます。この HSM タイプが利用可能なリージョンを確認するには、AWS CloudHSM 「料金計算ツール」を参照してください。
パフォーマンス	各 HSM タイプのパフォーマンスを確認するには、AWS CloudHSM パフォーマンス を参照してください。	
証明書	FIPS 140-2、PCI DSS、PCI PIN、SOC2、PCI-3DS に準拠しています。	PCI DSS 準拠。

[1] 詳細については、「Deprecation notifications」を参照してください。

クライアント SDK を AWS CloudHSM クラスターに接続する

クライアント SDK 5 またはクライアント SDK 3 のいずれかを用いてクラスターに接続するには、まず 2 つの操作を行う必要があります。

- EC2 インスタンス上に発行証明書を配置する
- クライアント SDK をクラスターにブートストラップする

各 EC2 インスタンス上に発行証明書を配置する

クラスターの初期化時には、発行証明書を作成します。クラスターに接続する各 EC2 インスタンス上のプラットフォームのために、デフォルトの場所への発行証明書をコピーします。

Linux

```
/opt/cloudhsm/etc/customerCA.crt
```

Windows

```
C:\ProgramData\Amazon\CloudHSM\customerCA.crt
```

発行証明書の場所を指定する

Client SDK 5 を用いて、構成ツールを使用して発行証明書の場所を指定できます。

PKCS #11 library

Linux クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe" --hsm-ca-
cert <customerCA certificate file>
```

OpenSSL Dynamic Engine

Linux クライアント SDK 5 の発行証明書を配置する

- 構成ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --hsm-ca-cert <customerCA certificate
 file>
```

JCE provider

Linux クライアント SDK 5 の発行証明書を配置する

- 設定ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-jce.exe" --hsm-ca-cert <customerCA
 certificate file>
```

CloudHSM CLI

Linux クライアント SDK 5 の発行証明書を配置する

- 設定ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-cli.exe" --hsm-ca-cert <customerCA
 certificate file>
```

さらなる詳細については、Configure Tool を参照してください。

クラスターの初期化、または証明書の作成と署名の詳細については、Initilize the Cluster を参照してください。

クライアント SDK をブートストラップする

ブートストラッププロセスは、使用しているクライアント SDK のバージョンによって異なりますが、クラスター内のいずれかのハードウェアセキュリティモジュール (HSM) の IP アドレスが必要です。クラスターに添付されている任意の HSM の IP アドレスを使用できます。クライアント SDK が接続すると、あらゆる追加の HSM の IP アドレスを取得し、ロードバランシングとクライアント側のキー同期操作を実行します。

クラスターのために IP アドレスを取得するには

HSM の IP アドレスを取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. クラスターの詳細ページを開くには、クラスターテーブルでクラスター ID を選択します。
4. IP アドレスを取得するには、[HSM] タブで、[ENI IP アドレス] にリストされている IP アドレスのいずれかを選択します。

HSM の IP アドレスを取得するには (AWS CLI)

- describe-clusters から AWS CLIコマンドを実行して、HSM の IP アドレスを取得します。コマンドからの出力では、HSM の IP アドレスは `EniIp` の値です。

```
$  aws cloudhsmv2 describe-clusters


{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
...
```

ブートストラップのさらなる詳細については、構成ツール を参照してください。

クライアント SDK 5 のブートストラップ

PKCS #11 library

クライアント SDK 5 用のための Linux の EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 -a <HSM IP addresses>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" -a <HSM IP
 addresses>
```

OpenSSL Dynamic Engine

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn -a <HSM IP addresses>
```

JCE provider

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce -a <HSM IP addresses>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" -a <HSM IP addresses>
```

CloudHSM CLI

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IP addresses of the HSMs>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI IP
 addresses of the HSMs>
```

> **Note**
>
> `--cluster-id` パラメータは `-a <HSM_IP_ADDRESSES>` の代わりに使用できます。`--cluster-id` の使用要件については、「Client SDK 5 設定ツール」を参照してください。

クライアント SDK 3 をブートストラップするには

クライアント SDK 3 用の Linux の EC2 インスタンスをブートストラップするには

- configure を使用して、クラスター内の HSM の IP アドレスを指定します。

```
sudo /opt/cloudhsm/bin/configure -a <IP address>
```

クライアント SDK 3 用の Windows EC2 インスタンスをブートストラップするには

- configure を使用して、クラスター内の HSM の IP アドレスを指定します。

```
C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe -a <HSM IP address>
```

さらなる設定の詳細については、??? を参照してください。

AWS CloudHSM クラスターでの HSMsの追加または削除

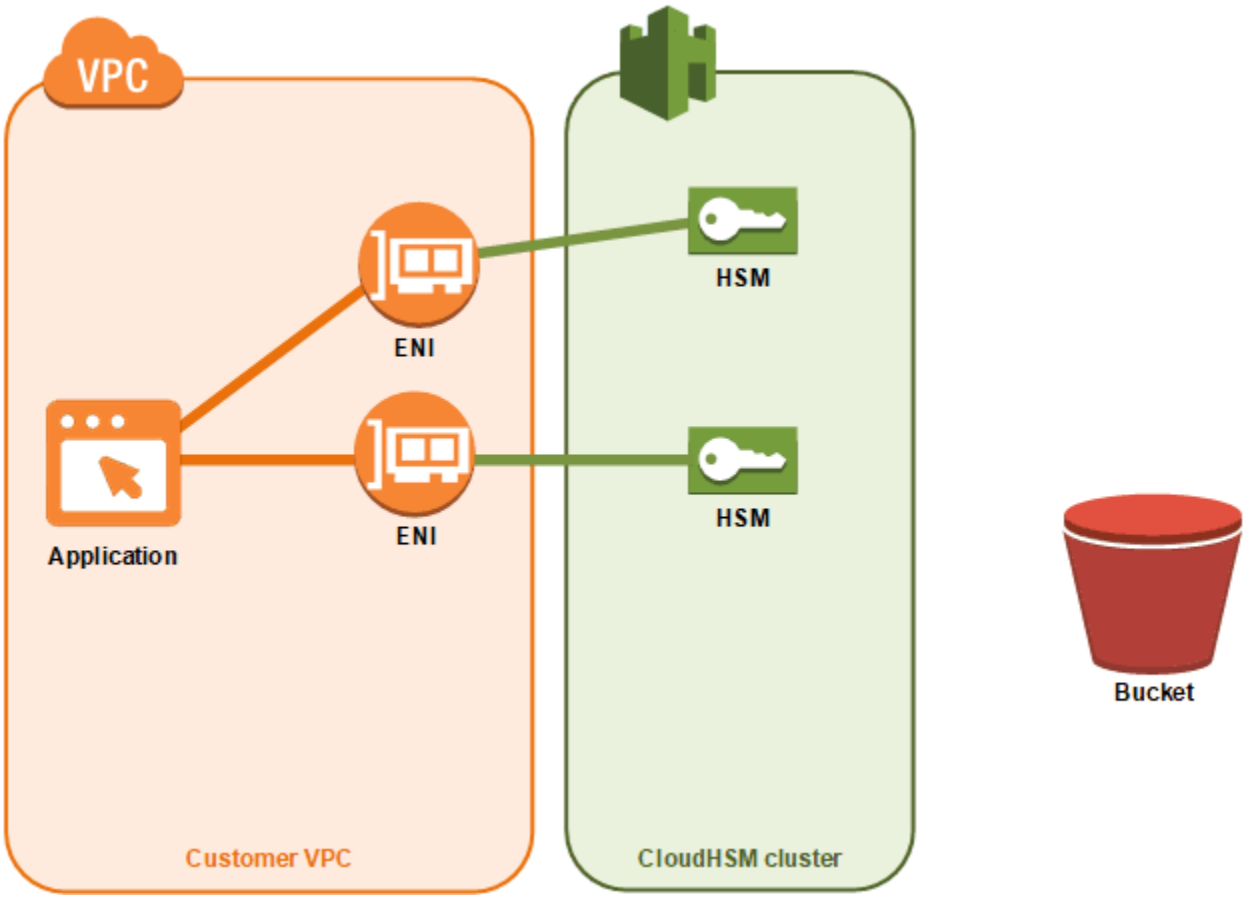
AWS CloudHSM クラスターをスケールアップまたはスケールダウンするには、 AWS CloudHSM コンソール、 SDK のいずれか、またはコマンドラインツール を使用して HSMs を追加または削除します。 AWS SDKs クラスターの負荷テストを行って予測すべきピーク負荷を決定し、高可用性を確保するためにクラスターに HSM を 1 つ追加することを推奨します。

トピック

- HSM の追加
- HSM の削除

HSM の追加

次の図は、クラスターに HSM を追加したときに発生するイベントを示しています。



1. 新しい HSM をクラスターに追加します。以下の手順では、この操作を AWS CloudHSM コンソール、AWS Command Line Interface (AWS CLI)、および AWS CloudHSM API で行う方法を示します。

   これは、ユーザーが行う唯一のアクションです。残りのイベントは自動的に実行されます。

2. AWS CloudHSM は、クラスター内の既存の HSM のバックアップコピーを作成します。詳細については、「バックアップ」を参照してください。
3. AWS CloudHSM は、バックアップを新しい HSM に復元します。これにより、HSM はクラスター内の他のインスタンスと同期されます。
4. クラスター内の既存の HSMs は、クラスター内に新しい HSM があることを AWS CloudHSM クライアントに通知します。
5. クライアントは、新しい HSM クライアントへの接続を確立します。

HSM を追加するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。

2. HSM を追加する先のクラスターを選択します。
3. [HSMs] タブで [Create HSM] を選択します。
4. 作成中の HSM のアベイラビリティーゾーン (AZ) を選択します。次に [作成] を選択します。

HSM を追加するには (AWS CLI)

- コマンドプロンプトで、create-hsm コマンドを発行し、作成する HSM 用のクラスター ID とアベイラビリティーゾーンを指定します。該当するクラスターのクラスター ID がわからない場合は、describe-clusters コマンドを発行します。アベイラビリティーゾーンは、us-east-2a、us-east-2b などの形式で指定します。

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-
zone <Availability Zone>
{
    "Hsm": {
        "State": "CREATE_IN_PROGRESS",
        "ClusterId": "cluster-5a73d5qzrdh",
        "HsmId": "hsm-lgavqitns2a",
        "SubnetId": "subnet-0e358c43",
        "AvailabilityZone": "us-east-2c",
        "EniId": "eni-bab18892",
        "EniIp": "10.0.3.10"
    }
}
```

HSM (AWS CloudHSM API) を追加するには

- CreateHsm リクエストを送信し、作成する HSM 用のクラスター ID とアベイラビリティーゾーンを指定します。

HSM の削除

AWS CloudHSM コンソール 、、AWS CLIまたは AWS CloudHSM API を使用して HSM を削除できます。

HSM を削除するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。

2. 削除する HSM が含まれているクラスターを選択します。
3. [HSMs] タブで、削除する HSM を選択します。次に、[Delete HSM] を選択します。
4. HSM を削除することを確定します。その後、[削除] をクリックします。

HSM を削除するには (AWS CLI)

- コマンドラインプロンプトで、delete-hsm コマンドを発行します。削除する HSM が含まれているクラスターの ID と、以下のいずれかの HSM 識別子を渡します。
  - HSM ID (`--hsm-id`)
  - HSM IP アドレス (`--eni-ip`)
  - HSM の Elastic Network Interface ID (`--eni-id`)

  これらの識別子の値がわからない場合は、describe-clusters コマンドを発行します。

```
$ aws cloudhsmv2 delete-hsm --cluster-id <cluster ID> --eni-ip <HSM IP address>
{
    "HsmId": "hsm-lgavqitns2a"
}
```

HSM (AWS CloudHSM API) を削除するには

- DeleteHsm リクエストを送信し、クラスター ID と削除する HSM の識別子を指定します。

AWS CloudHSM クラスターの削除

クラスターを削除する前に、すべての HSM をクラスターから削除する必要があります。詳細については、「HSM の削除」を参照してください。

すべての HSMs、AWS CloudHSM コンソール 、 ()、または API を使用してクラスターを削除できます。 AWS Command Line InterfaceAWS CLI AWS CloudHSM

クラスターを削除するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. 削除するクラスターを選択します。次に、[クラスターの削除] を選択します。

3. クラスターを削除することを確認し、[削除] を選択します。

クラスターを削除するには (AWS CLI)

- コマンドプロンプトで、delete-cluster コマンドを発行し、削除するクラスターの ID を渡します。クラスター ID がわからない場合は、describe-clusters コマンドを発行します。

```
$ aws cloudhsmv2 delete-cluster --cluster-id <cluster ID>
{
    "Cluster": {
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "SourceBackupId": "backup-rtq2dwi2gq6",
        "SecurityGroup": "sg-40399d28",
        "CreateTimestamp": 1504903546.035,
        "SubnetMapping": {
            "us-east-2a": "subnet-f1d6e798",
            "us-east-2c": "subnet-0e358c43",
            "us-east-2b": "subnet-40ed9d3b"
        },
        "ClusterId": "cluster-kdmrayrc7gi",
        "VpcId": "vpc-641d3c0d",
        "State": "DELETE_IN_PROGRESS",
        "HsmType": "hsm1.medium",
        "StateMessage": "The cluster is being deleted.",
        "Hsms": [],
        "BackupPolicy": "DEFAULT"
    }
}
```

AWS CloudHSM クラスターを削除するには (API)

- DeleteCluster リクエストを送信し、削除するクラスターの ID を指定します。

バックアップからの AWS CloudHSM クラスターの作成

バックアップから AWS CloudHSM クラスターを復元するには、このトピックの手順に従います。クラスターには、バックアップにあったものと同じユーザー、キーマテリアル、証明書、設定、および

ポリシーが含まれます。バックアップの管理に関する詳細については、バックアップの管理 を参照してください。

バックアップからのクラスターの作成 (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. [クラスターを作成] を選択します。
3. [Cluster configuration] セクションで、以下の操作を実行します。
   a. [VPC] で、作成するクラスターの VPC を選択します。
   b. [AZ(s)] で、クラスターに追加する各アベイラビリティーゾーンのプライベートサブネットを選択します。
4. [Cluster source] セクションで、以下の操作を行います。
   a. [Restore cluster from existing backup] を選択します。
   b. 復元するバックアップを選択します。
5. [次へ: レビュー] を選択します。
6. クラスター設定を確認し、[Create cluster] を選択します。
7. サービスがバックアップを保持する期間を指定します。

   デフォルトの保存期間である 90 日を受け入れるか、7 ~ 379 日の間に新しい値を入力します。このサービスは、ここで指定した値よりも古いこのクラスター内のバックアップを自動的に削除します。これは後で変更できます。詳細については、「バックアップの保持を設定」を参照してください。
8. [Next] (次へ) を選択します。
9. (オプション) タグキーとオプションのタグ値を入力します。クラスターに複数のタグを追加するには、タグの追加 を選択します。
10. [Review] (レビュー) を選択します。
11. クラスター設定を確認し、[Create cluster (クラスターの作成)] を選択します。

> **Tip**
>
> このクラスターに、復元したバックアップにあったのと同じユーザー、キーマテリアル、証明書、設定、ポリシーを含む HSM を作成するには、クラスターに HSM を追加します。

バックアップからのクラスターの作成 (AWS CLI)

バックアップ ID を判別するには、describe-backups コマンドを発行します。

- コマンドラインプロンプトで、create-cluster コマンドを発行します。HSM インスタンスタイプ、HSM を作成するサブネットのサブネット ID、および復元するバックアップのバックアップ ID を指定します。

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm1.medium \
                                --subnet-ids <subnet ID 1> <subnet ID 2> <subnet ID
 N> \
                                --source-backup-id <backup ID>
{
    "Cluster": {
        "HsmType": "hsm1.medium",
        "VpcId": "vpc-641d3c0d",
        "Hsms": [],
        "State": "CREATE_IN_PROGRESS",
        "SourceBackupId": "backup-rtq2dwi2gq6",
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
         },
        "SecurityGroup": "sg-640fab0c",
        "CreateTimestamp": 1504907311.112,
        "SubnetMapping": {
            "us-east-2c": "subnet-0e358c43",
            "us-east-2a": "subnet-f1d6e798",
            "us-east-2b": "subnet-40ed9d3b"
        },
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "ClusterId": "cluster-jxhlf7644ne"
    }
}
```

バックアップからクラスターを作成する (AWS CloudHSM API)

API を使用してバックアップからクラスターを作成する方法については、次のトピックを参照してください。

- CreateCluster

AWS CloudHSM バックアップの管理

AWS CloudHSM は、少なくとも 24 時間に 1 回クラスターの定期的なバックアップを作成します。各バックアップは、次のデータの暗号化されたコピーを含んでいます。

- ユーザー (CO、CU、および AU)
- キーマテリアルと証明書
- ハードウェアセキュリティモジュール (HSM) の設定とポリシー

バックアップを作成するようサービスに指示することはできませんが、特定の操作を行うことで強制的にバックアップを作成させることは可能です。以下のいずれかのアクションを実行すると、サービスがバックアップを作成します。

- クラスターをアクティベートするには
- HSM をアクティブクラスターに追加します。
- アクティブクラスターから HSM を削除

AWS CloudHSM は、クラスターの作成時に設定したバックアップ保持ポリシーに基づいてバックアップを削除します。バックアップ保持ポリシーの管理については、バックアップの保持を設定 を参照してください。

トピック

バックアップの使用

1 つ、または複数のアクティブな HSM が含まれていたクラスターに HSM を追加すると、サービスは最新のバックアップを新しい HSM に復元します。バックアップを使用して、使用頻度が少ない HSM を管理します。HSM が不要な場合は、それを削除してバックアップをトリガーします。後に、

その HSM が必要になったときに、同じクラスター内に新しい HSM を作成します。この操作により、以前に HSM の削除操作で作成したバックアップが復元されます。

有効期限切れのキー、または非アクティブなユーザーの削除

有効期限切れのキー、または非アクティブなユーザーなど、特定の暗号化マテリアルを環境から削除できます。これは 2 ステッププロセスです。まず、HSM からこれらのマテリアルを削除します。次に、既存のバックアップをすべて削除します。このプロセスに従うと、バックアップから新しいクラスターを初期化するときに、削除された情報が復元されないようになります。詳細については、「the section called “バックアップの削除と復元”」を参照してください。

ディザスタリカバリの検討

バックアップからクラスターを作成することができます。これは、クラスターのリカバリ・ポイントを設定するために行う場合があります。リカバリ・ポイントに必要なすべてのユーザー、キー・マテリアル、証明書を含むバックアップを指定し、そのバックアップを使用して新しいクラスターを作成します。バックアップからクラスタを作成する方法の詳細については、バックアップからクラスターを作成する を参照してください。

また、クラスターのバックアップを別のリージョンにコピーして、そこで元のクラスターのクローンとして新しいクラスターを作成することもできます。このプロセスは、さまざまな理由 (例: 災害対策プロセスの簡素化) で使用できます。バックアップをリージョンにコピーする方法の詳細については、リージョン間のバックアップのコピー を参照してください。

バックアップの削除と復元

バックアップを削除した後、サービスはバックアップを 7 日間保持し、その間にバックアップを復元することができます。7 日間の期間を過ぎると、バックアップを復元することはできなくなります。バックアップの管理の詳細については、バックアップの管理 を参照してください。

バックアップの削除と復元 (コンソール)

バックアップを削除するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. ナビゲーションペインで、[バックアップ] を選択します。
4. 削除するバックアップを選択します。

5. 選択したバックアップを削除するには、[アクション]、[削除] を選択します。

   [バックアップの削除] ダイアログボックスが表示されます。

6. [削除] を選択します。

   バックアップの状態が に変わりますPENDING_DELETE。削除をリクエストしてから最大 7 日間は、削除を保留しているバックアップを復元することができます。

バックアップを復元するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. ナビゲーションペインで、[バックアップ] を選択します。
4. PENDING_DELETE 復元する状態のバックアップを選択します。
5. 選択したバックアップを復元するには、[アクション]、[復元] を選択します。

バックアップの削除と復元 (AWS CLI)

バックアップのステータスを確認するか、またはバックアップの ID を検索するには describe-backups コマンドを AWS CLIから実行します。

バックアップを削除するには (AWS CLI)

- コマンドプロンプトで、delete-backup コマンドを実行し、削除するバックアップの ID を渡します。

```
$ aws cloudhsmv2 delete-backup --backup-id <backup ID>
{
    "Backup": {
        "CreateTimestamp": 1534461854.64,
        "ClusterId": "cluster-dygnwhmscg5",
        "BackupId": "backup-ro5c4er4aac",
        "BackupState": "PENDING_DELETION",
        "DeleteTimestamp": 1536339805.522,
        "HsmType": "hsm1.medium",
        "Mode": "FIPS"
    }
}
```

バックアップを復元する方法 (AWS CLI)

- バックアップを復元するには、restore-backup コマンドを発行し、PENDING_DELETION 状態のバックアップの ID を渡します。

```
$ aws cloudhsmv2 restore-backup --backup-id <backup ID>
{
    "Backup": {
        "ClusterId": "cluster-dygnwhmscg5",
        "CreateTimestamp": 1534461854.64,
        "BackupState": "READY",
        "BackupId": "backup-ro5c4er4aac"
    }
}
```

バックアップを一覧表示する方法 (AWS CLI)

- PENDING_DELETION の状態にあるすべてのバックアップのリストを表示するには、describe-backups コマンドを実行し、フィルタとして states=PENDING_DELETION を含めます。

```
$ aws cloudhsmv2 describe-backups --filters states=PENDING_DELETION
{
    "Backups": [
        {
            "BackupId": "backup-ro5c4er4aac",
            "BackupState": "PENDING_DELETION",
            "ClusterId": "cluster-dygnwhmscg5",
            "CreateTimestamp": 1534461854.64,
            "DeleteTimestap": 1536339805.522,
            "HsmType": "hsm2m.medium",
            "Mode": "NON_FIPS",
            "NeverExpires": false,
            "TagList": []
        }
}
```

バックアップの削除と復元 (AWS CloudHSM API)

API を使用してバックアップを削除および復元する方法については、次のトピックを参照してください。

- DeleteBackup
- RestoreBackup

AWS CloudHSM バックアップ保持ポリシーの設定

2020 年 11 月 18 日より前に作成されたクラスターは免除 されるため、クラスタのデフォルトのバックアップ保持ポリシーは90日です。この期間は 7 日から 379 日の間の任意の数値に設定できます。 AWS CloudHSM はクラスターの最後のバックアップを削除しません。バックアップの管理に関する詳細については、バックアップの管理 を参照してください。

バックアップ保持ポリシーについて

AWS CloudHSM は、クラスターの作成時に設定したバックアップ保持ポリシーに基づいてバックアップを消去します。バックアップ保持ポリシーは、クラスターに適用されます。バックアップを別のリージョンに移動すると、そのバックアップはクラスターに関連付けられなくなり、バックアップ保持ポリシーもなくなります。クラスターに関連付けられていないバックアップは手動で削除する必要があります。クラスターの最後のバックアップは削除 AWS CloudHSM されません。

AWS CloudTrail は、削除の対象となるバックアップを報告します。サービスが削除したバックアップは、手動で削除したバックアップ を復元する場合と同様に、復元できます。競合状態を回避するには、サービスによって削除されたバックアップを復元する前に、クラスターのバックアップ保持ポリシーを変更する必要があります。保持ポリシーを変更せず、選択したバックアップを保持したい場合は、クラスターのバックアップ保持ポリシーから バックアップを除外する ように指定できます。

既存のクラスター除外

AWS CloudHSM は、2020 年 11 月 18 日にマネージドバックアップ保持を開始しました。2020 年 11 月 18 日より前に作成されたクラスターには、90 日とクラスタの経過時間を足したバックアップ保持ポリシーが適用されます。例えば、2019 年 11 月 18 日にクラスターを作成した場合、サービスはクラスターに 1 年間と 90 日 (455 日) のバックアップ保持ポリシーを割り当てます。

> **Note**
>
> サポート (https://aws.amazon.com/support) に連絡すると、管理されたバックアップの保持を完全にオプトアウトできます。

バックアップ保持の設定 (コンソール)

バックアップ保持ポリシーを設定するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. アクティブ状態のクラスターのクラスター ID をクリックして、そのクラスターのバックアップ保持ポリシーを管理します。
4. バックアップ保持ポリシーを変更するには、[アクション]、[バックアップ保持期間の変更] を選択します。

   [バックアップ保持期間の変更] ダイアログボックスが表示されます。
5. バックアップ保持期間 (日単位) に、7 日 ~ 379 日の値を入力します。
6. [バックアップ保持期間の変更] を選択する。

バックアップ保持ポリシーからバックアップを除外または含めるには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. バックアップを表示するには、ナビゲーションペインで [バックアップ] を選択します。
3. [準備完了] 状態のバックアップのバックアップ ID をクリックして、除外または含めます。
4. リポジトリの バックアップの詳細 ページで、次のいずれかのアクションを実行します。
   - 有効期限 に日付があるバックアップを除外するには、[アクション]、[有効期限を無効にする] の順に選択します。
   - 有効期限のないバックアップを含めるには、[アクション]、[クラスタ保持ポリシーの使用] の順に選択します。

バックアップ保持の設定 (AWS CLI)

バックアップのステータスを確認するか、またはバックアップの ID を検索するには describe-backups コマンドを AWS CLIから実行します。

バックアップ保持ポリシーを設定するには (AWS CLI)

- コマンドラインプロンプトで、modify-cluster コマンドを発行します。クラスター ID とバックアップ保持ポリシーを指定します。

```
$ aws cloudhsmv2 modify-cluster --cluster-id <cluster ID> \
                                --backup-retention-policy  Type=DAYS,Value=<number
 of days to retain backups>
{
   "Cluster": {
      "BackupPolicy": "DEFAULT",
      "BackupRetentionPolicy": {
         "Type": "DAYS",
         "Value": 90
      },
      "Certificates": {},
      "ClusterId": "cluster-kdmrayrc7gi",
      "CreateTimestamp": 1504903546.035,
      "Hsms": [],
      "HsmType": "hsm1.medium",
      "SecurityGroup": "sg-40399d28",
      "State": "ACTIVE",
      "SubnetMapping": {
         "us-east-2a": "subnet-f1d6e798",
         "us-east-2c": "subnet-0e358c43",
         "us-east-2b": "subnet-40ed9d3b"
      },
      "TagList": [
         {
            "Key": "Cost Center",
            "Value": "12345"
         }
      ],
      "VpcId": "vpc-641d3c0d"
   }
}
```

バックアップ保持ポリシーからバックアップを除外するには (AWS CLI)

- コマンドラインプロンプトで、modify-backup-attributes コマンドを発行します。バックアップ ID を指定し、never-expires フラグを設定し、バックアップを保存します。

```
$ aws cloudhsmv2 modify-backup-attributes --backup-id <backup ID> \
                                          --never-expires
{
   "Backup": {
```

```
      "BackupId": "backup-ro5c4er4aac",
      "BackupState": "READY",
      "ClusterId": "cluster-dygnwhmscg5",
      "NeverExpires": true
   }
}
```

バックアップ保持ポリシーにバックアップを含めるには (AWS CLI)

- コマンドラインプロンプトで、modify-backup-attributes コマンドを発行します。バックアップ ID を指定し、バックアップ保持ポリシーにバックアップを含めるように no-never-expires フラグを設定します。つまり、サービスは最終的にバックアップを削除します。

```
$ aws cloudhsmv2 modify-backup-attributes --backup-id <backup ID> \
                                          --no-never-expires
{
   "Backup": {
      "BackupId": "backup-ro5c4er4aac",
      "BackupState": "READY",
      "ClusterId": "cluster-dygnwhmscg5",
      "NeverExpires": false
   }
}
```

バックアップ保持の設定 (AWS CloudHSM API)

API を使用してバックアップの保存管理を管理する方法については、次のトピックを参照してください。

- ModifyCluster
- ModifyBackupAttributes

AWS リージョン間でのバックアップのコピー

リージョン間の復元力、グローバルワークロード、および disaster recovery を含んださまざまな理由のために、リージョン間でバックアップをコピーできます。バックアップをコピーすると、それらはコピー先のリージョンに CREATE_IN_PROGRESS のステータスを用いて現れます。正常にコピーが実行されると、バックアップのステータスは READY に変わります。コピーが失敗した場合、バッ

クアップのステータスが DELETED に変わります。入力パラメータにエラーがないかどうかを確認し、オペレーションを再度実行する前に、指定した送信元バックアップのステータスが DELETED ではないことを確認します。バックアップ、あるいはバックアップからクラスターを作成する方法の詳細については、バックアップの管理 または バックアップからクラスターを作成する を参照してください。

次の点に注意してください。

- クラスターバックアップを送信先リージョンにコピーするには、適切な IAM ポリシーの許可がアカウントに必要です。バックアップを別のリージョンにコピーするには、バックアップがある送信元リージョンへのアクセスが IAM ポリシーで許可されている必要があります。リージョン間のコピーが完了したら、コピーしたバックアップを操作するために、IAM ポリシーを使用して、送信先リージョンへのアクセスを許可する必要があります。これには CreateCluster オペレーションの使用が含まれます。詳細については、「IAM 管理者の作成」を参照してください。
- 元のクラスターと、送信先リージョンのバックアップから作成された可能性のあるクラスターはリンクされません。これらのクラスターは別々に管理する必要があります。詳細については、「クラスターの管理」を参照してください。
- AWS 制限されたリージョンと標準リージョン間でバックアップをコピーすることはできません。バックアップは、 AWS GovCloud （ 米国東部) リージョンと AWS GovCloud （ 米国西部) リージョン間でコピーできます。

バックアップを異なるリージョン (コンソール) にコピーする

異なるリージョン (コンソール)にバックアップをコピーするには

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. ナビゲーションペインで、[バックアップ] を選択します。
4. バックアップを選択し、別のリージョンにコピーします。
5. 選択したバックアップをコピーするには、Actions, Copy backup to another region を選択します。

   [バックアップを別のリージョンにコピーする] ダイアログボックスが表示されます。
6. Destination region で、Select a region からリージョンを選択します。
7. (オプション) タグキーとオプションのタグ値を入力します。クラスターに複数のタグを追加するには、Add tag を選択します。

8. [Copy backup] (バックアップをコピー) を選択します。

バックアップを異なるリージョン (AWS CLI) にコピーする

バックアップ ID を判別するには、describe-backups コマンドを実行します。

異なるリージョン (AWS CLI) にバックアップをコピーする方法

- コマンドラインプロンプトで、 copy-backup-to-region コマンドを実行します。送信先のリージョンと、送信元バックアップのバックアップ ID を指定します。バックアップ ID を指定した場合は、関連付けられたバックアップがコピーされます。

```
$ aws cloudhsmv2 copy-backup-to-region --destination-region <destination region> \
                                       --backup-id <backup ID>
```

異なるリージョンへのバックアップのコピー (AWS CloudHSM API)

API を使用してバックアップを削除および復元する方法については、次のトピックを参照してください。

- CopyBackupToRegion

共有バックアップの使用

CloudHSM は AWS Resource Access Manager （ AWS RAM) と統合してリソース共有を有効にします。 は、一部の CloudHSM リソースを他の AWS アカウント と共有したり、 を介して共有したりできるサービス AWS RAM です AWS Organizations。では AWS RAM、リソース共有 を作成して、所有しているリソースを共有します。リソース共有は、共有するリソースと、それらを共有するコンシューマーを指定します。コンシューマーには以下が含まれます。

- の組織 AWS アカウント 内外の特定 AWS Organizations
- 内の組織内の組織単位 AWS Organizations
- の組織全体 AWS Organizations

の詳細については AWS RAM、「 AWS RAM ユーザーガイド」を参照してください。

このトピックでは、所有しているリソースの共有方法と、共有されているリソースの使用方法を説明します。

内容

バックアップを共有するための前提条件

- バックアップを共有するには、 でバックアップを所有している必要があります AWS アカウント。つまり、自分のアカウントにそのリソースが割り当てられているか、プロビジョニングされている必要があります。共有されているバックアップを共有することはできません。
- バックアップを共有するには、バックアップが READY 状態である必要があります。
- の組織または組織単位とバックアップを共有するには AWS Organizations、 との共有を有効にする必要があります AWS Organizations。詳細については、「AWS RAM ユーザーガイド」の「AWS Organizationsで共有を有効化する」を参照してください。

バックアップの共有

バックアップを他の と共有すると AWS アカウント、バックアップに保存されているキーとユーザーを含むクラスターをバックアップから復元できるようになります。

バックアップを共有するには、リソース共有に追加する必要があります。リソース共有とは、 AWS アカウント間で自身のリソースを共有するための AWS RAM リソースです。リソース共有では、共有対象のリソースと、共有先のコンシューマーを指定します。CloudHSM コンソールを使用してバックアップを共有する場合は、既存のリソース共有に追加します。新しいリソース共有にバックアップを追加するには、まずAWS RAM コンソール を使用してリソース共有を作成する必要があります。

ユーザーが の組織に属 AWS Organizations していて、組織内での共有が有効になっている場合、組織内のコンシューマーには共有バックアップへのアクセス許可が自動的に付与されます。それ以外の

場合、コンシューマーはリソース共有への参加の招待を受け取り、招待を承諾すると共有バックアップへのアクセスが許可されます。

AWS RAM コンソールまたは を使用して、所有しているバックアップを共有できます AWS CLI。

AWS RAM コンソールを使用して所有しているバックアップを共有するには

「AWS RAM ユーザーガイド 」の「リソース共有の作成」を参照してください。

所有しているバックアップを共有するには (AWS RAM コマンド)

create-resource-shareコマンドを実行します。

所有しているバックアップを共有するには (CloudHSM コマンド)

> **Important**
>
> CloudHSM PutResourcePolicy オペレーションを使用してバックアップを共有できますが、代わりに (AWS RAM) を使用する AWS Resource Access Manager ことをお勧めします。を使用すると AWS RAM 、ポリシーが作成され、複数のリソースが一度に共有され、共有リソースの検出可能性が向上するため、複数の利点が得られます。を使用して PutResourcePolicy いて、コンシューマーが共有したバックアップを記述できるようにする場合は、 AWS RAM PromoteResourceShareCreatedFromPolicy API オペレーションを使用してバックアップを標準の AWS RAM リソース共有に昇格させる必要があります。

put-resource-policyコマンドを実行します。

1. という名前のファイルpolicy.jsonを作成し、次のポリシーをコピーします。

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS":"<consumer-aws-account-id-or-user>"
    },
    "Action":[
      "cloudhsm:CreateCluster",
      "cloudhsm:DescribeBackups"],
    "Resource":"<arn-of-backup-to-share>"
  }]
```

```
}
```

2. バックアップ ARN と識別子policy.jsonを使用して更新し、共有します。次の例では、123456789012 で識別される AWS アカウントのルートユーザーに読み取り専用アクセスを許可します。

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS": [
        "account-id"
      ]
    },
    "Action":[
      "cloudhsm:CreateCluster",
      "cloudhsm:DescribeBackups"],
    "Resource":"arn:aws:cloudhsm:us-west-2:123456789012:backup/backup-123"
  }]
}
```

> **Important**
>
> アクセス許可は、アカウントレベルでのみ DescribeBackups に付与できます。バックアップを別の顧客と共有する場合、そのアカウントで DescribeBackups アクセス許可を持つプリンシパルはバックアップを記述できます。

3. put-resource-policy コマンドを実行します。

```
$ aws cloudhsmv2 put-resource-policy --resource-arn <resource-arn> --policy file://
policy.json
```

> **Note**
>
> この時点で、コンシューマーはバックアップを使用できますが、共有パラメータを使用して DescribeBackups レスポンスに表示されません。次のステップでは、バックアップをレスポンスに含めるために AWS RAM リソース共有を昇格させる方法について説明します。

4. AWS RAM リソース共有 ARN を取得します。

```
$ aws ram list-resources --resource-owner SELF --resource-arns <backup-arn>
```

これにより、次のようなレスポンスが返されます。

```
{
  "resources": [
    {
      "arn": "<project-arn>",
      "type": "<type>",
      "resourceShareArn": "<resource-share-arn>",
      "creationTime": "<creation-time>",
      "lastUpdatedTime": "<last-update-time>"
    }
  ]
}
```

レスポンスから、次のステップで使用する *#resource-share-arn#* 値をコピーします。

5. AWS RAM promote-resource-share-created-from-policy コマンドを実行します。

```
$ aws ram promote-resource-share-created-from-policy --resource-share-
arn <resource-share-arn>
```

6. リソース共有が昇格したことを検証するには、 コマンドを実行します AWS RAM get-resource-shares 。

```
$ aws ram get-resource-shares --resource-owner SELF --resource-share-
arns <resource-share-arn>
```

ポリシーが昇格されると、レスポンスにfeatureSetリストされている は になりますSTANDARD。つまり、バックアップは、ポリシー内の新しいアカウントで記述できるということです。

共有バックアップの共有解除

リソースの共有を解除すると、コンシューマーはそのリソースを使用してクラスターを復元できなくなります。コンシューマーは、共有バックアップから復元したクラスターに引き続きアクセスできます。

所有している共有バックアップの共有を解除するには、リソース共有から削除する必要があります。これは、 AWS RAM コンソールまたは を使用して実行できます AWS CLI。

AWS RAM コンソールを使用して所有している共有バックアップの共有を解除するには

「AWS RAM ユーザーガイド」の「リソース共有の更新」を参照してください。

所有している共有バックアップの共有を解除するには (AWS RAM コマンド)

disassociate-resource-shareコマンドを実行します。

所有している共有バックアップの共有を解除するには (CloudHSM コマンド)

delete-resource-policyコマンドを実行します。

```
$ aws cloudhsmv2 delete-resource-policy --resource-arn <resource-arn>
```

共有バックアップの特定

コンシューマーは、CloudHSM コンソールと を使用して、共有されているバックアップを特定できます AWS CLI。

CloudHSM コンソールを使用して共有されているバックアップを特定するには

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. を変更するには AWS リージョン、ページの右上隅にあるリージョンセレクタを使用します。
3. ナビゲーションペインで、[バックアップ] を選択します。
4. テーブルで、共有バックアップタブを選択します。

を使用して共有されているバックアップを特定するには AWS CLI

describe-backups コマンドと `--shared`パラメータを使用して、共有されているバックアップを返します。

共有バックアップのアクセス許可

所有者のアクセス許可

バックアップ所有者は、共有バックアップを記述および管理したり、クラスターの復元に使用できます。

コンシューマーのアクセス許可

バックアップコンシューマーは共有バックアップを変更できませんが、それを記述してクラスターの復元に使用できます。

請求と使用量測定

バックアップの共有には追加料金はかかりません。

AWS CloudHSM リソースのタグ付け

タグは、 AWS リソースに割り当てるラベルです。 AWS CloudHSM クラスターにタグを割り当てることができます。各タグは 1 つのタグキーと 1 つのタグ値で構成されており、いずれもお客様が定義します。たとえば、タグキーは Cost Center、タグ値は 12345 などです。タグキーは、クラスターごとに一意にする必要があります。

タグは、さまざまな目的で使用できます。1 つの一般的な用途は、 AWS コストの分類と追跡です。自社のカテゴリ たとえばコストセンター、アプリケーション名、所有者を表すタグを適用すると、複数のサービスにわたってコストを分類することができます。 AWS リソースにタグを追加すると、は使用量とコストをタグ別に集計したコスト配分レポート AWS を生成します。このレポートを使用すると、すべての AWS CloudHSM コストを 1 つの明細項目として表示するのではなく、プロジェクトまたはアプリケーションの観点から AWS CloudHSM コストを表示できます。

タグを使用したコスト配分の詳細については、「AWS Billing ユーザーガイド」の「コスト配分タグの使用」を参照してください。

タグの追加、更新、一覧表示、削除を行うには、AWS CloudHSM コンソール、あるいは AWS SDK かコマンドラインツールのどちらかを使用できます。

トピック

タグの追加または更新するには

タグの追加または更新は、AWS CloudHSM コンソール、AWS Command Line Interface (AWS CLI)、または AWS CloudHSM API で行うことができます。

タグを追加または更新するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. タグ付けするクラスターを選択します。
3. [タグ] を選択します。
4. タグを追加するには、次の操作を行います。

a. [タグの編集]、[タグの追加] の順に選択します。

b. [Key (キー)] にタグのキーを入力します。

c. (オプション) [Value (値)] にタグの値を入力します。

d. [保存] を選択します。

5. タグを更新するには、次の操作を行います。

a. [タグの編集] を選択します。

> **Note**
>
> 既存のタグのタグキーを更新すると、コンソールによって既存のタグが削除され、新しいタグが作成されます。

b. 新しいタグ値を入力します。

c. [保存] を選択します。

タグを追加または更新するには (AWS CLI)

1. コマンドプロンプトで、tag-resource コマンドを発行し、タグとタグ付けするクラスターの ID を指定します。クラスター ID がわからない場合は、describe-clusters コマンドを発行します。

```
$ aws cloudhsmv2 tag-resource --resource-id <cluster ID> \
                              --tag-list Key="<tag key>",Value="<tag value>"
```

2. タグを更新するには、同じコマンドを使用しますが、既存のタグキーを指定します。既存のタグに新しいタグ値を指定すると、タグは新しい値で上書きされます。

タグを追加または更新するには (AWS CloudHSM API)

- TagResource リクエストを送信します。タグとタグ付けするクラスターの ID を指定します。

タグを一覧表にする

クラスターのタグは、AWS CloudHSM コンソール 、、AWS CLIまたは AWS CloudHSM API から一覧表示できます。

タグを一覧表示するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. タグを一覧表示するクラスターを選択します。
3. [タグ] を選択します。

タグを一覧表示するには (AWS CLI)

- コマンドプロンプトで、list-tags コマンドを発行し、タグを一覧表示するクラスターの ID を指定します。クラスター ID がわからない場合は、describe-clusters コマンドを発行します。

```
$ aws cloudhsmv2 list-tags --resource-id <cluster ID>
{
    "TagList": [
        {
            "Key": "Cost Center",
            "Value": "12345"
        }
    ]
}
```

タグを一覧表示するには (AWS CloudHSM API)

- ListTags リクエストを送信し、タグを一覧表示するクラスターの ID を指定します。

タグの削除

クラスターからタグを削除するには、AWS CloudHSM コンソール、AWS CLI、または AWS CloudHSM API を使用できます。

タグを削除するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. タグを削除するクラスターを選択します。
3. [タグ] を選択します。
4. [タグの編集] を選択し、削除するタグの [タグの削除] を選択します。
5. [保存] を選択します。

タグを削除するには (AWS CLI)

- コマンドプロンプトで untag-resource コマンドを発行し、削除するタグのタグキーと、タグを削除するクラスターの ID を指定します。を使用してタグ AWS CLI を削除する場合は、タグ値ではなくタグキーのみを指定します。

```
$ aws cloudhsmv2 untag-resource --resource-id <cluster ID> \
                                --tag-key-list "<tag key>"
```

タグを削除するには (AWS CloudHSM API)

- AWS CloudHSM API でUntagResourceリクエストを送信し、クラスターの ID と削除するタグを指定します。

での HSM ユーザーとキーの管理 AWS CloudHSM

AWS CloudHSM クラスターを暗号化処理に使用する前に、クラスター内の HSMs にユーザーとキーを作成する必要があります。 AWS CloudHSMの HSM ユーザーとキーの管理の詳細ついては、次のトピックを参照してください。クォーラム認証 (M of N アクセスコントロールとも呼ばれます) の使用方法も学習できます。

トピック

- での HSM ユーザーの管理 AWS CloudHSM
- でのキーの管理 AWS CloudHSM
- クローンされたクラスターを管理する

での HSM ユーザーの管理 AWS CloudHSM

では AWS CloudHSM、CloudHSM CLI または CloudHSM 管理ユーティリティ (CMU) コマンドラインツールを使用して、HSM でユーザーを作成および管理する必要があります。CloudHSM CLI は最新の SDK バージョンシリーズ で使用するように設計されていますが、CMU は 以前の SDK バージョンシリーズ で使用するように設計されています。

トピック

- CloudHSM CLI による HSM ユーザーの管理
- CloudHSM 管理ユーティリティ (CMU) を使用する HSM ユーザーを管理する

CloudHSM CLI による HSM ユーザーの管理

CloudHSM CLI コマンドラインツールを使用して、最新の SDK を使用して HSM でユーザーを作成および管理できます。

トピック

- HSM ユーザーを理解する
- HSM ユーザーの許可テーブル
- CloudHSM CLI を使用してユーザーを管理する
- CloudHSM CLI を使用して MFA を管理する
- CloudHSM CLI を使用したクォーラム認証の管理 (M of N のアクセスコントロール)

HSM ユーザーを理解する

HSM 上で実行するほとんどのオペレーションでは、HSM ユーザーの認証情報が必要です。HSM は各 HSM ユーザーを認証し、各 HSM ユーザーに設定されている タイプ により、ユーザーとして HSM で実行できるオペレーションが決定されます。

> **Note**
>
> HSM ユーザーは IAM ユーザーとは異なります。正しい認証情報を持つ IAM ユーザーは、AWS API を介してリソースを操作することで HSM を作成できます。HSM を作成したら、HSM ユーザー認証情報を使用して HSM でのオペレーションを認証する必要があります。

ユーザータイプ

- 非アクティブ管理者
- 管理
- Crypto user (CU)
- Appliance user (AU)

非アクティブ管理者

CloudHSM CLI では、非アクティブ化された管理者は、 AWS CloudHSM クラスター内のアクティブ化されたことのない最初の HSM にのみ存在する一時的なユーザーです。クラスターをアクティブ化する には、CloudHSM CLI で cluster activate コマンドを実行します。このコマンドを実行すると、非アクティブ化された管理者にはパスワードの変更を求めるプロンプトが表示されます。パスワードを変更すると、非アクティブ化された管理者は管理者になります。

管理

CloudHSM CLI では、管理者はユーザー管理オペレーションを実行できます。たとえば、ユーザーの作成および削除と、ユーザーパスワードの変更を行うことなどができます。管理者の詳細については、「HSM ユーザーの許可テーブル」を参照してください。

Crypto user (CU)

暗号化ユーザー (CU) は、以下のキー管理および暗号化のオペレーションを行うことができます。

- キー管理 - 暗号化キーの作成、削除、共有、インポート、エクスポートを行います。
- 暗号化オペレーション - 暗号化キーを使用して、暗号化、復号、署名、検証などを行います。

詳細については、「HSM ユーザーの許可テーブル」を参照してください。

Appliance user (AU)

アプライアンスユーザー (AU) は、クラスターの HSMs。 は AU AWS CloudHSM を使用して、AWS CloudHSM クラスター内の HSMsを同期します。AU は、 によって提供されるすべての HSMs に存在し AWS CloudHSM、アクセス許可が制限されています。詳細については、「HSM ユーザーの許可テーブル」を参照してください。

AWS はHSMs でオペレーションを実行できません。ユーザーまたはキーを表示または変更 AWS することはできません。また、これらのキーを使用して暗号化オペレーションを実行することはできません。

HSM ユーザーの許可テーブル

以下の表は、オペレーションを実行できる HSM ユーザーまたはセッションのタイプ別にソートされた HSM オペレーションリストです。

	管理	Crypto User (CU)	Appliance User (AU)	未認証セッション
基本的なクラスター情報を取得する[1]	はい	はい	はい	はい
自分のパスワードを変更する	はい	はい	はい	該当しない
ユーザーのパスワードを変更する	はい	いいえ	いいえ	いいえ

	管理	Crypto User (CU)	Appliance User (AU)	未認証セッション
ユーザーを追加、削除する	 はい	 いいえ	 いいえ	 いいえ
同期のステータスを取得する[2]	 はい	 はい	 はい	 いいえ
マスクされたオブジェクトを抽出、挿入する[3]	 はい	 はい	 はい	 いいえ
キー管理機能[4]	 いいえ	 はい	 いいえ	 いいえ
暗号化、復号する	 いいえ	 はい	 いいえ	 いいえ
署名、検証する	 いいえ	 はい	 いいえ	 いいえ
ダイジェストと HMAC の生成	 いいえ	 はい	 いいえ	 いいえ

- [1] 基本情報には、クラスター内の HSM 数、各 HSM の IP アドレス、モデル、シリアル番号、デバイス ID、ファームウェア ID などが含まれます。
- [2] ユーザーは、HSM のキーに対応するダイジェスト (ハッシュ) のセットを取得できます。アプリケーションは、これらのダイジェストのセットを比較して、クラスター内の HSM の同期状態を把握します。
- [3] マスクされたオブジェクトは、HSM を離れる前に暗号化されるキーです。これらのオブジェクトを HSM の外部で復号することはできません。これらは、抽出された HSM と同じクラスターにある HSM に挿入された後にのみ復号されます。アプリケーションはマスクされたオブジェクトを抽出して挿入し、クラスター内の HSM を同期します。
- [4] キー管理機能には、キーの属性の作成、削除、ラップ、ラップ解除、変更が含まれます。

CloudHSM CLI を使用してユーザーを管理する

このトピックでは、CloudHSM CLI でハードウェアセキュリティモジュール (HSM) ユーザーを管理する step-by-step 方法について説明します。CloudHSM CLI または HSM ユーザーの詳細情報は、「CloudHSM CLI」と「CloudHSM CLI を使用する」を参照してください。

セクション

- CloudHSM CLI を使用した HSM ユーザー管理について
- CloudHSM CLI をダウンロード
- CloudHSM ユーザーを管理する方法CLI

CloudHSM CLI を使用した HSM ユーザー管理について

HSM ユーザーを管理するには、管理者のユーザー名とパスワードを使用して HSM にログインする必要があります。管理者のみユーザーを管理できます。HSM には、admin という名前のデフォルト管理者が含まれています。admin クラスターのアクティブ化 の際に必要なパスワードを設定しました。

CloudHSM CLI を使用する場合、設定ツールでローカル設定を更新する必要があります。CloudHSM CLI で設定ツールを実行する手順については、「CloudHSM コマンドラインインターフェイス (CLI) の使用開始」を参照してください。-a パラメータには、クラスター内の HSM の IP アドレスを追加する必要があります。複数の HSM をお持ちの方は、任意の IP アドレスを使用できます。これで確実に CloudHSM CLI がクラスター全体に加えた変更を伝播できます。CloudHSM CLI はローカルファイルを使用してクラスター情報を追跡することに注意してください。特定のホストから CloudHSM CLI を最後に使用後にクラスターが変更されている場合、該当するホストに保存されてい

るローカル設定ファイルにそれらの変更を追加する必要があります。CloudHSM CLI を使用している間は、HSM を削除しないでください。

HSM の IP アドレスを取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. クラスターの詳細ページを開くには、クラスターテーブルでクラスター ID を選択します。
4. IP アドレスを取得するには、[HSM] タブで、[ENI IP アドレス] にリストされている IP アドレスのいずれかを選択します。

HSM の IP アドレスを取得するには (AWS CLI)

- describe-clusters から AWS CLIコマンドを実行して、HSM の IP アドレスを取得します。コマンドの出力では、HSM の IP アドレスは `EniIp` の値です。

```
$  aws cloudhsmv2 describe-clusters


{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
...
```

CloudHSM CLI をダウンロード

CloudHSM CLI の最新バージョンは、Client SDK 5 の HSM ユーザー管理タスクに使用できます。CloudHSM CLI をダウンロードしてインストールするには、「CloudHSM CLI のインストールと設定」の指示に従ってください。

CloudHSM ユーザーを管理する方法CLI

このセクションでは、CloudHSM CLI を使用した HSM ユーザーを管理用の基本的なコマンドについて説明します。

> **Note**
>
> 注: CloudHSM CLI のユーザーコマンドは、CloudHSM CLI ユーザーコマンドリファレンスに記載されています

トピック

管理者を作成するには

管理者を作成するには、以下の手順に従います。

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてクラスターにログインします。

   ```
   aws-cloudhsm > login --username <USERNAME> --role admin
   ```

3. システムからパスワードの入力を求められます。パスワードを入力すると、出力はコマンドが成功したことを表示します。

```
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

4. 次のコマンドを入力して管理者を作成します。

```
aws-cloudhsm > user create --username <USERNAME> --role admin
```

5. 新しいユーザーのパスワードを入力します。
6. パスワードを再入力して、入力したパスワードが正しいことを確認します。

Crypto User の作成

ユーザーを作成するには、次の手順に従います。

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてクラスターにログインします。

```
aws-cloudhsm > login --username <USERNAME> --role admin
```

3. システムからパスワードの入力を求められます。パスワードを入力すると、出力はコマンドが成功したことを表示します。

```
Enter password:
{
  "error_code": 0,
```

```
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

4. 以下のコマンドを入力して、crypto user を作成します。

```
aws-cloudhsm > user create --username <USERNAME> --role crypto-user
```

5. 新しい crypto user のパスワードを入力します。
6. パスワードを再入力して、入力したパスワードが正しいことを確認します。

クラスター上のすべての HSM ユーザーを一覧表示

user list コマンドを使用して、クラスター上のすべてのユーザーを一覧表示します。user list を実行するのに、ログインする必要はありません。すべてのユーザータイプでユーザーを一覧表示できます。

クラスター内のすべてのユーザーを一覧表示するには、次の手順に従います

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. クラスター内のすべてのユーザーを一覧表示するには、以下のコマンドを入力します。

```
aws-cloudhsm > user list
```

user list の詳細については、ユーザーリスト を参照してください。

HSM ユーザーパスワードの変更

user change-password コマンドを使用してパスワードを変更します。

ユーザータイプとパスワードは大文字と小文字が区別されますが、ユーザー名では区別されません。

管理者、Crypto User (CU)、Appliance User (AU) は、自分のパスワードを変更できます。別のユーザーのパスワードを変更する場合、管理者としてログインする必要があります。ただし、現在ログインしているユーザーのパスワードを変更することはできません。

自分のパスワードの変更

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、変更するパスワードを使用するユーザーとしてログインします。

   ```
   aws-cloudhsm > login --username <USERNAME> --role <ROLE>
   ```

3. ユーザーのパスワードを入力します。

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin1",
       "role": "admin"
     }
   }
   ```

4. user change-password コマンドを入力します。

   ```
   aws-cloudhsm > user change-password --username <USERNAME> --role <ROLE>
   ```

5. 新しいパスワードを入力します。
6. 新しいパスワードを再入力します。

別のユーザーのパスワードの変更

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてログインします。

   ```
   aws-cloudhsm > login --username <USERNAME> --role admin
   ```

3. 管理者のパスワードを入力します。

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin1",
       "role": "admin"
     }
   }
   ```

4. パスワードを変更するユーザーのユーザー名とともに user change-password コマンドを入力します。

   ```
   aws-cloudhsm > user change-password --username <USERNAME> --role <ROLE>
   ```

5. 新しいパスワードを入力します。
6. 新しいパスワードを再入力します。

user change-password については、「user change-password」を参照してください。

HSM ユーザーの削除

user delete を使用してユーザーを削除します。別のユーザーを削除する場合、管理者としてログインする必要があります。

> **Tip**
>
> キーを所有している暗号化ユーザー (CU) を削除することはできません。

ユーザーの削除

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてクラスターにログインします。

   ```
   aws-cloudhsm > login --username <USERNAME> --role admin
   ```

3. システムからパスワードの入力を求められます。パスワードを入力すると、出力はコマンドが成功したことを表示します。

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin",
       "role": "admin"
     }
   }
   ```

4. ユーザーを削除するには、user delete コマンドを使用します。

   ```
   aws-cloudhsm > user delete --username <USERNAME> --role <ROLE>
   ```

user delete の詳細情報は、deleteUser を参照してください。

CloudHSM CLI を使用して MFA を管理する

セキュリティを強化するために、クラスターの保護に役立つ多要素認証 (MFA) を構成することができます。詳細については、以下のトピックを参照してください。

トピック

HSM ユーザー用 MFA について

MFA が有効な HSM ユーザーアカウントを使用してクラスターにログインする場合、CloudHSM CLI とパスワード (最初の要素、既知の情報) を提供すると、CloudHSM CLI によってトークンが提供され、トークンに署名するよう求められます。

2 番目の要素 (自分の持っているもの) を提供する場合、すでに作成して HSM ユーザーに紐づけしたキーペアからプライベートキーを使用してトークンに署名します。クラスターにアクセスする場合、署名付きトークンを CloudHSM CLI に指定します。

ユーザーの MFA 設定の詳細については、「CloudHSM CLI 用の MFA のセットアップ」を参照してください

クォーラム認証と MFA

クラスターは、クォーラム認証と MFA に同じキーを使用します。これは、MFA が有効になっているユーザーが実質的に MofN または Quroum アクセス制御に登録されていることを意味します。同じ HSM ユーザーに対して MFA 認証とクォーラム認証を正常に実行する際、次の点を考慮する必要があります。

- 現在、ユーザーに対してクォーラム認証を使用している場合は、クォーラムのユーザーに対して作成したものと同じキーペアを使用し、ユーザーに対して MFA を有効化する必要があります。
- クォーラム認証ユーザーではない非 MFA ユーザーに MFA 要件を追加する場合は、そのユーザーを MFA 認証を備えた Quroum (MofN) 登録ユーザーとして登録します。
- MFA 要件を削除するか、クォーラム認証ユーザーでもある MFA ユーザーのパスワードを変更する場合、クォーラム (MofN) ユーザーとしてそのユーザーの登録も削除されます。
- MFA 要件を削除するか、クォーラム認証ユーザーでもある MFA ユーザーのパスワードを変更する場合、それでもそのユーザーがクォーラム認証に加わる必要がある場合、当該ユーザーをクォーラム (MofN) ユーザーとして再登録する必要があります。

認証の詳細情報は、「クォーラム (M of N) の管理」を参照してください。

HSM ユーザーに対する MFA を使用した作業

このトピックでは、CloudHSM CLI を使用して多要素認証 (MFA) を管理するための情報と手順について説明します。CloudHSM CLI の詳細については、「CloudHSM コマンドラインインターフェイス (CLI)」を参照してください。

トピック

- MFA キーペアの要件
- CloudHSM CLI 用の MFA のセットアップ
- MFA を有効にしたユーザーの作成
- MFA を有効にしたユーザーのログイン
- MFA が有効になっているユーザーのキーをローテーションする
- MFA パブリックキーが登録されたら、管理者ユーザーの MFA パブリックキーの登録を解除する
- トークンファイルのリファレンス

HSM ユーザーの操作方法の詳細については、「CloudHSM コマンドラインインターフェイス (CLI)」を参照してください

MFA キーペアの要件

HSM ユーザーの MFA を有効にするには、新しいキーペアを作成するか、次の要件を満たす既存のキーを使用します。

- キータイプ : 非対称
- キーの使用方法 : 署名と検証
- キースペック: RSA_2048
- 署名アルゴリズムには以下が含まれます。 sha256WithRSAEncryption

> **Note**
>
> クォーラム認証を使用している場合、またはクォーラム認証を使用する予定の場合は、「クォーラム認証と MFA」を参照してください

CloudHSM CLI とキーペアを使用して、MFA を有効化した新しい管理者ユーザーを作成できます。

CloudHSM CLI 用の MFA のセットアップ

CloudHSM CLI 用の MFA をセットアップするには、次の手順を行います。

1. トークン署名戦略を使用して MFA をセットアップするには、まず 2048 ビットの RSA プライベートキーと関連するパブリックキーを生成する必要があります。

```
$ openssl genrsa -out officer1.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................+++++
.....................................................................+++++
e is 65537 (0x010001)

$ openssl rsa -in officer1.key -outform PEM -pubout -out officer1.pub
writing RSA key
```

2. CloudHSM CLI を使用して、ユーザーアカウントにログインします。

```
$ cloudhsm-cli interactive
aws-cloudhsm > login --username admin --role admin --cluster-id <cluster ID>
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

3. 次に、コマンドを実行して MFA ストラテジーを変更します。パラメーター --token を指定する必要があります。このパラメータは、署名されていないトークンが書き込まれるファイルを指定します。

```
aws-cloudhsm > user change-mfa token-sign --token unsigned-tokens.json --
username <USERNAME> --role crypto-user --change-quorum
Enter password:
Confirm password:
```

4. これで、署名が必要な未署名のトークンを含むファイルが作成されました: unsigned-tokens.json。このファイル内のトークンの数は、クラスター内の HSM の数によって異な

ります。各トークンは 1 つの HSM を表します。このファイルは JSON 形式で、プライベートキーを持っていることを証明するための署名が必要なトークンが含まれています。

```
$ cat unsigned-tokens.json
{
  "version": "2.0",
  "tokens": [
{
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
  ]
}
```

5. 次のステップは、ステップ 1 で作成したプライベートキーを使用してこれらのトークンに署名することです。署名をファイルに戻します。まず、base64 でエンコードされたトークンを抽出してデコードする必要があります。

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

6. これで、ステップ 1 で作成した RSA プライベートキーを使用して署名できるバイナリトークンができました。

```
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
```

```
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token3.bin \
      -out token3.sig.bin
```

7. これで、トークンのバイナリ署名ができました。Base64 を使用してエンコードし、トークンファイルに戻す必要があります。

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

8. 最後に、base64 値をコピーしてトークンファイルに貼り付けます。

```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
```

```
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
  ]
}
```


9. トークンファイルに必要な署名がすべて揃ったので、次に進むことができます。署名されたトークンを含むファイルの名前を入力し、Enter キーを押します。最後に、パブリックキーのパスを入力します。

```
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:officer1.pub
{
  "error_code": 0,
  "data": {
    "username": "<USERNAME>",
    "role": "crypto-user"
  }
}
```

これで、MFA を使用してユーザーを設定できました。

```
{
    "username": "<USERNAME>",
    "role": "crypto-user",
    "locked": "false",
    "mfa": [
      {
        "strategy": "token-sign",
        "status": "enabled"
      }
    ],
    "cluster-coverage": "full"
```

```
},
```

MFA を有効にしたユーザーの作成

次の手順に従って、MFA を有効にしたユーザーを作成します。

1. CloudHSM CLI を使用し、管理者として HSM にログインします。
2. user create コマンドを使用して、任意のユーザーを作成します。次に、CloudHSM CLI 用の MFA のセットアップ の手順に従ってユーザーの MFA を設定します。

MFA を有効にしたユーザーのログイン

次の手順に従って、MFA が有効になっているユーザーをログインさせます。

1. CloudHSM CLI の login mfa-token-sign コマンドを使用して、MFA が有効になっているユーザーのログインプロセスを MFA で開始します。

```
aws-cloudhsm > login --username <USERNAME> --role <ROLE> mfa-token-sign --token
 unsigned-tokens.json
Enter password:
```

2. パスワードを入力します。次に、署名されていないトークンと署名されたトークンのペアを含むトークンファイルへのパスを入力するよう求められます。ここで、署名付きトークンは、プライベートキーを使用して生成されたものです。

```
aws-cloudhsm > login --username <USERNAME> --role <ROLE> mfa-token-sign --token
 unsigned-tokens.json
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
```

3. 署名済みトークンのファイルパスを入力するように求められますが、別の端末で署名されていないトークンファイルを調べることができます。署名が必要な未署名のトークンを含むファイルを特定します: unsigned-tokens.json。このファイル内のトークンの数は、クラスター内の HSM の数によって異なります。各トークンは 1 つの HSM を表します。このファイルは JSON 形式で、プライベートキーを持っていることを証明するための署名が必要なトークンが含まれています。

```
$ cat unsigned-tokens.json
{
```

```
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
  ]
}
```

4. ステップ 2 で作成したプライベートキーを使用して、署名されていないトークンに署名します。まず、base64 でエンコードされたトークンを抽出してデコードする必要があります。

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

5. これで、バイナリトークンが作成されました。MFA セットアップのステップ 1 で作成した RSA プライベートキーを使用して署名します。

```
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
```

```
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token3.bin \
      -out token3.sig.bin
```

6. これで、トークンのバイナリ署名ができました。base64 を使用してエンコードし、トークンファイルに戻します。

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

7. 最後に、base64 値をコピーしてトークンファイルに貼り付けます。

```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
```

```
  ]
}
```

8. トークンファイルに必要な署名がすべて揃ったので、次に進むことができます。署名されたトークンを含むファイルの名前を入力し、Enter キーを押します。これで、正常にログインできたはずです。

```
aws-cloudhsm > login --username <USERNAME> --role <ROLE> mfa-token-sign --token
 unsigned-tokens.json
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
{
  "error_code": 0,
  "data": {
    "username": "<USERNAME>",
    "role": "<ROLE>"
  }
}
```

MFA が有効になっているユーザーのキーをローテーションする

MFA が有効になっているユーザーのキーをローテーションするには、次の手順に従います。

<result>

生成された JSON 形式のトークンファイルにプライベートキーで署名し、新しい MFA パブリックキーを登録しました。
</result>

1. CloudHSM CLI を使用して、任意の管理者または MFA が有効になっている特定のユーザーとして HSM にログインします (詳細については、「MFA が有効になっているユーザーのログイン」を参照)。
2. 次に、コマンドを実行して MFA ストラテジーを変更します。パラメーター --token を指定する必要があります。このパラメータは、署名されていないトークンが書き込まれるファイルを指定します。

```
aws-cloudhsm > user change-mfa token-sign --token unsigned-tokens.json --
username <USERNAME> --role crypto-user --change-quorum
Enter password:
Confirm password:
```

3. 署名が必要な未署名のトークンを含むファイルを特定します: unsigned-tokens.json。このファイル内のトークンの数は、クラスター内の HSM の数によって異なります。各トークンは 1 つの HSM を表します。このファイルは JSON 形式で、プライベートキーを持っていることを証明するための署名が必要なトークンが含まれています。これは、現在登録されているパブリックキーのローテーションに使用したい新しい RSA パブリック/プライベートキーペアからの新しいプライベートキーになります。

```
$cat unsigned-tokens.json
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
  ]
}
```

4. これらのトークンには、セットアップ時に作成したプライベートキーで署名します。まず、base64 でエンコードされたトークンを抽出してデコードする必要があります。

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

5. これで、バイナリトークンが作成されました。セットアップ時に作成した RSA プライベートキーを使用して署名します。

```
$ openssl pkeyutl -sign \
```

```
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token3.bin \
      -out token3.sig.bin
```

6. これで、トークンのバイナリ署名ができました。base64 を使用してエンコードし、トークンファイルに戻します。

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

7. 最後に、base64 値をコピーしてトークンファイルに貼り付けます。

```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
```

```
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
  ]
}
```

8. トークンファイルに必要な署名がすべて揃ったので、次に進むことができます。署名されたトークンを含むファイルの名前を入力し、Enter キーを押します。最後に、新しいパブリックキーのパスを入力します。これで、user listの出力の一部として、次の内容が表示されます。

```
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:officer1.pub
{
  "error_code": 0,
  "data": {
    "username": "<USERNAME>",
    "role": "crypto-user"
  }
}
```

これで、ユーザーに MFA を設定できました。

```
{
    "username": "<USERNAME>",
    "role": "crypto-user",
    "locked": "false",
    "mfa": [
      {
        "strategy": "token-sign",
        "status": "enabled"
      }
```

```
    ],
    "cluster-coverage": "full"
},
```

MFA パブリックキーが登録されたら、管理者ユーザーの MFA パブリックキーの登録を解除する

MFA パブリックキーが登録されているときに、管理者ユーザーの MFA パブリックキーを登録解除するには、次の手順に従います。

1. CloudHSM CLI を使用し、MFA を有効化した管理者として HSM にログインします。
2. user change-mfa token-sign コマンドを使用して、ユーザーの MFA を削除します。

```
aws-cloudhsm > user change-mfa token-sign --username <USERNAME> --role admin --
deregister --change-quorum
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "<USERNAME>",
    "role": "admin"
  }
}
```

トークンファイルのリファレンス

MFA パブリックキーを登録するとき、または MFA を使用してログインしようとしたときに生成されるトークンファイルには、次の内容が含まれます。

- トークン: JSON オブジェクトリテラルの形式で、base64 でエンコードされた署名なし/署名付きトークンのペアの配列。
- 署名なし: Base64 でエンコードされ、SHA256 ハッシュされたトークン。
- 署名済み: RSA 2048 ビットプライベートを使用した、署名されていないトークンの base64 でエンコードされた署名付きトークン (署名)。

```
{
  "version": "2.0",
  "tokens": [
```

```
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/TK0PVaxLN42X
+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37+j/
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQdErI
    }
  ]
}
```


CloudHSM CLI を使用したクォーラム認証の管理 (M of N のアクセスコントロール)

AWS CloudHSM クラスター内の HSMsクォーラム認証をサポートしています。クォーラム認証は M of N アクセスコントロールとも呼ばれます。クォーラム認証を使用すると、HSM の単一のユーザーは HSM でクォーラム管理されたオペレーションを行うことができません。代わりに、HSM ユーザーの最小数 (少なくとも 2 人) が、これらのオペレーションを協力して行う必要があります。クォーラム認証を使用すると、複数の HSM ユーザーからの承認を要求することで、さらに保護レイヤーを追加できます。

クォーラム認証は次のオペレーションを制御できます。

- 管理者による HSM ユーザーの管理 – HSM ユーザーの作成と削除、および、別の HSM ユーザーのパスワードの変更。詳細については、「管理者用クォーラム認証を使用する」を参照してください。

次のトピックでは、 AWS CloudHSMでのクォーラム認証についてさらに詳細な情報を提供します。

トピック

トークン署名戦略によるクォーラム認証の概要

以下のステップは、クォーラム認証のプロセスの概要を示しています。特定のステップとツールについては、管理者用クォーラム認証を使用する を参照してください。

1. 各 HSM ユーザーは署名のための非対称キーを作成します。これは HSM の外部で行い、キーを適切に保護します。
2. 各 HSM ユーザーは HSM にログインし、署名キーの公開部分 (パブリックキー) を HSM に登録します。
3. HSM ユーザーがクォーラム管理されたオペレーションを実行する場合は、HSM にログインし、クォーラムトークンを取得します。
4. HSM ユーザーは、クォーラムトークンを 1 人または複数の他の HSM ユーザーに付与し、承認を求めます。
5. 他の HSM ユーザーは、キーを使用してクォーラムトークンに暗号で署名することにより承認します。これは HSM の外部で行われます。
6. HSM ユーザーが必要な数の承認を得たら、同じユーザーが HSM にログインし、必要な承認 (署名) をすべて含む署名付きクォーラムトークンファイルを提供して、--approval 引数を指定してクォーラム制御オペレーションを実行します。
7. HSM では、それぞれの署名した人の登録されたパブリックキーを使用して署名を確認します。署名が有効な場合、HSM はトークンを承認し、クォーラム制御されたオペレーションが実行されます。

クォーラム認証に関するその他の詳細

AWS CloudHSMでのクォーラム認証の使用に関する次の追加の情報に注意してください。

- HSM ユーザーは自分のクォーラムトークンに署名できます。つまり、リクエストするユーザーはクォーラム認証に必要な承認の 1 つを提供できます。
- クォーラム管理されたオペレーションに対して、最小数のクォーラム承認者を選択します。選択できる最小数は 2 で、選択できる最大数は 8 です。
- HSM はクォーラムトークンを最大 1024 保存できます。HSM にすでに 1024 トークンある場合、新しく作成しようとすると、HSM は期限切れのトークンの 1 つを消去します。デフォルトでは、トークンは作成後 10 分で有効期限が切れます。
- 多要素認証 (MFA) が有効になっている場合、クラスターは、クォーラム認証と MFA に同じキーを使用します。クォーラム認証と 2 要素認証の詳細については、「CloudHSM CLI を使用して MFA を管理する」を参照してください。
- 各 HSM には、サービスごとに一度に 1 つのトークンしか含めることができません。

クォーラム認証をサポートするサービス名とタイプ

管理サービス: クォーラム認証は、ユーザーの作成、ユーザーの削除、ユーザーパスワードの変更、クォーラム値の設定、クォーラム機能と MFA 機能の無効化などの管理者権限を持つサービスに使用されます。

各サービスタイプはさらに適格なサービス名に分類されます。このサービス名には、実行可能なクォーラムがサポートする特定のサービスオペレーションのセットが含まれます。

サービス名	サービスタイプ	サービスオペレーション
ユーザー	管理	• user create • user delete • user change-password • user change-mfa
クォーラム	管理	• クォーラムトークン署名 set-quorum-value

管理者用クォーラム認証を使用する: 初回セットアップ

次のトピックでは、管理者がクォーラム認証を使用できるようにするためのハードウェアセキュリティモジュール (HSM)の設定に必要なステップについて説明します。管理者のクォーラム認証を最

初に設定する場合に、これらのステップを 1 回だけ実行する必要があります。これらのステップが完了したら、管理者用クォーラム認証を使用する を参照してください。

トピック

- 前提条件
- 署名のためのキーの作成と登録
- HSM のクォーラム最小値を設定する

前提条件

この例を理解するには、CloudHSM CLI についての知識が必要です。この例では、 user list コマンドからの次の出力に示すように、 AWS CloudHSM クラスターには 2 HSMs があり、それぞれが同じ管理者を持ちます。ユーザー作成の詳細については、CloudHSM CLI を使用する を参照してください。

```
aws-cloudhsm>user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "admin2",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "admin3",
        "role": "admin",
        "locked": "false",
        "mfa": [],
```

```
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "admin4",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

署名のためのキーの作成と登録

クォーラム認証を使用する場合、各管理者が以下のすべてのステップを実行する必要があります。

トピック

- RSA キーペアの作成
- 登録トークンの作成と署名
- HSM でパブリックキーを登録する

RSA キーペアの作成

様々なキーペアを作成、保護する方法があります。次の例では、OpenSSL 使用方法を説明しています。

Example — OpenSSL でプライベートキーを作成する

次の例は、OpenSSL を使用してパスフレーズで保護された 2048 ビットの RSA キーを作成する方法を示しています。この例を使用するには、*<admin.key>* を、キーの保存先のファイル名に置き換えてください。

```
$ openssl genrsa -out <admin.key> -aes256 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
Enter pass phrase for admin.key:
Verifying - Enter pass phrase for admin.key:
```

次に、作成したプライベートキーを使用してパブリックキーを生成します。

Example — OpenSSL でパブリックキーを作成する

以下の例は、OpenSSL を使用して先ほど作成したプライベートキーからパブリックキーを作成する方法を示しています。

```
$ openssl rsa -in admin.key -outform PEM -pubout -out admin1.pub
Enter pass phrase for admin.key:
writing RSA key
```

登録トークンの作成と署名

トークンを作成し、前のステップで生成したプライベートキーを使用して署名します。

Example – 登録トークンの作成と署名

1. CloudHSM CLI を起動するには、次のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. quorum token-sign generate コマンドを実行して登録トークンを作成します。

```
aws-cloudhsm > quorum token-sign generate --service registration --token /path/
tokenfile
{
  "error_code": 0,
  "data": {
    "path": "/path/tokenfile"
  }
}
```

3. quorum token-sign generate コマンドは、指定されたファイルパスに登録トークンを生成します。トークンファイルを調査します。

```
$ cat /path/tokenfile{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": ""
    }
  ]
}
```

トークンファイルは、次のもので構成されます。

- approval_data: base64 でエンコードされランダム化されたデータトークン。raw データが最大 245 バイトを超えないもの。
- unsigned: base64 でエンコードされ、SHA256 ハッシュされた approval_data のトークン。
- signed: OpenSSL で以前に生成された RSA 2048 ビットのプライベートキーを使用した、署名されていないトークンの base64 でエンコードされた署名付きトークン (署名)。

プライベートキーを使用して署名なしトークンに署名し、プライベートキーへのアクセス権があることを示します。管理者をクォーラムユーザーとして AWS CloudHSM クラスターに登録するには、登録トークンファイルに署名とパブリックキーが完全に入力されている必要があります。

Example – 署名なし登録トークンへ署名する

1. base64 でエンコードされた署名なしトークンをデコードし、バイナリファイルに入れます。

```
$ echo -n '6BMUj6mUjjko6ZLCEdzGlWpR5sILhFJfqhW1ej3Oq1g=' | base64 -d > admin.bin
```

2. OpenSSL とプライベートキーを使用して現在の署名なしバイナリ登録トークンに署名し、バイナリ署名ファイルを作成します。

```
$ openssl pkeyutl -sign \
-inkey admin.key \
-pkeyopt digest:sha256 \
-keyform PEM \
-in admin.bin \
-out admin.sig.bin
```

3. バイナリ署名を base64 にエンコードします。

```
$ base64 -w0 admin.sig.bin > admin.sig.b64
```

4. base64 でエンコードされた署名をコピーしてトークンファイルに貼り付けます。

```
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": <signed token in base64 encoding>
    }
  ]
}
```

HSM でパブリックキーを登録する

キーを作成したら、管理者はパブリックキーを AWS CloudHSM クラスターに登録する必要があります。

HSM にパブリックキーの登録するには

1. CloudHSM CLI を起動するには、次のコマンドを使用します。

Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
```

2. CloudHSM CLI を使用して、管理者としてログインします。

```
aws-cloudhsm > login --username admin --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

3. ユーザー変更クォーラムトークン署名登録 コマンドを使用してパブリックキーを登録します。詳細については、次の例を参照するか、または help user change-quorum token-sign register コマンドを使用してください。

Example – AWS CloudHSM クラスターにパブリックキーを登録する

以下の例では、CloudHSM CLI で user change-quorum token-sign register コマンドを使用して、管理者のパブリックキーを HSM に登録する方法を示しています。このコマンドを使用するには、管理者が HSM にログインしている必要があります。以下の値を自分の値に置き換えてください。

```
aws-cloudhsm > user change-quorum token-sign register --public-key </path/admin.pub> --
signed-token </path/tokenfile>
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

> **Note**
>
> /path/admin.pub: パブリックキー PEM ファイルへのファイルパス
> 必須: はい
> /path/tokenfile: ユーザーのプライベートキーによって署名されたトークンを含むファイルパス
> 必須: はい

すべての管理者がパブリックキーを登録すると、user list コマンドの出力のクォーラムフィールドには次のように表示され、有効になっているクォーラム戦略が使用中であることが示されます。

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin2",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
```

```
      },
      {
        "username": "admin3",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin4",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

HSM のクォーラム最小値を設定する

クォーラム認証を使用するには、管理者が HSM にログインしてクォーラム最小値を設定する必要があります。これは、HSM ユーザー管理オペレーションを実行するために必要な管理者承認の最小数

です。HSM 上の任意の管理者は、署名用のキーを登録していない管理者を含むクォーラム最小値を設定できます。クォーラム最小値はいつでも変更できます。詳細情報は、「最小値を変更」を参照してください。

HSM のクォーラム最小値の設定

1. CloudHSM CLI を起動するには、次のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. CloudHSM CLI を使用して、管理者として口グインします。

   ```
   aws-cloudhsm > login --username admin --role admin
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin",
       "role": "admin"
     }
   }
   ```

3. クォーラム最小値を設定する場合、クォーラムトークン署名 set-quorum-value コマンドを使用します。詳細については、次の例を参照するか、または help quorum token-sign set-quorum-value コマンドを使用してください。

Example HSM のクォーラム最小値を設定する

この例では、クォーラム最小値 2 を使用します。最大は HSM 上の管理者の合計数で、2 から 8 までの任意の値を選択できます。この例では、HSM には 4 人の管理者がいるため、設定可能な最大値は 4 です。

次のコマンド例を使用するには、最後の数値 (*<2>*) を所望のフォーラム最小値に置き換えてください。

```
aws-cloudhsm > quorum token-sign set-quorum-value --service user --value <2>
{
  "error_code": 0,
  "data": "Set quorum value successful"
}
```

この例では、サービスは、クォーラム最小値を設定しようとしている HSM サービスを示しています。クォーラムトークン署名 list-quorum-values コマンドは、サービスに含まれる HSM サービスのタイプ、名前、および説明を示しています。

管理サービス: クォーラム認証は、ユーザーの作成、ユーザーの削除、ユーザーパスワードの変更、クォーラム値の設定、クォーラム機能と MFA 機能の無効化などの管理者権限を持つサービスに使用されます。

各サービスタイプはさらに適格なサービス名に分類されます。このサービス名には、実行可能なクォーラムがサポートする特定のサービスオペレーションのセットが含まれます。

サービス名	サービスタイプ	サービスオペレーション
ユーザー	管理	• user create • user delete • user change-password • user change-mfa
クォーラム	管理	• クォーラムトークン署名 set-quorum-value

サービスのクォーラム最小値を取得するには、quorum token-sign list-quorum-values コマンドを使用します。

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 1
  }
}
```

前述の quorum token-sign list-quorum-values コマンドの出力は、ユーザー管理オペレーションを担当する HSM ユーザーサービスのクォーラム最小値が 2 になったことを示しています。これらのステップが完了したら、クォーラムの使用 (M of N) を参照してください。

管理者用クォーラム認証を使用する

HSM の管理者は、 AWS CloudHSM クラスター内の次のオペレーションのクォーラム認証を設定できます。

- user create
- user delete
- user change-password
- user change-mfa

AWS CloudHSM クラスターがクォーラム認証用に設定されると、管理者は HSM ユーザー管理オペレーションを自分で実行できなくなります。次の例は、管理者が HSM で新しいユーザーを作成しようとしたときの出力を示しています。コマンドは失敗し、クォーラム認証が必要であることを示すエラーが表示されます。

```
aws-cloudhsm > user create --username user1 --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 1,
  "data": "Quorum approval is required for this operation"
}
```

HSM ユーザー管理オペレーションを実行するには、管理者は以下のタスクを完了する必要があります。

トピック

- クォーラムトークンの取得
- 承認する管理者から署名を取得する
- AWS CloudHSM クラスターでトークンを承認し、ユーザー管理オペレーションを実行する

クォーラムトークンの取得

まず、管理者は CloudHSM CLI を使用してクォーラムトークンをリクエストする必要があります。

クォーラムトークンを取得するには

1. CloudHSM CLI を起動するには、次のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてクラスターにログインします。

   ```
   aws-cloudhsm>login --username admin --role admin
   ```

3. quorum token-sign generate コマンドを使用してクォーラムトークンを生成します。詳細については、次の例を参照するか、または help quorum token-sign generate コマンドを使用してください。

Example – クォーラムトークンを生成する

この例では、ユーザー名を admin とする管理者のクォーラムトークンを取得し、そのトークンを admin.token というファイルに保存します。この例のコマンドを使用するには、以下の値を独自のものに置き換えてください。

- *<admin>* – トークンを取得する管理者の名前。HSM にログインし、このコマンドを実行している管理者と同じであることが必要です。
- *<admin.token>* – クォーラムトークンを保存するファイルの名前。

次のコマンドで、user は生成するトークンを使用できるサービス名を識別します。この場合、トークンは HSM ユーザー管理オペレーション (user サービス) 用です。

```
aws-cloudhsm > login --username <ADMIN> --role <ADMIN> --password <PASSWORD>
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
```

```
  }
}

aws-cloudhsm > quorum token-sign generate --service user --token </path/admin.token>
{
  "error_code": 0,
  "data": {
    "path": "/home/tfile"
  }
}
```

quorum token-sign generate コマンドは、指定されたファイルパスでユーザーサービスのクォーラムトークンを生成します。トークンファイルは以下のように検査できます。

```
$cat </path/admin.token>
{
  "version": "2.0",
  "approval_data": "AAEAAwAAABgAAAAAAAAAAJ9eFkfcP3mNzJAlfK
+OWbNhZG1pbgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABj5vbeAAAAAAAAAAAAAQADAAAAFQAAAAAAAAAW/
v5Euk83amq1fij0zyvD2FkbWluAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGPm9t4AAAAAAAAAAAABAAMAAAAUA
+b23gAAAAAAAAAA",
  "token": "0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=",
  "signatures": []
}
```

トークンファイルは、次のもので構成されます。

- approval_data: HSM によって生成された base64 でエンコードされた raw データトークン。
- token: base64 でエンコードされ、SHA-256 ハッシュされた approval_data のトークン
- signatures: base64 でエンコードされた署名なしトークンの署名済みトークン (署名) の配列。承認者の各署名は JSON オブジェクトリテラルの形式になっています。

  ```
  {
        "username": "<APPROVER_USERNAME>",
        "signature": "<APPROVER_RSA2048_BIT_SIGNATURE>"
  }
  ```

  各署名は、パブリックキーが HSM に登録されている承認者が、対応する RSA 2048 ビットプライベートキーを使用した結果から作成されます。

生成されたユーザーサービスのクォーラムトークンが CloudHSM クラスターに存在していることは、quorum token-sign list コマンドを実行することによって確認できます。

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": [
      {
        "username": "admin",
        "service": "user",
        "approvals-required": {
          "value": 2
        },
        "number-of-approvals": {
          "value": 0
        },
        "token-timeout-seconds": {
          "value": 597
        },
        "cluster-coverage": "full"
      }
    ]
  }
}
```

token-timeout-seconds 時間は、生成されたトークンの有効期限が切れる前に承認されるまでのタイムアウト時間を秒単位で示します。

承認する管理者から署名を取得する

クォーラムトークンを持つ管理者は、そのトークンを他の管理者に承認してもらう必要があります。他の管理者は、承認を与えるために、署名キーを使用してトークンを暗号で署名します。この署名は HSM 外で行われます。

トークンの署名にはさまざまな方法が使用されます。次の例では、OpenSSL を使用しています。別の署名ツールを使用する場合は、そのツールで必ず管理者のプライベートキー (署名キー) を使用してトークンの SHA-256 ダイジェストに署名します。

Example – 承認する管理者から署名を取得する

この例では、トークン (admin) を持つ管理者に少なくとも 2 つの承認が必要です。以下のコマンド例では、2 人の管理者が OpenSSL を使用してトークンに暗号で署名する方法を示します。

1. base64 でエンコードされた署名なしトークンをデコードし、バイナリファイルに入れます。

```
$echo -n '0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=' | base64 -d > admin.bin
```

2. OpenSSL と、承認者 (admin3) のそれぞれのプライベートキーを使用して、ユーザーサービスの現在のバイナリクォーラム署名なしトークンに署名し、バイナリ署名ファイルを作成します。

```
$openssl pkeyutl -sign \
-inkey admin3.key \
-pkeyopt digest:sha256 \
-keyform PEM \
-in admin.bin \
-out admin.sig.bin
```

3. バイナリ署名を base64 にエンコードします。

```
$base64 -w0 admin.sig.bin > admin.sig.b64
```

4. 最後に、承認者署名用に以前に指定した JSON オブジェクトリテラル形式に従って、base64 でエンコードされた署名をコピーしてトークンファイルに貼り付けます。

```
{
  "version": "2.0",
  "approval_data": "AAEAAwAAABgAAAAAAAAAAJ9eFkfcP3mNzJAlfK
+OWbNhZG1pbgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABj5vbeAAAAAAAAAAAAAQADAAAAFQAAAAAAAAAW
v5Euk83amq1fij0zyvD2FkbWluAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGPm9t4AAAAAAAAAAAABAAMAA
+b23gAAAAAAAAAA",
  "token": "0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=",
  "signatures": [
    {
      "username": "admin2",
      "signature": "O6qx7/mUaVkYYVr1PW7l8JJko+Kh3e8zBIqdk3tAiNy+1rW
+OsDtvYujhEU4aOFVLcrUFmyB/CX9OQmgJLgx/pyK+ZPEH+GoJGqk9YZ7X1nOXwZRP9g7hKV
+7XCtg9TuDFtHYWDpBfz2jWiu2fXfX4/
jTs4f2xIfFPIDKcSP8fhxjQ63xEcCf1jzGha6rDQMu4xUWWdtDgfT7um7EJ9dXNoHqLB7cTzphaubNaEFbFPXQ1siGm
ssktwyruGFLpXs1n0tJOEglGhx2qbYTs+omKWZdORl5WIWEXW3IXw/
Dg5vVObrNpvG0eZKO8nSMc27+cyPySc+ZbNw=="
    },
    {
      "username": "admin3",
      "signature": "O6qx7/mUaVkYYVr1PW7l8JJko+Kh3e8zBIqdk3tAiNy+1rW
+OsDtvYujhEU4aOFVLcrUFmyB/CX9OQmgJLgx/pyK+ZPEH+GoJGqk9YZ7X1nOXwZRP9g7hKV
```

```
+7XCtg9TuDFtHYWDpBfz2jWiu2fXfX4/
jTs4f2xIfFPIDKcSP8fhxjQ63xEcCf1jzGha6rDQMu4xUWWdtDgfT7um7EJ9dXNoHqLB7cTzphaubNaEFbFPXQ1siGm
ssktwyruGFLpXs1n0tJOEglGhx2qbYTs+omKWZdORl5WIWEXW3IXw/
Dg5vVObrNpvG0eZKO8nSMc27+cyPySc+ZbNw=="
    }
  ]
}
```

AWS CloudHSM クラスターでトークンを承認し、ユーザー管理オペレーションを実行する

前のセクションで説明したように、管理者は必要な承認/署名を取得すると、管理者は以下のユーザー管理オペレーションのいずれかとともにそのトークンを AWS CloudHSM クラスターに提供することができます。

- 作成
- 削除
- change-password
- user change-mfa

これらのコマンドの詳しい使用方法については、CloudHSM CLI を使用する を参照してください。

トランザクション中、トークンは AWS CloudHSM クラスター内で承認され、リクエストされたユーザー管理オペレーションを実行します。ユーザー管理オペレーションが成功するかどうかは、承認された有効なクォーラムトークンと有効なユーザー管理オペレーションの両方に左右されます。

管理者は、トークンを 1 つのオペレーションにのみ使用できます。そのオペレーションが成功すると、トークンは無効になります。別の HSM ユーザー管理オペレーションを行うには、管理者は上記の手順を繰り返す必要があります。つまり、管理者は新しいクォーラムトークンを取得し、承認者から新しい署名を取得した後で、要求されたユーザー管理オペレーションによって HSM で新しいトークンを承認して使用する必要があります。

> **Note**
>
> クォーラムトークンは、現在のログインセッションが開いている間だけ有効です。CloudHSM CLI からログアウトするか、ネットワークが切断された場合、トークンは無効になります。同様に、承認されたトークンは CloudHSM CLI 内でのみ使用できます。別のアプリケーションでの認証には使用できません。

Example 管理者として新規ユーザーを作成する

次の例では、ログイン中の管理者が HSM で新しいユーザーを作成しています。

```
aws-cloudhsm > user create --username user1 --role crypto-user --approval /path/
admin.token
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "user1",
    "role": "crypto-user"
  }
}
```

その後、管理者は新しいユーザーの作成を確認する user list コマンドを入力します。

```
aws-cloudhsm > user list{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin2",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
```

```
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin3",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin4",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "user1",
        "role": "crypto-user",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
```

```
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

管理者が別の HSM ユーザー管理オペレーションを実行しようとするとクォーラム認証エラーが発生して失敗します。

```
aws-cloudhsm > user delete --username user1 --role crypto-user
{
  "error_code": 1,
  "data": "Quorum approval is required for this operation"
}
```

以下に示すように、quorum token-sign list コマンドは管理者に承認されたトークンがないことを示しています。別の HSM ユーザー管理オペレーションを実行するには、管理者は新しいクォーラムトークンを生成し、承認者から新しい署名を取得し、--approval 引数を使用して目的のユーザー管理オペレーションを実行して、ユーザー管理オペレーションの実行中に承認され使用されるクォーラムトークンを供給する必要があります。

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": []
  }
}
```

管理者のクォーラム最小値を変更する

管理者 がクォーラム認証を使用できるようにクォーラム最小値を設定した後で、クォーラム最小値を変更する場合があります。承認者の数が現在のクォーラム最小値以上の場合のみ、HSM はクォーラム最小値の変更を許可します。例えば、クォーラム最小値が 2 の場合、クォーラム最小値を変更するには最低 2 つの管理者の承認が必要です。

> **Note**
>
> ユーザーサービスのクォーラム値は、常にクォーラムサービスのクォーラム値より小さくなければなりません。クォーラムサービスやユーザーサービスなどのサービス名については、「クォーラム認証をサポートするサービス名とタイプ」を参照してください。

クォーラム最小値の変更のためにクォーラムの承認を取得する場合、quorum token-sign set-quorum-value コマンドを使用する quorum service のためのクォーラムトークンが必要です。quorum token-sign set-quorum-value コマンドを使用して quorum service のクォーラムトークンを生成するには、クォーラムサービスが 1 より大きい必要があります。つまり、ユーザーサービスのクォーラム最小値を変更するには、クォーラムサービスのクォーラム最小値の変更が必要になる場合があります。

管理者のクォーラム最小値を変更するには

1. CloudHSM CLI インタラクティブモードを起動するには、以下のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
   ```

2. login コマンドを使用して、管理者としてクラスターにログインします。

   ```
   aws-cloudhsm>login --username <admin> --role admin
   ```

3. quorum token-sign list-quorum-values コマンドを使用して、すべてのサービス名のクォーラム最小値を取得します。詳細については、以下の例を参照してください。
4. クォーラムサービスのクォーラム最小値がユーザーサービスの値より小さい場合、quorum token-sign set-quorum-value コマンドを使用してクォーラムサービスの値を変更します。クォーラムサービスの値を、1 (ユーザーサービスの値と同じ) またはそれより大きい値に変更します。詳細については、次の例を参照してください。
5. クォーラムサービスをトークンを使用できるサービスとして指定するように注意し、クォーラムトークンを生成します。
6. 他の管理者からの承認 (署名) の取得。

7. AWS CloudHSM クラスターでトークンを承認し、ユーザー管理オペレーションを実行します。
8. quorum token-sign set-quorum-value コマンドを使用して、ユーザーサービスのクォーラム最小値を変更します。

Example - クォーラム最小値を取得してクォーラムサービスの値を変更する

次のコマンド例では、ユーザーサービスのクォーラム最小値が現在 2 であることを示しています。

```
aws-cloudhsm > quorum token-sign list-quorum-values{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 1
  }
}
```

クォーラムサービスのクォーラム最小値を変更するには、ユーザーサービスの値と同じがそれより大きい値を設定して quorum token-sign set-quorum-value コマンドを使用します。次の例では、クォーラムサービスのクォーラム最小値をユーザーサービスに設定されているのと同じ値である 2 に設定します。

```
aws-cloudhsm > quorum token-sign set-quorum-value --service quorum --value 2{
  "error_code": 0,
  "data": "Set quorum value successful"
}
```

次のコマンド例は、ユーザーサービスとクォーラムサービスでクォーラム最小値が現在 2 であることを示しています。

```
aws-cloudhsm > quorum token-sign list-quorum-values{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 2
  }
}
```

CloudHSM 管理ユーティリティ (CMU) を使用する HSM ユーザーを管理する

では AWS CloudHSM、CloudHSM CLI または CloudHSM 管理ユーティリティ (CMU) コマンドラインツールを使用して、HSM でユーザーを作成および管理する必要があります。CloudHSM CLI は最新の SDK で使用するように設計されていますが、CMU は以前の SDK で使用するように設計されています。

トピック

HSM ユーザーを理解する

HSM 上で実行するほとんどのオペレーションでは、HSM ユーザーの認証情報が必要です。HSM は各 HSM ユーザーを認証し、各 HSM ユーザーに設定されている タイプ により、ユーザーとして HSM で実行できるオペレーションが決定されます。

> **Note**
>
> HSM ユーザーは IAM ユーザーとは異なります。正しい認証情報を持つ IAM ユーザーは、AWS API を介してリソースを操作することで HSM を作成できます。HSM を作成したら、HSM ユーザー認証情報を使用して HSM でのオペレーションを認証する必要があります。

ユーザータイプ

- PreCrypto Officer (PRECO)
- Crypto Officer (CO)
- Crypto user (CU)

- Appliance user (AU)

PreCrypto Officer (PRECO)

クラウド管理ユーティリティ (CMU) とキー管理ユーティリティ (KMU) のどちらでも、PRECO は AWS CloudHSM クラスター内の最初の HSM にのみ存在する一時的なユーザーです。新しいクラスターの最初の HSM には、このクラスターがアクティブ化されたことがないことを示す PRECO ユーザーが含まれています。クラスターをアクティブ化する には、cloudhsm-cli を実行して cluster activate コマンドを実行します。HSM にログインし、PRECO のパスワードを変更します。パスワードを変更すると、このユーザーは Crypto Officer (CO) になります。

Crypto Officer (CO)

クラウド管理ユーティリティ (CMU) とキー管理ユーティリティ (KMU) のどちらでも、Crypto Officer (CO) がユーザー管理オペレーションを実行できます。たとえば、ユーザーの作成および削除と、ユーザーパスワードの変更を行うことなどができます。CO ユーザーの詳細情報は、HSM ユーザーの許可テーブル を参照してください。新しいクラスターを有効にすると、ユーザーは、Precrypto Officer (PRECO) から Crypto Officer (CO) に変わります。-->

Crypto user (CU)

暗号化ユーザー (CU) は、以下のキー管理および暗号化のオペレーションを行うことができます。

- キー管理 - 暗号化キーの作成、削除、共有、インポート、エクスポートを行います。
- 暗号化オペレーション - 暗号化キーを使用して、暗号化、復号、署名、検証などを行います。

詳細については、「HSM ユーザーの許可テーブル」を参照してください。

Appliance user (AU)

アプライアンスユーザー (AU) は、クラスターの HSMs。 は AU AWS CloudHSM を使用して AWS CloudHSM 、クラスター内の HSMsを同期します。AU は、 によって提供されるすべての HSMs に存在し AWS CloudHSM、アクセス許可が制限されています。詳細については、「HSM ユーザーの許可テーブル」を参照してください。

AWS はHSMs でオペレーションを実行できません。ユーザーまたはキーを表示または変更 AWS することはできません。また、これらのキーを使用して暗号化オペレーションを実行することはできません。

HSM ユーザーの許可テーブル

以下の表は、オペレーションを実行できる HSM ユーザーまたはセッションのタイプ別にソートされた HSM オペレーションリストです。

	Crypto Officer (CO)	暗号化ユーザー (CU)	Appliance User (AU)	未認証セッション
基本的なクラスター情報を取得する[1]	 はい	 はい	 はい	 はい
自分のパスワードを変更する	 はい	 はい	 はい	該当しない
ユーザーのパスワードを変更する	 はい	 いいえ	 いいえ	 いいえ
ユーザーを追加、削除する	 はい	 いいえ	 いいえ	 いいえ
同期のステータスを取得する[2]	 はい	 はい	 はい	 いいえ
マスクされたオブジェクトを抽出、挿入する[3]	 はい	 はい	 はい	 いいえ

	Crypto Officer (CO)	暗号化ユーザー (CU)	Appliance User (AU)	未認証セッション
キー管理機能[4]	 いいえ	 はい	 いいえ	 いいえ
暗号化、復号する	 いいえ	 はい	 いいえ	 いいえ
署名、検証する	 いいえ	 はい	 いいえ	 いいえ
ダイジェストと HMAC の生成	 いいえ	 はい	 いいえ	 いいえ

- [1] 基本情報には、クラスター内の HSM 数、各 HSM の IP アドレス、モデル、シリアル番号、デバイス ID、ファームウェア ID などが含まれます。
- [2] ユーザーは、HSM のキーに対応するダイジェスト (ハッシュ) のセットを取得できます。アプリケーションは、これらのダイジェストのセットを比較して、クラスター内の HSM の同期状態を把握します。
- [3] マスクされたオブジェクトは、HSM を離れる前に暗号化されるキーです。これらのオブジェクトを HSM の外部で復号することはできません。これらは、抽出された HSM と同じクラスターにある HSM に挿入された後にのみ復号されます。アプリケーションはマスクされたオブジェクトを抽出して挿入し、クラスター内の HSM を同期します。
- [4] キー管理機能には、キーの属性の作成、削除、ラップ、ラップ解除、変更が含まれます。

CloudHSM 管理ユーティリティ (CMU) を使用したユーザーの管理

このトピックでは、クライアント SDK に付属するコマンドラインツールである CloudHSM 管理ユーティリティ (CMU) を使用して、ハードウェアセキュリティモジュール (HSM) ユーザーを管理する step-by-step 方法について説明します。CMU または HSM ユーザーの詳細情報は、CloudHSM 管理ユーティリティ と HSM ユーザーを理解する を参照してください。

セクション

- CMU を使用した HSM ユーザー管理について
- CloudHSM 管理ユーティリティのダウンロード
- CMU で HSM ユーザーを管理する方法

CMU を使用した HSM ユーザー管理について

HSM ユーザーを管理するには、暗号化オフィサー (CO) のユーザー名とパスワードを使用して HSM にログインする必要があります。CO のみユーザーを管理できます。HSM には、admin という名前のデフォルト CO が含まれています。admin クラスターのアクティブ化 の際に必要なパスワードを設定しました。

CMU を使用する場合、設定ツールでローカル設定を更新する必要があります。CMU はクラスターへの独自の接続を作成しますが、この接続はクラスターを 認識しません。クラスター情報を追跡するため、CMU はローカル設定ファイルを保持します。これは、毎回 CMU を使用する際、まず、`--cmu` パラメータを指定して 設定 コマンドラインツールを実行し、設定ファイルを更新する必要があります。Client SDK 3.2.1 以前のバージョンを使用している場合、`--cmu` とは異なるパラメータを使用する必要があります。詳細については、「the section called “Client SDK 3.2.1 以前のバージョンでの CMU の使用”」を参照してください。

`--cmu` パラメータには、クラスター内の HSM の IP アドレスを追加する必要があります。複数の HSM をお持ちの方は、任意の IP アドレスを使用できます。これで確実に CMU がクラスター全体に加えた変更を伝播できます。CMU はローカルファイルを使用してクラスター情報を追跡することに注意してください。特定のホストから CMU を最後に使用後にクラスターが変更されている場合、該当するホストに保存されているローカル設定ファイルにそれらの変更を追加する必要があります。CMU の使用中に HSM を追加または削除しないでください。

HSM の IP アドレスを取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。

3. クラスターの詳細ページを開くには、クラスターテーブルでクラスター ID を選択します。
4. IP アドレスを取得するには、[HSM] タブで、[ENI IP アドレス] にリストされている IP アドレスのいずれかを選択します。

HSM の IP アドレスを取得するには (AWS CLI)

- describe-clusters から AWS CLIコマンドを実行して、HSM の IP アドレスを取得します。コマンドの出力では、HSM の IP アドレスは `EniIp` の値です。

```
$  aws cloudhsmv2 describe-clusters


{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
...
```

Client SDK 3.2.1 以前のバージョンでの CMU の使用

Client SDK 3.3.0 では、 `--cmu`パラメータのサポート AWS CloudHSM が追加されました。これにより、CMU の設定ファイルを更新するプロセスが簡素化されます。Client SDK 3.2.1 以前のバージョンの CMU を使用している場合は、`-a` と `-m` パラメータを引き続き使用して設定ファイルを更新する必要があります。パラメータの詳細情報は、設定ツール を参照してください。

CloudHSM 管理ユーティリティのダウンロード

Client SDK 5 および Client SDK 3 の使用の有無を問わず、 HSM ユーザー管理タスクで最新バージョンの CMU を使用できます。

CMU のダウンロードおよびインストール

- CMU をダウンロードおよびインストールします。

  Amazon Linux

  ```
  $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-mgmt-util-latest.el6.x86_64.rpm
  ```

  ```
  $ sudo yum install ./cloudhsm-mgmt-util-latest.el6.x86_64.rpm
  ```

  Amazon Linux 2

  ```
  $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
  ```

  ```
  $ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
  ```

  CentOS 7.8+

  ```
  $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
  ```

  ```
  $ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
  ```

  CentOS 8.3+

  ```
  $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-mgmt-util-latest.el8.x86_64.rpm
  ```

  ```
  $ sudo yum install ./cloudhsm-mgmt-util-latest.el8.x86_64.rpm
  ```

  RHEL 7 (7.8+)

  ```
  $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
  ```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-mgmt-util_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

Windows Server 2012

1. CloudHSM 管理ユーティリティ をダウンロードします。
2. Windows 管理権限を持つ CMU インストーラ (AWSCloudHSMManagementUtil-latest.msi) を実行します。

Windows Server 2012 R2

1. CloudHSM 管理ユーティリティ をダウンロードします。
2. Windows 管理権限を持つ CMU インストーラ (AWSCloudHSMManagementUtil-latest.msi) を実行します。

Windows Server 2016

1. CloudHSM 管理ユーティリティ をダウンロードします。
2. Windows 管理権限を持つ CMU インストーラ (AWSCloudHSMManagementUtil-latest.msi) を実行します。

CMU で HSM ユーザーを管理する方法

このセクションでは、CMU を使用した HSM ユーザーを管理用の基本的なコマンドについて説明します。

HSM ユーザー の作成

HSM に createUser を使用して新しいユーザーを作成します。ユーザーを作成する場合、CO としてログインする必要があります。

新しい CO ユーザーの作成

1. 設定ツールで CMU 設定を更新します。

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
   ```

2. CMU を開始します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. CO ユーザーとして HSM にログインします。

```
aws-cloudhsm>loginHSM CO admin co12345
```

接続 CMU リストの数が、クラスター内の HSM 数と一致していることを確認します。一致しない場合、ログアウトして最初からやり直してください。

4. createUser を使用して、 パスワードを **example_officer** に設定して **password1** という CO ユーザーを作成します。

```
aws-cloudhsm>createUser CO example_officer password1
```

CMU は、ユーザーの作成オペレーションについてプロンプトを表示します。

```
************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?
```

5. タイプ **y**。

新しい CU ユーザーの作成

1. 設定ツールで CMU 設定を更新します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
```

2. CMU を開始します。

Linux

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

Windows

```
C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. CO ユーザーとして HSM にログインします。

```
aws-cloudhsm>loginHSM CO admin co12345
```

接続 CMU リストの数が、クラスター内の HSM 数と一致していることを確認します。一致しない場合、ログアウトして最初からやり直してください。

4. createUser を使用して、**example_user** という CU ユーザーをパスワードは **password1** で作成します。

```
aws-cloudhsm>createUser CU example_user password1
```

CMU は、ユーザーの作成オペレーションについてプロンプトを表示します。

```
**********************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?
```

5. タイプ **y**。

createUser の詳細情報は、createUser を参照してください。

クラスター上のすべての HSM ユーザーを一覧表示

listUsers コマンドを使用して、クラスター上のすべてのユーザーを一覧表示します。listUsers を実行する際、ログインは不要であり、すべてのユーザータイプでユーザーをリストアップできます。

クラスター上のすべてのユーザーを一覧表示

1. 設定ツールで CMU 設定を更新します。

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
   ```

2. CMU を開始します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. listUsers を使用して、クラスター上のすべてのユーザーを一覧表示します。

   ```
   aws-cloudhsm>listUsers
   ```

   CMU は、クラスター上のすべてのユーザーを一覧表示します。

   ```
   Users on server 0(10.0.2.9):
   Number of users found:4

      User Id             User Type       User Name
    MofnPubKey    LoginFailureCnt         2FA
   ```

```
        1               AU              app_user                        NO
              0               NO
        2               CO              example_officer                 NO
              0               NO
        3               CU              example_user                    NO
              0               NO
Users on server 1(10.0.3.11):
Number of users found:4

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
        1               AU              app_user                        NO
              0               NO
        2               CO              example_officer                 NO
              0               NO
        3               CU              example_user                    NO
              0               NO
Users on server 2(10.0.1.12):
Number of users found:4

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
        1               AU              app_user                        NO
              0               NO
        2               CO              example_officer                 NO
              0               NO
        3               CU              example_user                    NO
              0               NO
```

listUsers の詳細情報は、listUsers を参照してください。

HSM ユーザーパスワードの変更

changePswd を使用してパスワードを変更します。

ユーザータイプとパスワードは大文字と小文字が区別されますが、ユーザー名では区別されません。

暗号化ユーザー (CU) とアプライアンスユーザー (AU) は、自分のパスワードのみ変更できます。別のユーザーのパスワードを変更する場合、CO としてログインする必要があります。ただし、現在ログインしているユーザーのパスワードを変更することはできません。

自分のパスワードの変更

1. 設定ツールで CMU 設定を更新します。

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
   ```

2. CMU を開始します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. HSM へログインします。

   ```
   aws-cloudhsm>loginHSM CO admin co12345
   ```

   接続 CMU リストの数が、クラスター内の HSM 数と一致していることを確認します。一致しない場合、ログアウトして最初からやり直してください。

4. changePswd を使用してユーザー自身のパスワードを変更します。

   ```
   aws-cloudhsm>changePswd CO example_officer <new password>
   ```

   CMU は、パスワードの変更オペレーションについてプロンプトを表示します。

   ```
   ************************CAUTION*********************************
   This is a CRITICAL operation, should be done on all nodes in the
   cluster. AWS does NOT synchronize these changes automatically with the
   nodes on which this operation is not executed or failed, please
   ```

```
ensure this operation is executed on all nodes in the cluster.
*************************************************************

Do you want to continue(y/n)?
```

5. タイプ **y**。

   CMU は、パスワードの変更オペレーションについてプロンプトを表示します。

```
Changing password for example_officer(CO) on 3 nodes
```

別のユーザーのパスワードの変更

1. 設定ツールで CMU 設定を更新します。

   Linux

```
$ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
```

   Windows

```
C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
```

2. CMU を開始します。

   Linux

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

   Windows

```
C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. CO ユーザーとして HSM にログインします。

```
aws-cloudhsm>loginHSM CO admin co12345
```

接続 CMU リストの数が、クラスター内の HSM 数と一致していることを確認します。一致しない場合、ログアウトして最初からやり直してください。

4. changePswd を使用して別のユーザーのパスワードを変更します。

```
aws-cloudhsm>changePswd CU example_user <new password>
```

CMU は、パスワードの変更オペレーションについてプロンプトを表示します。

```
**********************CAUTION*****************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**********************************************************

Do you want to continue(y/n)?
```

5. タイプ **y**。

CMU は、パスワードの変更オペレーションについてプロンプトを表示します。

```
Changing password for example_user(CU) on 3 nodes
```

changePswd の詳細情報は、[パスワードの変更] を参照してください。

HSM ユーザーの削除

deleteUser を使用してユーザーを削除します。別のユーザーを削除する場合、CO としてログインする必要があります。

> **Tip**
>
> キーを所有している暗号化ユーザー (CU) を削除することはできません。

ユーザーの削除

1. 設定ツールで CMU 設定を更新します。

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
   ```

2. CMU を開始します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. CO ユーザーとして HSM にログインします。

   ```
   aws-cloudhsm>loginHSM CO admin co12345
   ```

   接続 CMU リストの数が、クラスター内の HSM 数と一致していることを確認します。一致しない場合、ログアウトして最初からやり直してください。

4. deleteUser を使用してユーザーを削除します。

   ```
   aws-cloudhsm>deleteUser CO example_officer
   ```

   CMU がユーザーを削除します。

   ```
   Deleting user example_officer(CO) on 3 nodes
   deleteUser success on server 0(10.0.2.9)
   deleteUser success on server 1(10.0.3.11)
   ```

```
deleteUser success on server 2(10.0.1.12)
```

deleteUser の詳細情報は、deleteUser を参照してください。

CloudHSM 管理ユーティリティ (CMU) を使用した暗号通貨担当者の 2 要素認証 (2FA) 管理

セキュリティを向上する場合、2 要素認証 (2FA) を設定してクラスターを保護できます。Crypto Officer (CO) に対してのみ 2FA を有効化できます。

> **Note**
>
> 暗号ユーザー (CU) またはアプリケーションに対して 2FA を有効化できません。2 要素認証 (2FA) は CO ユーザーのみ対象です。

トピック

- HSM ユーザー用 2FA について
- HSM ユーザー向け 2FA を使用した作業

HSM ユーザー用 2FA について

2FA 有効のハードウェアサービスモジュール (HSM) アカウントでクラスターにログインする場合、cloudhsm_mgmt_util (CMU) にパスワード (最初の要素、ユーザーが知っているもの) を指定します。CMU はトークンを提供し、トークンに署名を要求するプロンプトを表示します。2 番目の要素 (自分の持っているもの) を提供する場合、すでに作成して HSM ユーザーに紐づけしたキーペアからプライベートキーを使用してトークンに署名します。クラスターにアクセスする場合、署名付きトークンを CMU に指定します。

クォーラム認証と 2FA

クラスターは、クォーラム認証と 2FA に同じキーを使用します。これは、2FA が有効なユーザーが M of N アクセスコントロール (MofN) に登録されていることを意味します。同じ HSM ユーザーに対して 2FA 認証とクォーラム認証を正常に実行する際、次の点を考慮する必要があります。

- 現在、ユーザーに対してクォーラム認証を使用している場合は、クォーラムのユーザーに対して作成したものと同じキーペアを使用し、ユーザーに対して 2FA を有効化する必要があります。

- クォーラム認証ユーザーではない非 2FA ユーザーに 2FA 要件を追加する場合、そのユーザーを 2FA 認証で MofN ユーザーとして登録します。
- 2FA 要件を削除するか、クォーラム認証ユーザーでもある 2FA ユーザーのパスワードを変更する場合、クォーラムのユーザーのMofN ユーザーとしての登録も削除されます。
- 2FA 要件を削除するか、クォーラム認証ユーザーでもある 2FA ユーザーのパスワードを変更する場合、それでもそのユーザーがクォーラム認証に加わる必要がある の場合、当該ユーザーを MofN ユーザーとして再登録する必要があります。

認証の詳細情報は、CMU を使用してクォーラム認証を管理する を参照してください。

HSM ユーザー向け 2FA を使用した作業

このセクションでは、2FA HSM ユーザーの作成、キーのローテーション、2FA 有効ユーザーとして HSM へのログインなど、HSM ユーザー向け 2FA の操作方法について説明します。HSM ユーザーの操作方法の詳細は、???、???、???、???、??? を参照してください。

2FA ユーザーの作成

HSM ユーザーに対して 2FA を有効化する場合、次の要件を満たすキーを使用します。

2FA キーペアの要件

新しいキーペアの作成や、以下の要件を満たす既存のキーを使用することもできます。

- キータイプ : 非対称
- キーの使用方法 : 署名と認証
- キースペック: RSA_2048
- 署名アルゴリズムには、以下が含まれます。
  - `sha256WithRSAEncryption`

> **Note**
>
> クォーラム認証を使用している場合、またはクォーラム認証を使用する予定の場合は、the section called “クォーラム認証と 2FA” を参照してください。

CMU とキーペアを使用して、2FA を有効化した新しい CO ユーザーを作成します。

2FA を有効化した状態での CO ユーザー作成

1. 1 つのターミナルで、以下のステップを実行します。

   a. HSM にアクセスし、CloudHSM 管理ユーティリティにログインします。

   ```
   /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   b. CO としてログインし、以下のコマンドを使用して 2FA で新しいユーザー MFA を作成します。

   ```
   aws-cloudhsm>createUser CO MFA <CO USER NAME> -2fa /home/ec2-user/authdata
   ************************CAUTION*********************************This is a
    CRITICAL operation, should be done on all nodes in the
   cluster. AWS does NOT synchronize these changes automatically with the
   nodes on which this operation is not executed or failed, please
   ensure this operation is executed on all nodes in the cluster.
   ************************************************************

   Do you want to continue(y/n)?

   yCreating User exampleuser3(CO) on 1 nodesAuthentication data written to: "/
   home/ec2-user/authdata"Generate Base64-encoded signatures for SHA256 digests in
    the authentication datafile.
   To generate the signatures, use the RSA private key, which is the second factor
    ofauthentication for this user. Paste the signatures and the corresponding
    public keyinto the authentication data file and provide
   the file path below.Leave this field blank to use the path initially
    provided.Enter filename:
   ```

   c. 上記のターミナルは、この状態のままにしておきます。Enter キーを押したり、ファイル名を入力したりしないでください。

2. 別のターミナルで、以下のステップを実行します。

   a. HSM にアクセスし、CloudHSM 管理ユーティリティにログインします。

   ```
   /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   b. 以下のコマンドを使用して、パブリックプライベートキーペアを生成します。

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt
 rsa_keygen_bits:2048
```

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

c. 以下のコマンドを実行して、authdata ファイルからダイジェストを抽出するための JSON クエリ機能をインストールします。

```
sudo yum install jq
```

d. ダイジェスト値を抽出するには、まず authdata ファイル内の以下のデータを検索します。

```
{
  "Version":"1.0",
  "PublicKey":"",
  "Data":[
    {
      "HsmId": <"HSM ID">,
      "Digest": <"DIGEST">,
      "Signature":""
    }
  ]
}
```

> **Note**
>
> 取得したダイジェストは base64 でエンコードされていますが、ダイジェストに署名するには、まずファイルをデコードしてから署名する必要があります。次のコマンドはダイジェストをデコードし、デコードされたコンテンツを 「digest1.bin」に保存します
>
> ```
> cat authdata | jq '.Data[0].Digest' | cut -c2- | rev | cut -c2- | rev |
>  base64 -d > digest1.bin
> ```

e. パブリックキーの内容を変換し、次に示すように「\n」を追加し、スペースを削除します。

```
-----BEGIN PUBLIC KEY-----\n<PUBLIC KEY>\n-----END PUBLIC KEY-----
```

> **Important**
>
> 上記のコマンドは、BEGIN PUBLIC KEY----- の直後に「\n」を追加する方法、「\n」とパブリックキーの最初の文字の間のスペースを削除する方法、-----END PUBLIC KEY の前に「\n」を追加する方法、および「\n」とパブリックキーの末尾の間のスペースを削除する方法を示しています。

これはパブリックキーの PEM 形式で、認証データファイルで受け入れられます。

f. パブリックキー PEM 形式のコンテンツを authdata ファイルのパブリックキーセクションに貼り付けます。

```
vi authdata
```

```
{
  "Version":"1.0",
  "PublicKey":"-----BEGIN PUBLIC KEY-----\n<"PUBLIC KEY">\n-----END PUBLIC
 KEY-----",
  "Data":[
{
  "HsmId":<"HSM ID">,
  "Digest":<"DIGEST">,
  "Signature":””
}
]
}
```

g. 次のコマンドを使用してトークン ファイルに署名します。

```
openssl pkeyutl -sign -in digest1.bin -inkey private_key.pem -pkeyopt
 digest:sha256 | base64

Output Expected:
<"THE SIGNATURE">
```

> **Note**
>
> 上記のコマンドで示したように、署名には openssl dgst の代わりに openssl pkeyutl を使用してください。

h. Authdata ファイルの「署名」フィールドに署名済みダイジェストを追加します。

```
vi authdata
```

```
{
    "Version": "1.0",
    "PublicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----",
    "Data": [
        {
            "HsmId": <"HSM ID">,
            "Digest": <"DIGEST">,
            "Signature": "Kkdl ... rkrvJ6Q=="
        },
        {
            "HsmId": <"HSM ID">,
            "Digest": <"DIGEST">,
            "Signature": "K1hxy ... Q261Q=="
        }
    ]
}
```

3. 最初のターミナルに戻り、**Enter** を押します。

```
Generate Base64-encoded signatures for SHA256 digests in the authentication
 datafile. To generate the signatures, use the RSA private key,
which is the second factor ofauthentication for this user. Paste the signatures and
 the corresponding public keyinto the authentication data file and provide the file
 path below. Leave this field blank to use the path initially provided.
Enter filename: >>>>> Press Enter here

createUser success on server 0(10.0.1.11)
```

HSM ユーザー用 2FA の管理

2FA ユーザーのパスワードの変更、2FA の有効化または無効化、2FA キーのローテーション化を行う場合、[パスワードの変更] を使用します。2FA を有効化にするたびに、2FA ログイン用のパブリックキーを指定する必要があります。

パスワードの変更では、以下のシナリオが実行されます。

- 2FA ユーザーのパスワード変更
- 非 2FA ユーザーのパスワード変更
- 非 2FA ユーザーに 2FA の追加
- 2FA ユーザーから 2FA の削除
- 2FA ユーザーのキーのローテーション化

また、タスクを組み合わせることもできます。たとえば、ユーザーから 2FA を削除すると同時にパスワードの変更や、2FA キーをローテーション化してユーザーパスワードの変更を行うことができます。

2FA が有効な CO ユーザーのパスワードの変更、またはキーのローテーション化

1. CMU を使用し、2FA が有効な CO として HSM にログインします。
2. changePswd を使用して、2FA が有効な CO ユーザーからパスワードを変更するか、キーをローテーション化します。`-2fa` パラメータを使用して、システムが `authdata` ファイルに書き込むファイルシステム内の位置を含みます。このファイルには、クラスター内の各 HSM のダイジェストが含まれています。

   ```
   aws-cloudhsm>changePswd CO example-user <new-password> -2fa /path/to/authdata
   ```

   CMU は、プライベートキーを使用して、`authdata` ファイル内のダイジェストに署名を要求するプロンプトが表示され、署名はパブリックキー付きで返却されます。
3. プライベートキーを使用して、`authdata` ファイル内のダイジェストに署名し、署名とパブリックキーを JSON 形式の `authdata` ファイルに追加後、CMU に `authdata` ファイルの位置を追加します。詳細情報は、the section called “設定リファレンス” を参照してください。

> **Note**
>
> クラスターは、クォーラム認証と 2FA に同じキーを使用します。クォーラム認証を使用している場合、またはクォーラム認証を使用する予定の場合は、the section called “クォーラム認証と 2FA” を参照してください。

2FA が有効な CO ユーザーの 2FA の無効化

1. CMU を使用し、2FA が有効な CO として HSM にログインします。
2. changePswd を使用して、2FA が有効な CO ユーザーから 2FA を削除します。

```
aws-cloudhsm>changePswd CO example-user <new password>
```

CMU は、パスワードの変更オペレーションを要求するプロンプトを表示します。

> **Note**
>
> 2FA 要件を削除するか、クォーラム認証ユーザーでもある 2FA ユーザーのパスワードを変更する場合、クォーラムのユーザーのMofN ユーザーとしての登録も削除されます。クォーラムユーザーおよび 2FA の詳細情報は、[the section called “クォーラム認証と 2FA”] を参照してください。

3. タイプ **y**。

CMU は、パスワードの変更オペレーションを確定します。

設定リファレンス

CMU で生成されたリクエストとレスポンスの両方に対応する `authdata` ファイル内の 2FA プロパティの例を以下に示します。

```
{
    "Version": "1.0",
    "PublicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----",
    "Data": [
        {
            "HsmId": "hsm-lgavqitns2a",
```

```
            "Digest": "k5O1p3f6foQRVQH7S8Rrjcau6h3TYqsSdr16A54+qG8=",
            "Signature": "Kkdl ... rkrvJ6Q=="
        },
        {
            "HsmId": "hsm-lgavqitns2a",
            "Digest": "IyBcx4I5Vyx1jztwvXinCBQd9lDx8oQe7iRrWjBAi1w=",
            "Signature": "K1hxy ... Q261Q=="
        }
    ]
}
```

[データ]

最上位のノード。クラスター内の各 HSM の下位ノードが含まれています。すべての 2FA コマンドのリクエストとレスポンスに表示されます。

ダイジェスト

これは、認証の 2 番目の要素を提供するために署名が必要です。すべての 2FA コマンドのリクエストで生成された CMU。

HsmId

ご利用のHSM の IDです。すべての 2FA コマンドのリクエストとレスポンスに表示されます。

PublicKey

生成したキーペアのパブリックキーの部分は、PEM 形式の文字列として挿入されます。createUser と changePswd の回答欄にこれを入力します。

署名

Base 64 でエンコードされた署名付きダイジェスト。2FA コマンドの回答欄にこれを入力します。

バージョン

認証データ JSON 形式のファイルのバージョン。すべての 2FA コマンドのリクエストとレスポンスに表示されます。

CloudHSM 管理ユーティリティ (CMU) を使用したクォーラム認証の管理 (M of N アクセスコントロール)

AWS CloudHSM クラスター内の HSMsクォーラム認証をサポートしています。クォーラム認証は M of N アクセスコントロールとも呼ばれます。クォーラム認証を使用すると、HSM の単一のユー

ザーは HSM でクォーラム管理されたオペレーションを行うことができません。代わりに、HSM ユーザーの最小数 (少なくとも 2 人) が、これらのオペレーションを協力して行う必要があります。クォーラム認証を使用すると、複数の HSM ユーザーからの承認を要求することで、さらに保護レイヤーを追加できます。

クォーラム認証は次のオペレーションを制御できます。

- Crypto officer (CO) による HSM ユーザーの管理 HSM ユーザーの作成と削除、および、別の HSM ユーザーのパスワードの変更。詳細については、「Crypto Officer 用のクォーラム認証の使用」を参照してください。

次のトピックでは、 AWS CloudHSMでのクォーラム認証についてさらに詳細な情報を提供します。

トピック

クォーラム認証の概要

以下のステップは、クォーラム認証のプロセスの概要を示しています。特定のステップとツールについては、Crypto Officer 用のクォーラム認証の使用 を参照してください。

1. 各 HSM ユーザーは署名のための非対称キーを作成します。これは HSM の外部で行い、キーを適切に保護します。
2. 各 HSM ユーザーは HSM にログインし、署名キーの公開部分 (パブリックキー) を HSM に登録します。
3. HSM ユーザーがクォーラム管理されたオペレーションを実行する場合、各ユーザーが HSM にログインし、クォーラムトークンを取得します。
4. HSM ユーザーは、クォーラムトークンを 1 人または複数の他の HSM ユーザーに付与し、承認を求めます。
5. 他の HSM ユーザーは、キーを使用してクォーラムトークンに暗号で署名することにより承認します。これは HSM の外部で行われます。

6. HSM ユーザーは、必要な数の承認を得ると、HSM にログインし、クォーラムトークンと承認 (署名) を HSM に渡します。
7. HSM では、それぞれの署名した人の登録されたパブリックキーを使用して署名を確認します。署名が有効である場合、HSM はトークンを承認します。
8. HSM ユーザーはクォーラム管理されたオペレーションを実行できます。

クォーラム認証に関するその他の詳細

AWS CloudHSMでのクォーラム認証の使用に関する次の追加の情報に注意してください。

- HSM ユーザーは自分のクォーラムトークンに署名できます。つまり、リクエストするユーザーはクォーラム認証に必要な承認の 1 つを提供できます。
- クォーラム管理されたオペレーションに対して、最小数のクォーラム承認者を選択します。選択できる最小数は 2 で、選択できる最大数は 8 です。
- HSM はクォーラムトークンを最大 1024 保存できます。HSM にすでに 1024 トークンある場合、新しく作成しようとすると、HSM は期限切れのトークンの 1 つを消去します。デフォルトでは、トークンは作成後 10 分で有効期限が切れます。
- クラスターは、クォーラム認証と 2 要素認証 (2FA) に同じキーを使用します。クォーラム認証と 2FA の使用の詳細については、クォーラム認証と 2FA を参照してください。

Crypto Officers のクォーラム認証の使用: 初回セットアップ

次のトピックでは、Crypto Officers (CO) がクォーラム認証を使用できるようにするためのハードウェアセキュリティモジュール (HSM)の設定に必要なステップについて説明します。CO のクォーラム認証を最初に設定する場合に、これらのステップを 1 回だけ実行する必要があります。これらのステップが完了したら、Crypto Officer 用のクォーラム認証の使用 を参照してください。

トピック

- 前提条件
- 署名のためのキーの作成と登録
- HSM のクォーラム最小値を設定する

前提条件

この例の理解には、cloudhsm_mgmt_util (CMU)コマンドラインツール についての知識が必要です。この例では、 listUsers コマンドからの次の出力に示すように、 AWS CloudHSM クラスターには 2

HSMs COsを持ちます。ユーザー作成の詳細については、HSM ユーザーの管理 を参照してください。

```
aws-cloudhsm>listUsers
Users on server 0(10.0.2.14):
Number of users found:7

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                NO
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              officer1                             NO
           0               NO
         4              CO              officer2                             NO
           0               NO
         5              CO              officer3                             NO
           0               NO
         6              CO              officer4                             NO
           0               NO
         7              CO              officer5                             NO
           0               NO
Users on server 1(10.0.1.4):
Number of users found:7

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                NO
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              officer1                             NO
           0               NO
         4              CO              officer2                             NO
           0               NO
         5              CO              officer3                             NO
           0               NO
         6              CO              officer4                             NO
           0               NO
         7              CO              officer5                             NO
           0               NO
```

署名のためのキーの作成と登録

クォーラム認証を使用する場合、各 CO が以下の すべて のステップをを実行する必要があります。

トピック

- RSA キーペアの作成
- 登録トークンの作成と署名
- HSM でパブリックキーを登録する

RSA キーペアの作成

様々なキーペアを作成、保護する方法があります。次の例では、OpenSSL 使用方法を説明しています。

Example — OpenSSL でプライベートキーを作成する

次の例は、OpenSSL を使用してパスフレーズで保護された 2048 ビットの RSA キーを作成する方法を示しています。この例を使用するには、*officer1.key* を、キーの保存先のファイル名に置き換えてください。

```
$ openssl genrsa -out officer1.key -aes256 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
Enter pass phrase for officer1.key:
Verifying - Enter pass phrase for officer1.key:
```

次に、作成したプライベートキーを使用してパブリックキーを生成します。

Example — OpenSSL でパブリックキーを作成する

以下の例は、OpenSSL を使用して先ほど作成したプライベートキーからパブリックキーを作成する方法を示しています。

```
$ openssl rsa -in officer1.key -outform PEM -pubout -out officer1.pub
Enter pass phrase for officer1.key:
writing RSA key
```

登録トークンの作成と署名

トークンを作成し、前のステップで生成したプライベートキーを使用して署名します。

Example — トークンを作成する

登録トークンは、最大 245 バイトのサイズを超えないランダムなデータを含むファイルのみです。プライベートキーを使用してトークンに署名し、プライベートキーへのアクセス権があることを示します。次のコマンドは、echo を使用して文字列をファイルにリダイレクトします。

```
$ echo "token to be signed" > officer1.token
```

トークンに署名し、署名ファイルに保存します。HSM で MofN ユーザーとして CO を登録する場合、署名付きトークン、署名なしトークン、およびパブリックキーが必要です。

Example トークンへ署名する

OpenSSL とプライベートキーを使用して登録トークンに署名し、署名ファイルを作成します。

```
$ openssl dgst -sha256 \
    -sign officer1.key \
    -out officer1.token.sig officer1.token
```

HSM でパブリックキーを登録する

キーを作成した後、CO はキーのパブリックパート (パブリックキー) を HSM に登録する必要があります。

HSM にパブリックキーの登録するには

1. 次のコマンドを使用して、cloudhsm_mgmt_util コマンドラインツールをスタートします。

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

2. loginHSM コマンドを使用して、CO ユーザーとして HSM にログインします。詳細については、「???」を参照してください。
3. registerQuorumPubKey コマンドを使用してパブリックキーを登録します。詳細については、次の例を参照するか、または help registerQuorumPubKey コマンドを使用してください。

Example - HSM のパブリックキーを登録する

以下の例では、cloudhsm_mgmt_util コマンドラインツールで registerQuorumPubKey コマンドを使用して、CO のパブリックキーを HSM に登録する方法を示しています。このコマンドを使用するには、CO が HSM にログインしている必要があります。以下の値を自分の値に置き換えてください。

```
aws-cloudhsm> registerQuorumPubKey CO <officer1> <officer1.token> <officer1.token.sig>
 <officer1.pub>

************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
registerQuorumPubKey success on server 0(10.0.2.14)
```

<officer1.token>

署名なし登録トークンを含むファイルへのパスです。最大ファイルサイズが 245 バイトの任意のランダムデータを持つことができます。

必須 : はい

<officer1.token.sig>

登録トークンの SHA256_PKCS メカニズム署名付きハッシュを含むファイルへのパスです。

必須: はい

<officer1.pub>

非対称 RSA-2048 キーペアの公開キーを含むファイルへのパスです。プライベートキーを使用して、登録トークンに署名します。

必須: はい

次の例に示すように、すべての CO がパブリックキーを登録した後、listUsers コマンドの出力の MofnPubKey 列にこれが表示されます。

```
aws-cloudhsm>listUsers
```

```
Users on server 0(10.0.2.14):
Number of users found:7

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                             NO
           0               NO
         2              AU              app_user                          NO
           0               NO
         3              CO              officer1                          YES
           0               NO
         4              CO              officer2                          YES
           0               NO
         5              CO              officer3                          YES
           0               NO
         6              CO              officer4                          YES
           0               NO
         7              CO              officer5                          YES
           0               NO
Users on server 1(10.0.1.4):
Number of users found:7

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                             NO
           0               NO
         2              AU              app_user                          NO
           0               NO
         3              CO              officer1                          YES
           0               NO
         4              CO              officer2                          YES
           0               NO
         5              CO              officer3                          YES
           0               NO
         6              CO              officer4                          YES
           0               NO
         7              CO              officer5                          YES
           0               NO
```

HSM のクォーラム最小値を設定する

CO のクォーラム認証を使用するには、CO が HSM にログインして、m 値とも呼ばれるクォーラム最小値を設定する必要があります。これは、HSM ユーザー管理オペレーションを実行するために必

要な CO 承認の最小数です。HSM 上の任意の CO は、署名用のキーを登録していない CO を含むクォーラム最小値を設定できます。クォーラム最小値はいつでも変更できます。詳細情報は、 最小値を変更 を参照してください。

HSM のクォーラム最小値の設定

1. 次のコマンドを使用して、cloudhsm_mgmt_util コマンドラインツールをスタートします。

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. loginHSM コマンドを使用して、CO ユーザーとして HSM にログインします。詳細については、「???」を参照してください。
3. クォーラム最小値を設定する場合、setMValue コマンドを使用します。詳細については、次の例を参照するか、または help setMValue コマンドを使用してください。

Example HSM のクォーラム最小値を設定する

この例では、クォーラム最小値 2 を使用します。最大は HSM 上の CO の合計数で、2 から 8 までの任意の値を選択できます。この例では HSM に 6 つの CO がいるため、指定可能な最大値は 6 です。

次のコマンド例を使用するには、最後の数値 (*2*) を所望のフォーラム最小値に置き換えてください。

```
aws-cloudhsm>setMValue 3 2
*************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****************************************************************

Do you want to continue(y/n)?y
Setting M Value(2) for 3 on 2 nodes
```

上記の例では、最初の数字 (3) は、クォーラム最小値を設定しようとしている HSM サービスを示しています。

次の表に、HSM サービス識別子とその名前、説明、およびサービスに含まれるコマンドを示しています。

サービス識別子	サービス名	サービスの説明	HSM コマンド
3	USER_MGMT	HSM ユーザー管理	• createUser • deleteUser • changePswd (別の HSM ユーザーのパスワードを変更した場合のみ適用されます)
4	MISC_CO	その他の CO サービス	• setMValue

サービスのクォーラム最小値を取得するには、次の例のように、getMValue コマンドを使用します。

```
aws-cloudhsm>getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

前述の getMValue コマンドの出力は、HSM ユーザー管理オペレーション (サービス 3) のクォーラム最小値が 2 になったことを示しています。

これらのステップが完了したら、Crypto Officer 用のクォーラム認証の使用 を参照してください。

Crypto Officer 用のクォーラム認証の使用

HSM の Crypto Officer (CO) は、HSM の以下のオペレーションに対してクォーラム認証を設定できます。

- HSM ユーザーの作成
- HSM ユーザーの削除
- 別の HSM ユーザーのパスワードの変更

HSM をクォーラム認証用に設定した後では、CO が単独で HSM ユーザー管理オペレーションを実行することはできません。次の例は、CO が HSM で新しいユーザーを作成しようとしたときの出力

を示しています。コマンドは RET_MXN_AUTH_FAILED エラーとなり、クォーラム認証が失敗したことを示しています。

```
aws-cloudhsm>createUser CU user1 password
************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?y
Creating User user1(CU) on 2 nodes
createUser failed: RET_MXN_AUTH_FAILED
creating user on server 0(10.0.2.14) failed

Retry/Ignore/Abort?(R/I/A):A
```

HSM ユーザー管理オペレーションを実行するには、CO は以下のタスクを完了する必要があります。

1. クォーラムトークン の取得
2. 他の CO からの承認 (署名) の取得
3. HSM でのトークンの承認
4. HSM ユーザー管理オペレーションの実行

HSM をクォーラム認証用にまだ設定していない場合は、今すぐに設定してください。詳細については、「初回の設定」を参照してください。

クォーラムトークンの取得

まず、CO は cloudhsm_mgmt_util コマンドラインツールを使用して クォーラムトークン をリクエストする必要があります。

クォーラムトークンを取得するには

1. 次のコマンドを使用して、cloudhsm_mgmt_util コマンドラインツールをスタートします。

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. loginHSM コマンドを使用して、CO ユーザーとして HSM にログインします。詳細については、「???」を参照してください。
3. getToken コマンドを使用してクォーラムトークンを取得します。詳細については、次の例を参照するか、または help getToken コマンドを使用してください。

Example クォーラムトークンを取得する

この例では、ユーザー名を officer1 とする CO のクォーラムトークンを取得し、そのトークンを `officer1.token` というファイルに保存します。この例のコマンドを使用するには、以下の値を独自のものに置き換えてください。

- *`officer1`* トークンを取得する CO の名前。HSM にログインし、このコマンドを実行している CO と同じであることが必要です。
- *`officer1.token`* クォーラムトークンを保存するファイルの名前。

次のコマンドで、3 は取得するトークンを使用できる サービス を識別します。この例のトークンは、HSM ユーザー管理オペレーション (サービス 3) で使用できます。詳細については、「HSM のクォーラム最小値を設定する」を参照してください。

```
aws-cloudhsm>getToken 3 officer1 officer1.token
getToken success on server 0(10.0.2.14)
Token:
Id:1
Service:3
Node:1
Key Handle:0
User:officer1
getToken success on server 1(10.0.1.4)
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
```

承認 CO からの署名の取得

クォーラムトークンを持つ CO は、そのトークンを他の CO に承認してもらう必要があります。他の CO は、承認を与えるために、署名キーを使用してトークンを暗号で署名します。この署名は HSM 外で行われます。

トークンの署名にはさまざまな方法が使用されます。次の例では、OpenSSL を使用しています。別の署名ツールを使用する場合は、そのツールで必ず CO のプライベートキー (署名キー) を使用してトークンの SHA-256 ダイジェストに署名します。

Example 承認 CO からの署名を取得する

この例では、トークン (officer1) を持つ CO に少なくとも 2 つの承認が必要です。以下のコマンド例では、2 つの CO が OpenSSL を使用してトークンに暗号で署名する方法を示します。

最初のコマンドでは、officer1 が自分のトークンに署名します。以下のコマンド例を使用するには、以下の値を独自のものに置き換えてください。

- *officer1.key* および *officer2.key* CO の署名キーが含まれているファイルの名前。
- *officer1.token.sig1* および *officer1.token.sig2* 署名を保存するファイルの名前。署名ごとに別のファイルに保存します。
- *officer1.token* CO が署名するトークンが格納されるファイルの名前。

```
$ openssl dgst -sha256 -sign officer1.key -out officer1.token.sig1 officer1.token
Enter pass phrase for officer1.key:
```

次のコマンドでは、officer2 が同じトークンに署名します。

```
$ openssl dgst -sha256 -sign officer2.key -out officer1.token.sig2 officer1.token
Enter pass phrase for officer2.key:
```

HSM での署名済みトークンの承認

CO は、他の CO から最小限の数の承認 (署名) を取得したら、署名済みトークンを HSM で承認する必要があります。

HSM で署名済みトークンの承認

1. トークン承認ファイルを作成します。詳細については、次の例を参照してください。
2. 次のコマンドを使用して、cloudhsm_mgmt_util コマンドラインツールをスタートします。

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

3. loginHSM コマンドを使用して、CO ユーザーとして HSM にログインします。詳細については、「???」を参照してください。
4. approveToken コマンドを使用して署名済みトークンを承認し、トークン承認ファイルを渡します。詳細については、次の例を参照してください。

Example トークン承認ファイルを作成し、署名済みトークンを HSM で承認する

トークン承認ファイルは、HSM で必要とされる特別な形式のテキストファイルです。このファイルには、トークン、その承認者、および承認者の署名が含まれます。トークン承認ファイルの例は次のとおりです。

```
# For "Multi Token File Path", type the path to the file that contains
# the token. You can type the same value for "Token File Path", but
# that's not required. The "Token File Path" line is required in any
# case, regardless of whether you type a value.
Multi Token File Path = officer1.token;
Token File Path = ;

# Total number of approvals
Number of Approvals = 2;

# Approver 1
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer1;
Approval File = officer1.token.sig1;

# Approver 2
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer2;
Approval File = officer1.token.sig2;
```

トークン承認ファイルの作成後、CO は cloudhsm_mgmt_util コマンドラインツールを使用して HSM にログインします。次に CO は approveToken コマンドを使用し、以下の例に示すように、トークンを承認します。*approval.txt* は、トークン承認ファイルの名前に置き換えてください。

```
aws-cloudhsm>approveToken approval.txt
approveToken success on server 0(10.0.2.14)
approveToken success on server 1(10.0.1.4)
```

このコマンドが成功すると、HSM でのクォーラムトークンの承認が完了します。トークンのステータスを確認するには、次の例に示すように、listTokens コマンドを使用します。コマンドの出力は、トークンに必要な数の承認があることを示しています。

トークンの有効期間は、トークンが HSM に保持される保証期間を示します。トークンの有効期間が過ぎた (ゼロ秒) 後でも、トークンを使用できます。

```
aws-cloudhsm>listTokens

======================
    Server 0(10.0.2.14)
======================
-------- Token - 0 ----------
Token:
Id:1
Service:3
Node:1
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
Current num of approvals : 2
Approver-0: officer1
Approver-1: officer2
Num of tokens = 1

======================
    Server 1(10.0.1.4)
======================
-------- Token - 0 ----------
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
```

```
Current num of approvals : 2
Approver-0: officer1
Approver-1: officer2
Num of tokens = 1

listTokens success
```

ユーザー管理オペレーションでのトークンの使用

前のセクションで示したように、トークンに必要な数の承認を取得すると、CO は以下のいずれかの HSM ユーザー管理オペレーションを実行できます。

- createUser コマンドを使用して HSM ユーザーを作成する
- deleteUser コマンドを使用して HSM ユーザーを削除する
- changePswd コマンドを使用して別の HSM ユーザーのパスワードを変更する

これらのコマンドの詳しい使用方法については、HSM ユーザーの管理 を参照してください。

CO は、トークンを 1 つのオペレーションにのみ使用できます。そのオペレーションが成功すると、トークンは無効になります。別の HSM ユーザー管理オペレーションを行うには、新しいクォーラムトークンを取得し、承認者から新しい署名を取得して、その新しいトークンを HSM で承認する必要があります。

> **Note**
>
> MofN トークンは、現在のログインセッションが開いている間だけ有効です。cloudhsm_mgmt_util からログアウトするか、ネットワーク接続が切断された場合、トークンは無効になります。同様に、承認されたトークンは cloudhsm_mgmt_util 内でのみ使用でき、他のアプリケーションでの認証には使用できません。

次のコマンド例で、CO は HSM で新しいユーザーを作成しています。

```
aws-cloudhsm>createUser CU user1 password
**************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************
```

```
Do you want to continue(y/n)?y
Creating User user1(CU) on 2 nodes
```

前のコマンドが成功すると、後続の listUsers コマンドに新しいユーザーが表示されます。

```
aws-cloudhsm>listUsers
Users on server 0(10.0.2.14):
Number of users found:8

    User Id             User Type       User Name                      MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                           NO
           0               NO
         2              AU              app_user                        NO
           0               NO
         3              CO              officer1                       YES
           0               NO
         4              CO              officer2                       YES
           0               NO
         5              CO              officer3                       YES
           0               NO
         6              CO              officer4                       YES
           0               NO
         7              CO              officer5                       YES
           0               NO
         8              CU              user1                           NO
           0               NO
Users on server 1(10.0.1.4):
Number of users found:8

    User Id             User Type       User Name                      MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                           NO
           0               NO
         2              AU              app_user                        NO
           0               NO
         3              CO              officer1                       YES
           0               NO
         4              CO              officer2                       YES
           0               NO
         5              CO              officer3                       YES
           0               NO
```

```
        6              CO             officer4                                 YES
          0               NO
        7              CO             officer5                                 YES
          0               NO
        8              CU             user1                                     NO
          0               NO
```

CO が別の HSM ユーザー管理オペレーションを実行しようとすると、次の例に示すように、クォーラム認証エラーが発生して失敗します。

```
aws-cloudhsm>deleteUser CU user1
Deleting user user1(CU) on 2 nodes
deleteUser failed: RET_MXN_AUTH_FAILED
deleteUser failed on server 0(10.0.2.14)

Retry/rollBack/Ignore?(R/B/I):I
deleteUser failed: RET_MXN_AUTH_FAILED
deleteUser failed on server 1(10.0.1.4)

Retry/rollBack/Ignore?(R/B/I):I
```

listTokens コマンドは、つぎの例のように、CO に承認済みトークンがないことを示しています。別の HSM ユーザー管理オペレーションを実行するには、新しいクォーラムトークンを取得し、承認者から新しい署名を取得して、その新しいトークンを HSM で承認する必要があります。

```
aws-cloudhsm>listTokens

=====================
    Server 0(10.0.2.14)
=====================
Num of tokens = 0

=====================
    Server 1(10.0.1.4)
=====================
Num of tokens = 0

listTokens success
```

Crypto Officer のクォーラム最小値を変更する

Crypto Officer (CO) がクォーラム認証を使用できるように クォーラム最小値を設定 した後で、クォーラム最小値を変更する場合があります。承認者の数が現在のクォーラム最小値以上の場合のみ、HSM はクォーラム最小値の変更を許可します。たとえば、クォーラム最小値が 2 の場合、クォーラム最小値を変更するには最低 2 つの CO の承認が必要です。

クォーラム最小値の変更のためにクォーラムの承認を取得する場合、setMValue コマンド (サービス 4) の クォーラムトークン が必要です。setMValue コマンド (サービス 4) のクォーラムトークンを取得するには、サービス 4 のクォーラム最小値を 1 より大きくする必要があります。つまり、CO (サービス 3) のクォーラム最小値を変更するには、サービス 4 のクォーラム最小値の変更が必要になる場合があります。

次の表に、HSM サービス識別子とその名前、説明、およびサービスに含まれるコマンドを示しています。

サービス識別子	サービス名	サービスの説明	HSM コマンド
3	USER_MGMT	HSM ユーザー管理	• createUser • deleteUser • changePswd (別の HSM ユーザーのパスワードを変更した場合のみ適用されます)
4	MISC_CO	その他の CO サービス	• setMValue

Crypto Officer のクォーラム最小値を変更するには

1. 次のコマンドを使用して、cloudhsm_mgmt_util コマンドラインツールをスタートします。

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. loginHSM コマンドを使用して、CO ユーザーとして HSM にログインします。詳細については、「???」を参照してください。

3. getMValue コマンドを使用して、サービス 3 のクォーラム最小値を取得します。詳細については、次の例を参照してください。
4. getMValue コマンドを使用して、サービス 4 のクォーラム最小値を取得します。詳細については、次の例を参照してください。
5. サービス 4 のクォーラム最小値がサービス 3 の値より小さい場合、setMValue コマンドを使用してサービス 4 の値を変更します。サービス 4 の値を、サービス 3 の値と同じまたはそれより大きい値に変更します。詳細については、次の例を参照してください。
6. サービス 4 をトークンを使用できるサービスとして指定するように注意し、クォーラムトークン を取得 します。
7. 他の CO からの承認 (署名) の取得
8. HSM でのトークンの承認
9. setMValue コマンドを使用して、サービス 3 (CO が実行するユーザー管理オペレーション) のクォーラム最小値を変更します。

Example - クォーラム最小値を取得してサービス 4 の値を変更する

次のコマンド例では、サービス 3 のクォーラム最小値が現在 2 であることを示しています。

```
aws-cloudhsm>getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

次のコマンド例では、サービス 4 のクォーラム最小値が現在 1 であることを示しています。

```
aws-cloudhsm>getMValue 4
MValue of service 4[MISC_CO] on server 0 : [1]
MValue of service 4[MISC_CO] on server 1 : [1]
```

サービス 4 のクォーラム最小値を変更するには、サービス 3 の値と同じがそれより大きい値を設定して setMValue コマンドを使用します。次の例では、サービス 4 のクォーラム最小値をサービス 3 に設定されているのと同じ値である 2 に設定します。

```
aws-cloudhsm>setMValue 4 2
************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
```

```
ensure this operation is executed on all nodes in the cluster.
*************************************************************

Do you want to continue(y/n)?y
Setting M Value(2) for 4 on 2 nodes
```

次のコマンド例は、サービス 3 とサービス 4 でクォーラム最小値が現在 2 であることを示しています。

```
aws-cloudhsm>getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

```
aws-cloudhsm>getMValue 4
MValue of service 4[MISC_CO] on server 0 : [2]
MValue of service 4[MISC_CO] on server 1 : [2]
```

でのキーの管理 AWS CloudHSM

では AWS CloudHSM、次のいずれかを使用して、クラスター内の HSMsのキーを管理します。

- PKCS #11 ライブラリ
- JCE プロバイダー
- CNG および KSP プロバイダー
- CloudHSM CLI

キーを管理する前には、暗号ユーザー (CU) のユーザー名とパスワードを使用して HSM にログインします。CU だけがキーを作成できます。キーを作成した CU は、その鍵を所有および管理します。

トピック

- でのキー同期と耐久性の設定 AWS CloudHSM
- での AES キーラッピング AWS CloudHSM
- での信頼されたキーの使用 AWS CloudHSM
- CloudHSM CLI によるキーの管理
- KMU と CMU によるキーの管理

でのキー同期と耐久性の設定 AWS CloudHSM

このトピックでは AWS CloudHSM、 のキー同期設定、お客様がクラスターでキーを操作する際に直面する一般的な問題、およびキーをより耐久性の高いものにするための戦略について説明します。

トピック

概念

Token keys

キー生成、インポート、またはラップ解除オペレーション中に作成する永続キー。 は、クラスター全体でトークンキーを AWS CloudHSM 同期します。

Session keys

クラスター内の 1 つのハードウェアセキュリティモジュール (HSM) にのみ存在するエフェメラルキー。 AWS CloudHSM は、クラスター間でセッションキーを同期しません。

Client-side key synchronization

キーの生成、インポート、またはアンラップ操作中に作成したトークンキーをクローンするクライアント側のプロセス。少なくとも 2 つの HSM を用いてクラスターを実行することで、トークンキーの耐久性を高めることができます。

Server-side key synchronization

クラスター内のすべての HSM に対して定期的にキーをクローンします。管理は必要ありません。

Client key durability settings

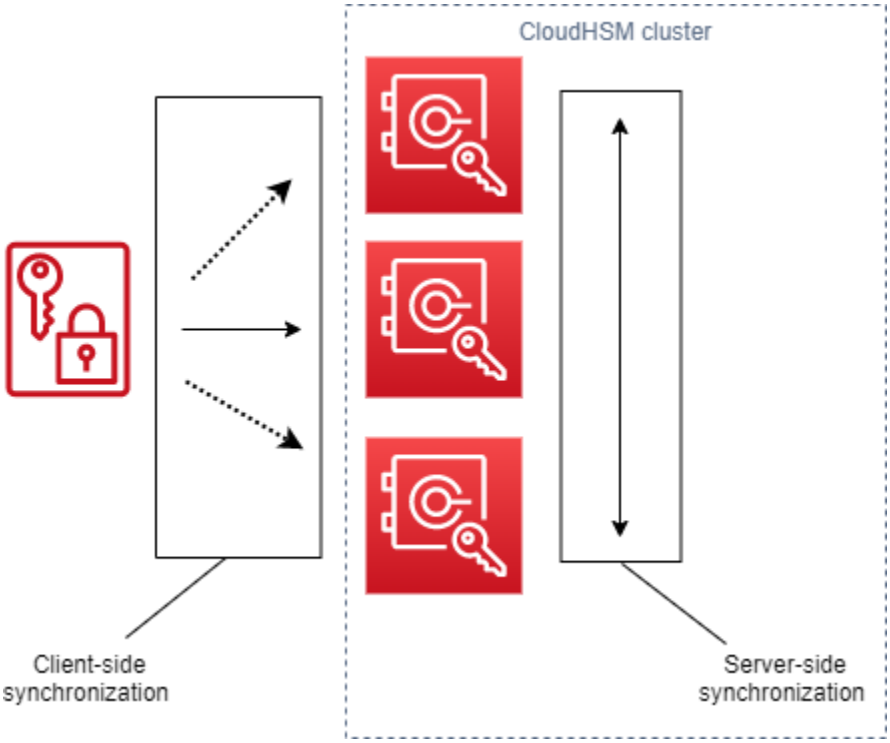
キーの耐久性に影響するクライアント上に構成した設定。[Client SDK 5] と [Client SDK 3] では動作が異なります。

- クライアント SDK 5 では、この設定を使用して、単一の HSM クラスターを実行します。

- クライアント SDK 3 では、この設定を使用して、キー作成オペレーションを成功させるために必要な HSM の数を指定します。

キーの同期について

AWS CloudHSM は、キー同期を使用して、クラスター内のすべての HSMs でトークンキーのクローンを作成します。キーの生成、インポート、またはアンラップ操作中、トークンキーを永続キーとして作成します。クラスターを通してこれらのキーを配信するには、CloudHSM はクライアント側とサーバー側の両方にキーの同期を提供します。



キー同期の目標 (サーバー側とクライアント側の両方) は、新しいキーを作成した後、できるだけ迅速にクラスター全体に、それを配信することです。これは、新しいキーを使用するために実行する後続の呼び出しが、クラスター内の利用可能な HSM にルーティングされる可能性があるため重要です。作成した呼び出しが キーなしで HSM にルーティングされた場合、呼び出しは失敗します。キー作成操作の後に実行される後続の呼び出しをアプリケーションで再試行するように指定することで、これらのタイプの障害を軽減できます。同期に必要な時間は、クラスターやその他の無形オブジェクト上のワークロードによって異なります。 CloudWatch メトリクスを使用して、この種の状況でアプリケーションが採用するタイミングを決定します。詳細については、「CloudWatch メトリクス」を参照してください。

クラウド環境でのキー同期の課題は、キーの耐久性です。1 つの HSM でキーを作成し、多くの場合、それらのキーを使ってすぐに開始します。キーがクラスター内の別のHSMに複製される前に、

その上にキーを作成する HSM に障害が発生した場合は、キーで暗号化されたいかなるものに対するキー および アクセスを失います。このリスクを軽減するために、client-side synchronization を提供します。クライアント側の同期は、キーの生成、インポート、またはアンラップ操作中に作成したキーをクローンするクライアント側のプロセスです。クローニングキーの作成時にクローンを作成すると、キーの耐久性が向上します。もちろん、1 つの HSM を用いてクラスター内のキーのクローンを作成することはできません。キーの耐久性を高めるために、最低 2 つの HSM を使用するようにクラスターを構成することをお勧めします。クライアント側の同期と 2 つの HSM を持つクラスターで、クラウド環境におけるキーの耐久性の課題に対応できます。

クライアントキーの耐久性設定の操作

キーの同期は主に自動プロセスですが、クライアント側のキーの耐久性設定を管理できます。クライアント側のキーの耐久性設定は、クライアント SDK 5 とクライアント SDK 3 では異なる動作をします。

- クライアント SDK 5 では、最低 2 つの HSM を用いてクラスターを実行することを求める key availability quorums のコンセプトを紹介します。クライアント側のキーの耐久性設定を使用して、2 つの HSM 要件をオプトアウトできます。quorums のさらなる詳細については、the section called “Client SDK 5 の概念” を参照してください。
- Client SDK 3 では、クライアント側のキーの耐久性設定を使用して、オペレーション全体を成功と見なすためにキーの作成を成功させる必要がある HSM の数を指定します。

Client SDK 5 のクライアントキーの耐久性設定

クライアント SDK 5 では、キーの同期は完全自動プロセスです。キー可用性クォーラムを用いて、アプリケーションがキーを使用できるようになる前に、新しく作成されたキーがクラスター内の 2 つの HSM 上に存在している必要があります。キーの可用性クォーラムを使用するには、クラスターに最低 2 つの HSM が必要です。

クラスター構成がキーの耐久性要件を満たしていない場合、トークンキーを作成または使用しようとすると、ログに次のエラーメッセージが表示されて失敗します。

```
Key <key handle> does not meet the availability requirements - The key must be
 available on at least 2 HSMs before being used.
```

クライアント構成設定を使用して、キーの可用性クォーラムをオプトアウトできます。たとえば、単一の HSM を用いてクラスターの実行をオプトアウトしたい場合があります。

Client SDK 5 の概念

Key Availability Quorum

AWS CloudHSM は、アプリケーションがキーを使用する前にキーが存在する必要があるクラスター内の HSMs の数を指定します。最低 2 つの HSM を持つクラスターが必要です。

クライアントキーの耐久性設定の管理

クライアントキーの耐久性設定を管理するには、Client SDK 5 用の構成ツールを使用する必要があります。

PKCS #11 library

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --disable-key-
availability-check
```

OpenSSL Dynamic Engine

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

JCE provider

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --disable-key-
availability-check
```

CloudHSM CLI

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-cli --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --disable-key-
availability-check
```

Client SDK 3 のクライアントキーの耐久性設定

[Client SDK 3] では、キーの同期は主に自動プロセスですが、クライアントキーの耐久性設定を使用してキーの耐久性を高めることができます。オペレーション全体を成功と見なすためにキーの作成を引き継ぐ必要がある HSM の数を指定します。クライアント側の同期では、選択した設定に関係なく、クラスター内のすべての HSM にキーのクローンするために、ベストエフォートの試行を行います。この設定では、指定した HSM の数の上にキーの作成が強制されます。値を指定し、システムがその数の HSM にキーをレプリケートできない場合、不要なキーマテリアルは自動的にクリーンアップされ、再試行できます。

> **Important**
>
> クライアントキーの耐久性設定を設定しない場合 (またはデフォルト値の 1 を使用する場合)、キーが失われる恐れがあります。サーバーサイドサービスがそのキーを別の HSM に複製する前に現在の HSM に障害が発生した場合、キーマテリアルは失われます。

キーの耐久性を最大化するには、クライアント側の同期のために少なくとも 2 つの HSM を指定することを検討してください。HSM をいくつ指定しても、クラスター上のワークロードは変わりません。クライアント側の同期では、クラスター内のすべての HSM に対して常にベストエフォート型のキーのクローンが試行されます。

推奨事項

- Minimum: クラスターあたり 2 つの HSM
- Maximum: クラスター内の HSM の総数より 1 少ない

クライアント側の同期に失敗すると、クライアントサービスは作成された可能性があり、不要になった可能性のある不要なキーをクリーンアップします。このクリーンアップは、常に機能するとは限らないベストエフォート対応です。クリーンアップが失敗した場合は、不要なキーマテリアルを削除する必要があります。さらなる詳細については、Key Synchronization Failures を参照してください。

クライアントキーの耐久性のための設定ファイルの設定

クライアントキーの耐久性設定を指定するには、`cloudhsm_client.cfg` を編集します。

クライアント設定ファイルを編集するには

1. cloudhsm_client.cfg を開きます。

   Linux ：

   ```
   /opt/cloudhsm/etc/cloudhsm_client.cfg
   ```

   Windows ：

   ```
   C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
   ```

2. ファイルのclientノードで、キー作成オペレーションを成功させるために が キーを正常に作成 AWS CloudHSM する必要がある HSMsの最小数の値を追加して create_object_minimum_nodes指定します。

   ```
   "create_object_minimum_nodes" : 2
   ```

   > **Note**
   >
   > [key_mgmt_util (KMU)]コマンドラインツール]には、クライアントキーの耐久性の追加設定があります。詳細については、the section called “KMU とクライアント側の同期” を参照してください。

設定リファレンス

これらは、cloudhsm_client.cfg の抜粋中に示されるクライアント側の同期プロパティです:

```
{
    "client": {
        "create_object_minimum_nodes" : 2,
        ...
    },

    ...
}
```

create_object_minimum_nodes

キーの生成、キーのインポート、またはキーアンラップ操作が成功したと見なすために必要な HSM の最小数を指定します。設定されると、デフォルトは、[1] になります。つまり、キーがオペレーション作成するたびに、クライアント側のサービスはクラスター内のすべての HSM でキーを作成しようとしますが、成功を返すためには、クラスター内の HSM 上で single key の生成を必要とするだけです。

KMU とクライアント側の同期

[key_mgmt_util (KMU)] コマンドラインツールを使用してキーを作成する場合は、オプションのコマンドラインパラメータ (`-min_srv`) を使って、キーのクローンを作成する HSM の数を limit します。コマンドラインパラメータと値を設定ファイルで指定すると、 は 2 つの値の LARGER を AWS CloudHSM 優先します。

詳細については、次のトピックを参照してください。

- genDSAKeyPair
- genECCKeyPair
- genRSAKeyPair
- genSymKey
- importPrivateKey
- importPubKey
- imSymKey
- insertMaskedObject
- unWrapKey

クローンされたクラスター間でのキーの同期

クライアント側とサーバー側の同期は、同じ クラスターの中でキーを同期させるためだけのためにあります。クラスターのバックアップを別のリージョンにコピーする場合は、クラスタ間でのキーを同期のために[cloudhsm_mgmt_util (CMU)] 中の [syncKey] コマンドを使用できます。クローンクラスタは、クロスリージョンの冗長性のため、または災害対策プロセスを簡素化するために使用できます。さらなる詳細については、syncKey を参照してください。

での AES キーラッピング AWS CloudHSM

このトピックでは、 の AES キーラッピングのオプションについて説明します AWS CloudHSM。AES キーラップは、AES キー (ラップキー) を使用して、任意のタイプの別のキー (ターゲットキー) をラップします。キーラップを使用して、保存されているキーを保護したり、安全でないネットワーク上でキーを送信したりします。

トピック

サポートされているアルゴリズム

AWS CloudHSM には AES キーラッピング用の 3 つのオプションがあり、各オプションはラップ前にターゲットキーをパディングする方法に基づいています。パディングは、キーラップを呼び出すときに、使用するアルゴリズムに従って自動的に行われます。次の表は、サポートされているアルゴリズムと関連する詳細の一覧で、アプリケーションに適したラップメカニズムを選択するのに役立ちます。

AES キーラップアルゴリズム	の仕様	サポートされているターゲットキーのタイプ	パディングスキーム	AWS CloudHSM クライアントの可用性
ゼロパディングを使用する AES キーラップ	RFC 5649 と SP 800 - 38F	すべて	必要に応じて、キービットの後にゼロを追加してブロック整列する	SDK 3.1 以降
パディングなしの AES キーラップ	RFC 3394 と SP 800 - 38F	AES や 3DES などのブロック整列キー	なし	SDK 3.1 以降
PKCS #5 パディングを使用する AES キーラップ	なし	すべて	ブロック整列するために、PKCS #5 パディングスキー	すべて

AES キーラップアルゴリズム	の仕様	サポートされているターゲットキーのタイプ	パディングスキーム	AWS CloudHSM クライアントの可用性
			ムに従って最低 8 バイトが追加されます	

上記の表の AES キーラップアルゴリズムをアプリケーションで使用する方法については、「AWS CloudHSMでの AES キー ラップの使用」を参照してください。

AES キーラップの初期化ベクトルについて

ラップの前に、CloudHSM はデータの整合性を保つためにターゲットキーに初期化ベクトル (IV) を追加します。各キーラップアルゴリズムには、許可される IV のタイプに特定の制限があります。で IV を設定するには AWS CloudHSM、次の 2 つのオプションがあります。

- 暗黙的: IV を NULL に設定すると、CloudHSM はラップおよびラップ解除オペレーションのためにそのアルゴリズムのデフォルト値を使用します (推奨)
- 明示的: デフォルトの IV 値をキーラップ関数に渡すことによって IV を設定します

> **Important**
>
> アプリケーションで使用している IV を理解する必要があります。キーをラップ解除するには、キーをラップするために使用したのと同じ IV を指定する必要があります。暗黙的な IV を使用してラップする場合は、暗黙的な IV を使用してラップ解除します。暗黙的な IV では、CloudHSM はデフォルト値を使用してラップ解除します。

次の表に、ラップアルゴリズムで指定する IV の許容値を示します。

AES キーラップアルゴリズム	暗黙的な IV	明示的な IV
ゼロパディングを使用する AES キーラップ	必須 デフォルト値: (IV は仕様に基づいて内部で計算されます)	許可されていません

AES キーラップアルゴリズム	暗黙的な IV	明示的な IV
パディングなしの AES キーラップ	許可 (推奨) デフォルト値: `0xA6A6A6A6A6A6A6A6`	許可 この値のみが受け入れられます: `0xA6A6A6A6A6A6A6A6`
PKCS #5 パディングを使用する AES キーラップ	許可 (推奨) デフォルト値: `0xA6A6A6A6A6A6A6A6`	許可 この値のみが受け入れられます: `0xA6A6A6A6A6A6A6A6`

での AES キーラップの使用 AWS CloudHSM

次のようにキーをラップおよびラップ解除します。

- PKCS #11 library] 中で、次の表に示すように `C_WrapKey` および `C_UnWrapKey` の関数のために適切なメカニズムを選択します。
- JCE provider では、次の表に示すように、適切なアルゴリズム、モードとパディングの組み合わせ、暗号メソッド `Cipher.WRAP_MODE` および `Cipher.UNWRAP_MODE` の実装を選択します。
- CloudHSM CLI で、次の表に示すように、サポートされている および アルゴリズムのリストから適切なキーアンラップアルゴリズムを選択します。 キーラップ
- key_mgmt_util (KMU) では、次の表に示すように wrapKey および unWrapKey コマンドを適切な m 値とともに使用します。

AES キーラップアルゴリズム	PKCS #11 メカニズム	Java メソッド	CloudHSM CLI サブコマンド	キー管理ユーティリティ (KMU) 引数
ゼロパディングを使用する AES キーラップ	• `CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD` (ベンダー定義のメカニズム)	`AESWrap/ECB/ZeroPadding`	aes-zero-pad	m = 6

AES キーラップアルゴリズム	PKCS #11 メカニズム	Java メソッド	CloudHSM CLI サブコマンド	キー管理ユーティリティ (KMU) 引数
パディングなしの AES キーラップ	• CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD (ベンダー定義のメカニズム)	AESWrap/ECB/NoPadding	aes-no-pad	m = 5
PKCS #5 パディングを使用する AES キーラップ	• CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD (ベンダー定義のメカニズム)	AESWrap/ECB/PKCS5Padding	aes-pkcs5 パッド	m = 4

での信頼されたキーの使用 AWS CloudHSM

AWS CloudHSM は、内部脅威からデータキーを保護するために、信頼できるキーラッピングをサポートしています。このトピックでは、データを保護する信頼できるキーの作り方を説明します。

トピック

- 信頼できるキーとは
- 信頼できるキーの属性
- 信頼できるキーを使用してデータキーをラップする方法
- データキーを信頼できるキーでアンラップする方法

信頼できるキーとは

信頼できるキーとは、他のキーをラップするために使用されるキーで、管理者や暗号化担当者 (CO) がその属性 CKA_TRUSTED を使用して信頼できるキーであることを明確に識別します。さらに、管理者と暗号責任者 (CO) は、CKA_UNWRAP_TEMPLATE と関連する属性を使用して、信頼できるキーによってデータキーがラップ解除された後に実行できるアクションを指定します。ラップ解除オペレーションを正常に行うには、信頼できるキーによってラップ解除されたデータキーにもこれらの属

性が含まれている必要があります。これにより、ラップされていないデータキーは意図した用途でのみ許可されることが保証されます。

信頼できるキーを用いてラップするすべてのデータキーを識別するために、属性 CKA_WRAP_WITH_TRUSTED を使用します。こうすることで、アプリケーションがそれらをアンラップするのに信頼できるキーのみを使えるように、データキーを制限します。データキーにこの属性を設定すると、その属性は読み取り専用となり、変更することができなくなります。これらの属性を配置すると、アプリケーションは信頼するキーでのみデータキーをアンラップできます。そしてアンラップは、常にこれらのデータキーの使用を制限しようとする属性を持つデータキーをもたらします。

信頼できるキーの属性

以下の属性により、キーを信頼できるキーとしてマークしたり、データキーを信頼できるキーでのみラップおよびラップ解除できるように指定したり、ラップ解除後にデータキーが実行できる操作を制御したりできます。

- CKA_TRUSTED: この属性 (CKA_UNWRAP_TEMPLATE に加えて) をデータキーをラップするキーに適用して、管理者または Crypto Officer (CO) が必要な努力を行っており、このキーを信頼していることを指定します。CKA_TRUSTED を設定できるのは管理者か CO だけです。Crypto User (CU) がそのキーを所有しますが、CKA_TRUSTED 属性を設定できるのは CO のみです。
- CKA_WRAP_WITH_TRUSTED: この属性を エクスポート可能なデータキーに適用して、このキーを CKA_TRUSTED としてマークされたキーでのみラップできるように指定します。いったん CKA_WRAP_WITH_TRUSTED を true に設定すると、属性は読み取り専用となり、属性を変更または削除することはできません。
- CKA_UNWRAP_TEMPLATE: この属性をラッピングキーに適用し (CKA_TRUSTED に加えて)、サービスがアンラップするデータキーにサービスを自動的に適用する必要がある属性名と値を指定します。アプリケーションがラップ解除用のキーを送信するとき、ラップ解除テンプレートを個別に指定できます。アンラップテンプレートを指定し、アプリケーションが独自のアンラップテンプレートを提供する場合、HSM は両方のテンプレートを使用して属性名と値をキーに適用します。ただし、ラッピングキーの CKA_UNWRAP_TEMPLATE の中の値が、アンラップ要求中にアプリケーションによって提供された属性と競合する場合、アンラップ要求は失敗します。

属性の詳細については、次のトピックを参照してください。

- PKCS #11 の主要属性
- JCE キーの属性

- CloudHSM CLI のキー属性

信頼できるキーを使用してデータキーをラップする方法

信頼できるキーを使用してデータキーをラップするには、次の 3 つの基本手順を実行する必要があります。

1. 信頼できるキーでラップする予定のデータキーについては、その CKA_WRAP_WITH_TRUSTED 属性を true に設定します。
2. データキーをラップする予定の信頼できるキーについては、その CKA_TRUSTED 属性を true に設定します。
3. 信頼できるキーを使用してデータキーをラップします。

ステップ 1: データキーの **CKA_WRAP_WITH_TRUSTED** を true に設定する

ラップしたいデータキーについて、以下のオプションのいずれかを選択してキーの CKA_WRAP_WITH_TRUSTED 属性を true に設定します。こうすることで、アプリケーションがそれらをラップするのに信頼できるキーのみを使えるように、データキーを制限します。

オプション 1: 新しいキーを生成する場合は、**CKA_WRAP_WITH_TRUSTED** を true に設定する

PKCS #11、JCE、または CloudHSM CLI を使用してキーを生成します。詳細については、次の例を参照してください。

PKCS #11

PKCS #11 でキーを生成するには、キーの CKA_WRAP_WITH_TRUSTED 属性を true に設定する必要があります。次の例に示すように、これを行うには、この属性をキーの CK_ATTRIBUTE template に含めてから、属性を true に設定します。

```
CK_BYTE_PTR label = "test_key";
CK_ATTRIBUTE template[] = {
        {CKA_WRAP_WITH_TRUSTED, &true_val,         sizeof(CK_BBOOL)},
        {CKA_LABEL,             label,             strlen(label)},
        ...
};
```

詳細については、PKCS #11 によるキーの生成を実演する公開サンプルを参照してください。

JCE

JCE でキーを生成するには、キーの WRAP_WITH_TRUSTED 属性を true に設定する必要があります。次の例に示すように、これを行うには、この属性をキーの KeyAttributesMap に含めてから、属性を true に設定します。

```
final String label = "test_key";
final KeyAttributesMap keySpec = new KeyAttributesMap();
keySpec.put(KeyAttribute.WRAP_WITH_TRUSTED, true);
keySpec.put(KeyAttribute.LABEL, label);
...
```

詳細については、JCE によるキーの生成のデモを行う公開サンプルを参照してください。

CloudHSM CLI

CloudHSM CLI でキーを生成するには、キーの wrap-with-trusted 属性を true に設定する必要があります。これを行うには、キー生成コマンドの適切な引数に wrap-with-trusted=true を含めます。

- 対称キーの場合は、attributes 引数に wrap-with-trusted を追加します。
- パブリックキーの場合は、public-attributes 引数に wrap-with-trusted を追加します。
- プライベートキーの場合は、private-attributes 引数に wrap-with-trusted を追加します。

キーペアの生成の詳細については、「キー generate-asymmetric-pair 」を参照してください。

対称キーの生成の詳細については、「key generate-symmetric 」を参照してください。

オプション 2: 既存のキーを使用する場合は、CloudHSM CLI を使用して **CKA_WRAP_WITH_TRUSTED** を true に設定する

既存のキーの CKA_WRAP_WITH_TRUSTED 属性を true に設定するには、以下の手順に従います。

1. login (ログイン) コマンドを使用して、Crypto User (CU) としてログインします。
2. key set-attribute コマンドを使用してキーの wrap-with-trusted 属性を true に設定します。

```
aws-cloudhsm > key set-attribute --filter attr.label=test_key --name wrap-with-
trusted --value true
```

```
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

ステップ 2: 信頼できるキーの **CKA_TRUSTED** を true に設定する

キーを信頼できるキーにするには、その CKA_TRUSTED 属性を true に設定する必要があります。これを行うには、CloudHSM CLI または CloudHSM 管理ユーティリティ (CMU) を使用できます。

- CloudHSM CLI を使用してキーの CKA_TRUSTED 属性を設定する場合は、「CloudHSM CLI を使用してキーを信頼できるものとしてマークする方法」を参照してください。
- CMU を使用してキーの CKA_TRUSTED 属性を設定する場合は、「CMU でキーを信頼できるものとしてマークする方法」を参照してください。

ステップ 3。信頼できるキーを使用してデータキーをラップする

ステップ 1 で参照したデータキーを、ステップ 2 で設定した信頼できるキーでラップするには、以下のリンクにあるコードサンプルを参照してください。それぞれがキーをラップする方法を示しています。

- AWS CloudHSM PKCS #11 の例
- AWS CloudHSM JCE の例

データキーを信頼できるキーでアンラップする方法

データキーのラップを解除するには、CKA_UNWRAP が true に設定されている信頼できるキーが必要です。このようなキーの条件を満たすには、次の基準も満たしている必要があります。

- キーの CKA_TRUSTED 属性を true に設定する必要があります。
- キーは CKA_UNWRAP_TEMPLATE と関連する属性を使用して、データキーがラップ解除された後に実行できるアクションを指定する必要があります。たとえば、ラップされていないキーを エクスポート不可にしたい場合は、CKA_UNWRAP_TEMPLATE の一部として CKA_EXPORTABLE = FALSE を設定します。

> **Note**
>
> CKA_UNWRAP_TEMPLATE は PKCS #11 でのみ使用できます。

アプリケーションがアンラップするキーを送信するとき、アプリケーションは独自のアンラップテンプレートを提供することもできます。アンラップテンプレートを指定し、アプリケーションが独自のアンラップテンプレートを提供する場合、HSM は両方のテンプレートを使用して属性名と値をキーに適用します。ただし、アンラップリクエスト中に信頼できるキーの CKA_UNWRAP_TEMPLATE の値がアプリケーションによって提供された属性と競合する場合、アンラップリクエストは失敗します。

データキーを信頼できるキーでアンラップする例については、この PKCS #11 の例を参照してください。

CloudHSM CLI によるキーの管理

最新の SDK バージョンシリーズ を使用している場合は、CloudHSM CLI を使用して AWS CloudHSM クラスター内のキーを管理します。詳細については、以下のトピックを参照してください。

- 信頼できるキーの使用では、CloudHSM CLI を使用して信頼できるキーを作成してデータを保護する方法について説明します。
- 「キーの生成」には、対称キー、RSA キー、EC キーなどのキーの作成手順が含まれています。
- 「キーの削除」では、キー所有者がキーを削除する方法について説明しています。
- 「キーの共有と共有解除」では、キー所有者がキーを共有および共有解除する方法について詳しく説明しています。
- 「キーをフィルタリングする」には、フィルタを使用してキーを検索する方法に関するガイドラインが記載されています。

CloudHSM CLI を使用してキーを生成します

キーを生成する前に、CloudHSM CLI を起動し、Crypto User (CU) としてログインする必要があります。HSM でキーを生成するには、生成したいキーと対応するタイプのコマンドを使用します。

トピック

- 対称キーの生成
- 非対称キーの生成

- 関連トピック

対称キーの生成

key generate-symmetric に記載されているコマンドを使用して対称キーを生成します。利用可能なオプションをすべて確認するには、help key generate-symmetric コマンドを使用します。

AES キーの生成

key generate-symmetric aes コマンドを使用して AES キーを生成します。利用可能なオプションをすべて確認するには、help key generate-symmetric aes コマンドを使用します。

Example

次の例では 32 バイトの AES キーを生成します。

```
aws-cloudhsm > key generate-symmetric aes \
    --label aes-example \
    --key-length-bytes 32
```

引数

<LABEL>

AES キーのユーザー定義ラベルを指定します。

必須: はい

<KEY-LENGTH-BYTES>

キーの長さをバイト単位で指定します。

有効値:

- 16、24、32

必須: はい

<KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された AES キーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされている AWS CloudHSM キー属性のリストについては、「」を参照してくださいCloudHSM CLI のキー属性。

必須: いいえ

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、key set-attribute を使用します。

デフォルトでは、生成されるキーは永続/トークンキーです。<SESSION> を使用するとこれは変更され、この引数で生成されたキーがセッション/エフェメラルであることが保証されます

必須: いいえ

汎用シークレットキーを生成します

key generate-symmetric generic-secret コマンドを使用して汎用シークレットキーを生成します。利用可能なオプションをすべて確認するには、help key generate-symmetric generic-secret コマンドを使用します。

Example

次の例では、32 バイトの汎用シークレットキーを生成します。

```
aws-cloudhsm > key generate-symmetric generic-secret \
    --label generic-secret-example \
    --key-length-bytes 32
```

引数

<LABEL>

汎用シークレットキーのユーザー定義ラベルを指定します。

必須: はい

<KEY-LENGTH-BYTES>

キーの長さをバイト単位で指定します。

有効値:

- 1 ~ 800

必須: はい

<KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された汎用シークレットキーに設定するキー属性のスペース区切りリストを指定します (例: token=true)。

サポートされている AWS CloudHSM キー属性のリストについては、「」を参照してくださいCloudHSM CLI のキー属性。

必須: いいえ

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、key set-attribute を使用します。

デフォルトでは、生成されるキーは永続/トークンキーです。<SESSION> を使用するとこれは変更され、この引数で生成されたキーがセッション/エフェメラルであることが保証されます

必須: いいえ

非対称キーの生成

キー generate-asymmetric-pair に記載されているコマンドを使用して非対称キーペアを生成します。

RSA キーの生成

key generate-asymmetric-pair rsa コマンドを使用して RSA キーペアを生成します 利用可能なオプションをすべて確認するには、help key generate-asymmetric-pair rsa コマンドを使用します。

Example

次の例では、RSA 2048 ビットのキーペアが生成されます。

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
    --public-exponent 65537 \
    --modulus-size-bits 2048 \
```

```
    --public-label rsa-public-example \
    --private-label rsa-private-example
```

引数

<PUBLIC_LABEL>

パブリックキーのユーザー定義ラベルを指定します。

必須: はい

<PRIVATE_LABEL>

プライベートキーのユーザー定義ラベルを指定します。

必須: はい

<MODULUS_SIZE_BITS>

モジュラスの長さをビット単位で指定します。最小値は 2048 です。

必須: はい

<PUBLIC_EXPONENT>

パブリック指数を指定します。値は、65537 以上の奇数にする必要があります

必須: はい

<PUBLIC_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された RSA パブリックキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされている AWS CloudHSM キー属性のリストについては、「」を参照してくださいCloudHSM CLI のキー属性。

必須: いいえ

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、key set-attribute を使用します。

デフォルトでは、生成されるキーは永続/トークンキーです。<SESSION> を使用するとこれは変更され、この引数で生成されたキーがセッション/エフェメラルであることが保証されます

必須: いいえ

EC (楕円曲線暗号) キーペアの生成

key generate-asymmetric-pair ec コマンドを使用して EC キーペアを生成します。サポートされている楕円曲線のリストを含め、利用可能なオプションをすべて確認するには、help key generate-asymmetric-pair ec コマンドを使用します。

Example

次の例では、Secp384r1 楕円曲線 を使用して EC キーペアを生成します。

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve secp384r1 \
    --public-label ec-public-example \
    --private-label ec-private-example
```

引数

<PUBLIC_LABEL>

パブリックキーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<PRIVATE_LABEL>

プライベートキーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<CURVE>

楕円曲線の識別子を指定します。

有効値:

- prime256v1
- secp256r1
- secp224r1
- secp384r1
- secp256k1
- secp521r1

必須: はい

<PUBLIC_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された EC パブリックキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされている AWS CloudHSM キー属性のリストについては、「」を参照してくださいCloudHSM CLI のキー属性。

必須: いいえ

<PRIVATE_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された EC プライベートキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされている AWS CloudHSM キー属性のリストについては、「」を参照してくださいCloudHSM CLI のキー属性。

必須: いいえ

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、key set-attribute を使用します。

デフォルトでは、生成されるキーは永続 (トークン) キーです。<SESSION> で渡すことでこれが変わり、この引数で生成されたキーがセッション (エフェメラル) キーであることが保証されます。

必須: いいえ

関連トピック

- CloudHSM CLI のキー属性
- キー generate-asymmetric-pair
- key generate-symmetric

CloudHSM CLI を使用してキーを削除する

このトピックの例を使用して、CloudHSMCLI でキーを削除します。キー所有者のみがキーを削除できます。

トピック

- 例: キーを削除する
- 関連トピック

例: キーを削除する

1. key list コマンドを実行して、削除するキーを特定します。

```
aws-cloudhsm > key list --filter attr.label="my_key_to_delete" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_key_to_delete",
          "id": "",
```

```
          "check-value": "0x29bbd1",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1217,
          "public-exponent": "0x010001",
          "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
```


2. キーを特定したら、キーの固有 `label` 属性を使用して key delete を実行し、キーを削除します。

```
aws-cloudhsm > key delete --filter attr.label="my_key_to_delete"
{
  "error_code": 0,
  "data": {
    "message": "Key deleted successfully"
  }
}
```

3. キーの固有 `label` 属性を使用して key list コマンドを実行し、キーが削除されたことを確認します。次の例に示すように、HSM クラスターには `my_key_to_delete` ラベルの付いたキーはありません。

```
aws-cloudhsm > key list --filter attr.label="my_key_to_delete"
{
  "error_code": 0,
  "data": {
    "matched_keys": [],
    "total_key_count": 0,
    "returned_key_count": 0
  }
}
```

関連トピック

- CloudHSM CLI のキー属性
- キー削除

CloudHSM CLI を使用してキーを共有または共有解除します

このトピックのコマンドを使用して、CloudHSMCLI でキーを共有または共有解除します。では AWS CloudHSM、キーを作成する Crypto User (CU) がキーを所有します。所有者は key share と key unshare コマンドを使用してキーを他の CU と共有または共有解除することができます。キーを共有するユーザーは、暗号化オペレーションでキーを使用することはできますが、そのキーのエクスポート、削除、または他のユーザーとの共有はできません。

キーを共有する前に、キーを所有する crypto user (CU) として HSM にログインする必要があります。

トピック

- 例: キーの共有と共有解除
- 関連トピック

例: キーの共有と共有解除

Example

次の例は、Crypto User (CU) `alice` でキーを共有および共有解除する方法を示しています。key share コマンドと key unshare コマンドに加えて、コマンドの共有と共有解除には CloudHSM CLI キーフィルタを使用する特定のキーと、キーを共有または共有解除するユーザーの特定のユーザー名も必要です。

1. まず、フィルタを指定して key list コマンドを実行して特定のキーを返し、そのキーがすでに誰と共有されているかを確認します。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
```

```
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
"0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
```

```
    "returned_key_count": 1
  }
}
```

2. shared-users 出力を表示して、キーが現在誰と共有されているかを確認します。
3. このキーを Crypto User (CU) alice と共有するには、以下のコマンドを入力します。

```
aws-cloudhsm > key share --filter attr.label="rsa_key_to_share" attr.class=private-
key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key shared successfully"
  }
}
```

このコマンドでは、key share コマンドに加えて、キーの固有ラベルと、キーを共有するユーザーの名前が使用されることに注意してください。

4. key list コマンドを実行して、キーが alice と共有されていることを確認します。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
```

```
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
            {
              "username": "alice",
              "key-coverage": "full"
            }
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
```

```
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

5. alice と同じキーを共有解除するには、以下の unshare コマンドを実行します。

```
aws-cloudhsm > key unshare --filter attr.label="rsa_key_to_share"
 attr.class=private-key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key unshared successfully"
  }
}
```

このコマンドでは、key unshare コマンドに加えて、キーの固有ラベルと、キーを共有するユーザーの名前が使用されることに注意してください。

6. key list コマンドをもう一度実行して、キーが暗号化ユーザー alice と共有解除されたことを確認します。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
```

```
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
```

```
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```


関連トピック

- CloudHSM CLI のキー属性
- キーシェア
- キー共有解除
- CloudHSM CLI を使用してキーをフィルタリングする

CloudHSM CLI を使用してキーをフィルタリングする

以下のキーコマンドを使用して、CloudHSM CLI の標準化されたキーフィルタリングメカニズムを利用します。

- key list
- key delete

- key share
- key unshare
- key set-attribute

CloudHSM CLI でキーを選択またはフィルタリングするには、キーコマンドは CloudHSM CLI のキー属性 に基づく標準化されたフィルタリングメカニズムを利用します。キーまたはキーのセットは、1 つまたは複数のキーを識別できる 1 つ以上の AWS CloudHSM 属性を使用してキーコマンドで指定できます。キーフィルタリングメカニズムは、現在ログインしているユーザーが所有および共有しているキーと、 AWS CloudHSM クラスター内のすべてのパブリックキーに対してのみ動作します。

トピック

- 要件
- 1 つのキーを検索するためのフィルタリング
- フィルタリングエラー
- 関連トピック

要件

キーをフィルタリングするには、Crypto User (CU) としてログインしている必要があります。

1 つのキーを検索するためのフィルタリング

以下の例では、フィルタとして使用する各属性は `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` の形式で書き込む必要があることに注意してください。例えばラベル属性でフィルタリングする場合は、`attr.label=my_label` と書き込みます。

Example 1 つの属性を使用して 1 つのキーを検索する

この例は、1 つの識別属性のみを使用して一意のキー 1 つにフィルタリングする方法を示しています。

```
aws-cloudhsm > key list --filter attr.label="my_unique_key_label" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
```

```
    {
      "key-reference": "0x00000000001c0686",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "alice",
            "key-coverage": "full"
          }
        ],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "my_unique_key_label",
        "id": "",
        "check-value": "0xae8ff0",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": true,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1219,
        "public-exponent": "0x010001",
```

```
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254c8f5
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

Example 複数の属性を使用して 1 つのキーを検索する

次の例では、複数のキー属性を使用して 1 つのキーを検索する方法を示します。

```
aws-cloudhsm > key list --filter attr.key-type=rsa attr.class=private-key attr.check-
value=0x29bbd1 --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            }
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_crypto_user",
          "id": "",
          "check-value": "0x29bbd1",
          "class": "my_test_key",
```

```
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1217,
          "public-exponent": "0x010001",
          "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990c2a7
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

Example フィルタリングしてキーセットを検索する

以下は、プライベート RSA キーのセットを検索するためのフィルタリング方法を示す例です。

```
aws-cloudhsm > key list --filter attr.key-type=rsa attr.class=private-key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
```

```
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
```

```
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254c8f5
          "modulus-size-bits": 2048
        }
      },
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            }
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_test_key",
          "id": "",
          "check-value": "0x29bbd1",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
```

```
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1217,
          "public-exponent": "0x010001",
          "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990c2a7
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 2,
    "returned_key_count": 2
  }
}
```

フィルタリングエラー

特定のキー操作は、一度に 1 つのキーに対してしか実行できません。これらの操作では、フィルタリング基準が十分に調整されておらず、複数のキーが条件に一致する場合、CloudHSM CLI はエラーを表示します。その一例として、キー削除を使用した例を以下に示します。

Example 一致するキーが多すぎるとフィルタリングエラーが発生します

```
aws-cloudhsm > key delete --filter attr.key-type=rsa
{
  "error_code": 1,
  "data": "Key selection criteria matched 48 keys. Refine selection criteria to select
 a single key."
}
```

関連トピック

- CloudHSM CLI のキー属性

CloudHSM CLI を使用してキーを信頼できるものとしてマークする方法

このセクションでは、CloudHSM CLI を使用してキーを信頼できるものとしてマークする方法について説明します。

1. CloudHSM CLI login コマンドを使用して、Crypto User (CU) としてログインします。

2. key list コマンドを使用して、信頼できるとマークしたいキーのキーリファレンスを特定します。次の例では、キーを key_to_be_trusted ラベル付きで表示します。

```
aws-cloudhsm > key list --filter attr.label=test_aes_trusted
        {
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000200333",
        "attributes": {
          "label": "test_aes_trusted"
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

3. ログアウト コマンドを使用して、Crypto User (CU) としてログインします。
4. login (ログイン) コマンドを使用して、管理者としてログインします。
5. key set-attribute コマンドと、ステップ 2 で特定したキーリファレンスを指定して、キーの信頼できる値を「true」に設定します。

```
aws-cloudhsm > key set-attribute --filter key-reference=<Key Reference> --name
 trusted --value true
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

KMU と CMU によるキーの管理

最新の SDK バージョンシリーズ を使用している場合は、CloudHSM CLI を使用して AWS CloudHSM クラスター内のキーを管理します。

以前の SDK バージョンシリーズ を使用している場合は、key_mgmt_util コマンドラインツールを使用して AWS CloudHSM 、クラスター内の HSMs のキーを管理できます。キーを管理する前に、AWS CloudHSM クライアントを起動し、key_mgmt_util を起動して、HSMsにログインする必要があります。さらなる詳細については、Getting Started with key_mgmt_util を参照してください。

- 「Using trusted keys」では、PKCS #11 ライブラリ属性と CMU を使用してデータを保護するための信頼できるキーを作成する方法について説明しています。
- 「キーの生成」には、対称キー、RSA キー、EC キーなどのキーの生成手順が含まれています。
- 「キーのインポート」には、キー所有者がキーをインポートする方法の詳細が記載されています。
- 「キーのエクスポート」には、キー所有者がキーをエクスポートする方法の詳細が記載されています。
- 「キーの削除」では、キー所有者がキーを削除する方法の詳細が記載されています。
- 「キーの共有と共有解除」では、キー所有者がキーを共有および共有解除する方法について詳しく説明しています。

キーの生成

HSM でキーを生成するには、生成したいキーと対応するタイプのコマンドを使用します。

トピック

- 対称キーの生成
- RSA キーペアの生成
- ECC (楕円曲線暗号) キーペアの生成

対称キーの生成

genSymKey コマンドを使用して、AES およびその他のタイプの対称キーを生成します。利用可能なオプションをすべて確認するには、genSymKey -h コマンドを使用します。

以下の例では、256 ビット AESキーが作成されます。

```
Command: genSymKey -t 31 -s 32 -l aes256
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524295

Cluster Error Status
```

```
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

RSA キーペアの生成

RSA キーペアを生成するには、genRSAKeyPair コマンドを使用します。利用可能なオプションをすべて確認するには、genRSAKeyPair -h コマンドを使用します。

次の例では、RSA 2048 ビットのキーペアが生成されます。

```
Command: genRSAKeyPair -m 2048 -e 65537 -l rsa2048
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 524294    private key handle: 524296

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

ECC (楕円曲線暗号) キーペアの生成

ECC キーペアを生成するには、genECCKeyPair コマンドを使用します。サポートされている楕円曲線のリストを含め、利用可能なオプションをすべて確認するには、genECCKeyPair -h コマンドを使用します。

次の例では、NIST FIPS publication 186-4 で定義されている P-384 楕円曲線を使用して ECC キーペアを生成します。

```
Command: genECCKeyPair -i 14 -l ecc-p384
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 524297    private key handle: 524298

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

キーのインポート

シークレットキー (対称キーおよび非対称プライベートキー) を HSM にインポートするには、まず HSM でラップキーを作成する必要があります。ラップキーなしで直接パブリックキーをインポートすることができます。

トピック

- シークレットキーのインポート
- パブリックキーのインポート

シークレットキーのインポート

シークレットキーをインポートするには、次のステップを実行します。シークレットキーをインポートする前に、ファイルに保存します。対称キーを raw バイトとして保存し、非対称プライベートキーを PEM 形式で保存します。

この例では、ファイルからプレーンテキストのシークレットキーを HSM にインポートする方法を示します。暗号化されたキーをファイルから HSM にインポートするには、 unWrapKey コマンドを使用します。

シークレットキーをインポートするには

1. genSymKey コマンドを使用してラッピングキーを作成します。次のコマンドは、現在のセッション中のみ有効な 128 ビット AES ラップキーを作成します。セッションキーまたは永続キーをラップキーとして使用できます。

```
Command: genSymKey -t 31 -s 16 -sess -l import-wrapping-key
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524299

Cluster Error Status
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

2. インポートするシークレットキーのタイプに応じて、次のいずれかのコマンドを使用します。

   - 対称キーをインポートするには、 imSymKey コマンドを使用します。次のコマンドは、前のステップで作成したラップキーを使って aes256.key という名前のファイルから AES キーをインポートします。利用可能なオプションをすべて確認するには、imSymKey -h コマンドを使用します。

```
Command: imSymKey -f aes256.key -t 31 -l aes256-imported -w 524299
Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Unwrapped.  Key Handle: 524300

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

- 非対称プライベートキーをインポートするには、 importPrivateKey コマンドを使用します。次のコマンドは、前のステップで作成したラップキーを使って rsa2048.key という名前のファイルからプライベートキーをインポートします。利用可能なオプションをすべて確認するには、importPrivateKey -h コマンドを使用します。

```
Command: importPrivateKey -f rsa2048.key -l rsa2048-imported -w 524299
BER encoded key length is 1216

Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Private Key Unwrapped.  Key Handle: 524301

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

パブリックキーのインポート

importPubKey コマンドを使用してパブリックキーをインポートします。利用可能なオプションをすべて確認するには、importPubKey -h コマンドを使用します。

次の例では、rsa2048.pub という名前のファイルから RSA パブリックキーをインポートします。

```
Command: importPubKey -f rsa2048.pub -l rsa2048-public-imported
Cfm3CreatePublicKey returned: 0x00 : HSM Return: SUCCESS

Public Key Handle: 524302

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

キーのエクスポート

シークレットキー (対称および非対称プライベートキー) を HSM からエクスポートするためには、まずラップキーを作成する必要があります。ラップキーなしで直接パブリックキーをエクスポートすることができます。

キーをエクスポートできるのは、キー所有者のみです。キーを共有するユーザーは、キーを暗号化オペレーションで使用することはできますが、エクスポートすることはできません。この例を実行する際は、必ず作成したキーをエクスポートします。

> **Important**
>
> exSymKey コマンドは、シークレットキーのプレーンテキスト (暗号化されていない) コピーをファイルに書き込みます。エクスポートプロセスではラップキーが必要ですが、ファイルにあるキーはラップされたキーではありません。キーのラップ (暗号化) されたコピーをエクスポートするには、wrapKey コマンドを使用します。

トピック

- シークレットキーのエクスポート
- パブリックキーのエクスポート

シークレットキーのエクスポート

シークレットキーをエクスポートするには、次のステップを実行します。

シークレットキーをエクスポートするには

1. genSymKey コマンドを使用してラッピングキーを作成します。次のコマンドは、現在のセッション中のみ有効な 128 ビット AES ラップキーを作成します。

```
Command: genSymKey -t 31 -s 16 -sess -l export-wrapping-key
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524304

Cluster Error Status
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

2. エクスポートするシークレットキーのタイプに応じて、次のいずれかのコマンドを使用します。

   - 対称キーをエクスポートするには、 exSymKey コマンドを使用します。次のコマンドでは、aes256.key.exp という名前のファイルに AES キーをエクスポートします。利用可能なオプションをすべて確認するには、exSymKey -h コマンドを使用します。

```
Command: exSymKey -k 524295 -out aes256.key.exp -w 524304
Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "aes256.key.exp"
```

> **Note**
>
> コマンドの出力には、「ラップされた対称キー」が出力ファイルに書かれている、とあります。ただし、出力ファイルにはプレーンテキストの (ラップされていない) キーが含まれています。ファイルにラップ (暗号化) されたキーをエクスポートするには、wrapKey コマンドを使用します。

   - プライベートキーをエクスポートするには、exportPrivateKey コマンドを使用します。次のコマンドでは、rsa2048.key.exp という名前のファイルにプライベートキーをエクスポートします。利用可能なオプションをすべて確認するには、exportPrivateKey -h コマンドを使用します。

```
Command: exportPrivateKey -k 524296 -out rsa2048.key.exp -w 524304
```

```
Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS

PEM formatted private key is written to rsa2048.key.exp
```

パブリックキーのエクスポート

exportPubKey コマンドを使用してパブリックキーをエクスポートします。利用可能なオプションをすべて確認するには、exportPubKey -h コマンドを使用します。

次の例では、rsa2048.pub.exp という名前のファイルに RSA パブリックキーをエクスポートします。

```
Command: exportPubKey -k 524294 -out rsa2048.pub.exp
PEM formatted public key is written to rsa2048.pub.key

Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS
```

キーの削除

次の例のように deleteKey コマンドを使用てキーを削除します。キーを削除できるのは、キー所有者のみです。

```
Command: deleteKey -k 524300
Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

キーの共有と共有解除

では AWS CloudHSM、キーを作成する CU がキーを所有します。所有者はキーを管理し、エクスポートおよび削除し、また、暗号化オペレーションでキーを使用できます。また、所有者は、他の CU ユーザーとキーを共有することもできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用することはできますが、そのキーのエクスポート、削除、または他のユーザーとの共有はできません。

キーの作成時に、 genSymKeyや genRSA コマンドの -uパラメータを使用するなどして、他の CU ユーザーとキーを共有できます。 genRSAKeyPair 別の HSM ユーザーと既存のキーを共有するには、cloudhsm_mgmt_util コマンドラインツールを使用します。これは、このセクションで説明しているほとんどのタスク (key_mgmt_util コマンドラインツールを使用するもの) とは異なります。

キーを共有する前に、cloudhsm_mgmt_util を起動し、暗号化を有効に end-to-endして、HSMsにログインする必要があります。キーを共有するには、キーを所有する crypto user (CU) として HSM にログインします。キーの所有者のみがキーを共有することができます。

shareKey コマンドを使用してキーを共有または共有解除します。キーのハンドルと、ユーザーの ID を指定します。複数のユーザーと共有または共有解除するには、ユーザー ID のカンマ区切りのリストを指定します。キーを共有するには、次の例のように、コマンドの最後のパラメーターとして 1 を使用します。共有解除するには、0 を使用します。

```
aws-cloudhsm>shareKey 524295 4 1
************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
************************************************************

Do you want to continue(y/n)?y
shareKey success on server 0(10.0.2.9)
shareKey success on server 1(10.0.3.11)
shareKey success on server 2(10.0.1.12)
```

shareKey コマンドの構文を次に示します。

```
aws-cloudhsm>shareKey <key handle> <user ID> <Boolean: 1 for share, 0 for unshare>
```

CMU でキーを信頼できるものとしてマークする方法

このセクションでは、CMU を使用してキーを信頼できるものとしてマークする方法について説明します。

1. loginHSM コマンドを使用して、Crypto Officer (CO) としてログインします。
2. OBJ_ATTR_TRUSTED (value 134) を true (1) に設定して setAttribute コマンドを使用します。

   ```
   setAttribute <Key Handle> 134 1
   ```

クローンされたクラスターを管理する

CloudHSM 管理ユーティリティ (CMU) を使用して、リモートリージョンのクラスターが、別のリージョンのクラスターのバックアップから最初に作成された場合、そのクラスターを同期します。たとえば、クラスターを別のリージョン (デスティネーション) にコピーし、後で元のクラスター (ソース) からの変更を同期するとします。このようなシナリオでは、CMU を使用してクラスターを同期します。これを行うには、新しい CMU 構成ファイルを作成し、新しいファイル内の両方のクラスターからハードウェアセキュリティモジュール (HSM) を指定し、CMU を使用してそのファイルを使用してクラスターに接続します。

クローンされたクラスター間で CMU を使用するには

1. 現在の設定ファイルのコピーを作成し、コピーの名前を別の名前に変更します。

   たとえば、次のファイルの場所を使用して、現在の設定ファイルのコピーを検索して作成し、コピーの名前を `cloudhsm_mgmt_config.cfg` から `syncConfig.cfg` に変更します。

   - Linux: `/opt/cloudhsm/etc/cloudhsm_mgmt_config.cfg`
   - Windows: `C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_config.cfg`

2. 名前を変更したコピーで、デスティネーション HSM (同期する必要がある外部リージョンの HSM) の Elastic Network Interface (ENI) IP を追加します。ソース HSM の 下 にデスティネーション HSM を追加することをお勧めします。

```
{
    ...
    "servers": [
        {
            ...
            "hostname": "<ENI Source IP>",
            ...
        },
        {
            ...
            "hostname": "<ENI Destination IP>",
            ...
        }
    ]
}
```

IP アドレスの入手方法については、「the section called “HSM の IP アドレスを取得する”」を参照してください。

3. 新しい設定ファイルで CMU を初期化します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/userSync.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\userSync.cfg
   ```

4. 必要なすべての HSM に CMU が接続されていることを確認するステータスメッセージを確認し、返った ENI IP のうち、どちらが各クラスターに対応するかを判断します。syncUser と syncKey を使用して、ユーザーとキーを手動で同期します。詳細については、「syncUser」そして「syncKey」を参照してください。

HSM の IP アドレスを取得する

HSM の IP アドレスを取得するには、このセクションを使用します。

HSM の IP アドレスを取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. クラスターの詳細ページを開くには、クラスターテーブルでクラスター ID を選択します。
4. IP アドレスを取得するには、[HSM] タブで、[ENI IP アドレス] にリストされている IP アドレスのいずれかを選択します。

HSM の IP アドレスを取得するには (AWS CLI)

- describe-clusters から AWS CLIコマンドを実行して、HSM の IP アドレスを取得します。コマンドの出力では、HSM の IP アドレスは `EniIp` の値です。

  ```
  $  aws cloudhsmv2 describe-clusters
  ```

```
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
...
```

関連トピック

- syncUser
- syncKey
- リージョン間のバックアップのコピー

AWS CloudHSM コマンドラインツール

このトピックでは、管理および AWS CloudHSMを使うために利用できるコマンドラインツールについて説明します。

トピック

コマンドラインツールを理解する

は、AWS リソースの管理に使用する AWS Command Line Interface （ AWS CLI) に加えて、HSM ユーザーとキーを HSM で作成および管理するためのコマンドラインツール AWS CloudHSM を提供します HSMs 。 AWS CloudHSM では、使い慣れた CLI を使用してクラスターを管理し、CloudHSM コマンドラインツールを使用して HSM を管理します。

さまざまなコマンドラインツールが用意されています。

クラスターと HSM を管理するには

の CloudHSMv2 コマンド AWS CLIと AWSPowerShell モジュールの HSM2 PowerShell コマンドレット

- これらのツールは、 AWS CloudHSM クラスターと HSMs。
- CLI の CloudHSMv2 コマンドで コマンドを使用するには、 をインストールして設定 AWS CLI する必要があります。
- AWSPowerShell モジュールの HSM2 PowerShell コマンドレットは、Windows PowerShell モジュールとクロスプラットフォーム PowerShell Core モジュールで使用できます。

HSM ユーザーを管理するには

CloudHSM CLI

- CloudHSM CLI を使用して、ユーザーの作成、ユーザーの削除、ユーザーの一覧作成、ユーザーパスワードの変更、ユーザー多要素認証 (MFA) の更新を行います。AWS CloudHSM クライアントソフトウェアには含まれていません。このツールのインストールに関するガイダンスについては、「Install and configure CloudHSM CLI」を参照してください。

ヘルパーツール

ツールとソフトウェアライブラリの使用には、次の 2 つの AWS CloudHSM ツールが役立ちます。

- configure ツールは、CloudHSM クライアント設定ファイルを更新します。これにより、 AWS CloudHSM はクラスター内の HSMsを同期できます。

  AWS CloudHSM には 2 つのメジャーバージョンがあり、Client SDK 5 が最新です。Client SDK 3 (以前のシリーズ) よりも多くの、さまざまな利点があります。
- pkpspeed は、ソフトウェアライブラリに依存しない HSM ハードウェアのパフォーマンスを測定します。

以前の SDK 向けツール

キー管理ツール (KMU) を使用して、対称キーと非対称キーペアを作成、削除、インポート、エクスポートします。

- key_mgmt_util。このツールは、 AWS CloudHSM クライアントソフトウェアに含まれています。

CloudHSM 管理ツール (CMU) を使用して HSM ユーザーを作成および削除します (ユーザー管理タスクのクォーラム認証の実装を含む)。

- cloudhsm_mgmt_util。このツールは、 AWS CloudHSM クライアントソフトウェアに含まれています。

設定ツール

AWS CloudHSM は、クラスター内のすべてのハードウェアセキュリティモジュール (HSM) 間でデータを自動的に同期します。configure ツールは、同期メカニズムが使用する設定ファイルの HSM データを更新します。configure を使用して、コマンドラインツールを使用する前に、特に、クラスター内の HSM が変更されたときに、HSM データを更新します。

AWS CloudHSM には、次の 2 つの主要な Client SDK バージョンが含まれています。

- Client SDK 5: これは最新かつデフォルトの Client SDK です。Client SDK 5 による利点については、「Client SDK 5 の利点」を参照してください。
- Client SDK 3: これは古い Client SDK です。プラットフォームおよび言語ベースのアプリケーションの互換性および管理ツール用のコンポーネントの完全なセットが含まれています。

Client SDK 3 から Client SDK 5 に移行する手順については、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。

トピック

- Client SDK 5 設定ツール
- Client SDK 3 設定ツール

Client SDK 5 設定ツール

クライアント SDK 5 設定ツールを使用して、クライアント側の構成ファイルを更新します。

クライアント SDK 5 の各コンポーネントには、構成ツールのファイル名にコンポーネントの指定子を含む構成ツールが含まれています。たとえば、クライアント SDK 5 の PKCS #11 ライブラリには、Linux上 `configure-pkcs11` またはWindows上 `configure-pkcs11.exe` で名付けられた構成ツールが含まれています。

構文

PKCS #11

```
configure-pkcs11[ .exe ]
              -a <ENI IP address>
              [--hsm-ca-cert <customerCA certificate file path>]
              [--cluster-id <cluster ID>]
```

```
             [--endpoint <endpoint>]
             [--region <region>]
             [--server-client-cert-file <client certificate file path>]
             [--server-client-key-file <client key file path>]
             [--log-level <error | warn | info | debug | trace>]
                  Default is <info>
             [--log-rotation <daily | weekly>]
                  Default is <daily>
             [--log-file <file name with path>]
                  Default is </opt/cloudhsm/run/cloudhsm-pkcs11.log>
                  Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-pkcs11.log>
             [--log-type <file | term>]
                  Default is <file>
             [-h | --help]
             [-V | --version]
             [--disable-key-availability-check]
             [--enable-key-availability-check]
             [--disable-validate-key-at-init]
             [--enable-validate-key-at-init]
                  This is the default for PKCS #11
```

OpenSSL

```
configure-dyn[ .exe ]
             -a <ENI IP address>
             [--hsm-ca-cert <customerCA certificate file path>]
             [--cluster-id <cluster ID>]
             [--endpoint <endpoint>]
             [--region <region>]
             [--server-client-cert-file <client certificate file path>]
             [--server-client-key-file <client key file path>]
             [--log-level <error | warn | info | debug | trace>]
                  Default is <error>
             [--log-type <file | term>]
                  Default is <term>
             [-h | --help]
             [-V | --version]
             [--disable-key-availability-check]
             [--enable-key-availability-check]
             [--disable-validate-key-at-init]
                  This is the default for OpenSSL
             [--enable-validate-key-at-init]
```

JCE

```
configure-jce[ .exe ]
              -a <ENI IP address>
              [--hsm-ca-cert <customerCA certificate file path>]
              [--cluster-id <cluster ID>]
              [--endpoint <endpoint>]
              [--region <region>]
              [--server-client-cert-file <client certificate file path>]
              [--server-client-key-file <client key file path>]
              [--log-level <error | warn | info | debug | trace>]
                   Default is <info>
              [--log-rotation <daily | weekly>]
                   Default is <daily>
              [--log-file <file name with path>]
                   Default is </opt/cloudhsm/run/cloudhsm-jce.log>
                   Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-jce.log>
              [--log-type <file | term>]
                   Default is <file>
              [-h | --help]
              [-V | --version]
              [--disable-key-availability-check]
              [--enable-key-availability-check]
              [--disable-validate-key-at-init]
                   This is the default for JCE
              [--enable-validate-key-at-init]
```

CloudHSM CLI

```
configure-cli[ .exe ]
              -a <ENI IP address>
              [--hsm-ca-cert <customerCA certificate file path>]
              [--cluster-id <cluster ID>]
              [--endpoint <endpoint>]
              [--region <region>]
              [--server-client-cert-file <client certificate file path>]
              [--server-client-key-file <client key file path>]
              [--log-level <error | warn | info | debug | trace>]
                   Default is <info>
              [--log-rotation <daily | weekly>]
                   Default is <daily>
              [--log-file <file name with path>]
```

```
                Default for Linux is </opt/cloudhsm/run/cloudhsm-cli.log>
                Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-cli.log>
            [--log-type <file | term>]
                Default setting is <file>
            [-h | --help]
            [-V | --version]
            [--disable-key-availability-check]
            [--enable-key-availability-check]
            [--disable-validate-key-at-init]
                This is the default for CloudHSM CLI
            [--enable-validate-key-at-init]
```

詳細設定

Client SDK 5 構成ツール固有の詳細設定のリストについては、「Advanced configurations for the Client SDK 5 configure tool」を参照してください。

> **Important**
>
> 設定を変更した後、変更内容を反映させるためにアプリケーションを再起動する必要があります。

例

これらの例は、Client SDK 5 の構成ツールの使用方法を示しています。

クライアント SDK 5 のブートストラップ

Example

この例では、Client SDK 5 の HSM データを更新するための -a パラメータを使用しています。-a パラメータの場合は、クラスターのいずれかの HSM の IP アドレスが必要です。

PKCS #11 library

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 -a <HSM IP addresses>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" -a <HSM IP
 addresses>
```

OpenSSL Dynamic Engine

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn -a <HSM IP addresses>
```

JCE provider

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce -a <HSM IP addresses>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" -a <HSM IP addresses>
```

CloudHSM CLI

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IP addresses of the HSMs>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI IP
 addresses of the HSMs>
```

> **Note**
>
> `--cluster-id` パラメータは `-a <HSM_IP_ADDRESSES>` の代わりに使用できます。`--cluster-id` の使用要件については、「Client SDK 5 設定ツール」を参照してください。

`-a` パラメータの詳細については、「the section called “パラメータ”」をご参照ください。

Client SDK 5 のクラスター、リージョン、エンドポイントの指定

Example

この例では、`cluster-id` パラメータを使用して、`DescribeClusters` 呼び出しを行うことにより、Client SDK 5 をブートストラップします。

PKCS #11 library

Client SDK 5 の Linux EC2 インスタンスを **cluster-id** 呼び出しでブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id cluster-1234567
```

Client SDK 5 用の Windows EC2 インスタンスを **cluster-id** でブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe" --cluster-
id cluster-1234567
```

OpenSSL Dynamic Engine

Client SDK 5 の Linux EC2 インスタンスを **cluster-id** 呼び出しでブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --cluster-id cluster-1234567
```

JCE provider

Client SDK 5 の Linux EC2 インスタンスを **cluster-id** 呼び出しでブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce --cluster-id cluster-1234567
```

Client SDK 5 用の Windows EC2 インスタンスを **cluster-id** でブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-jce.exe" --cluster-
id cluster-1234567
```

CloudHSM CLI

Client SDK 5 の Linux EC2 インスタンスを **cluster-id** 呼び出しでブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli --cluster-id cluster-1234567
```

Client SDK 5 用の Windows EC2 インスタンスを **cluster-id** でブートストラップするには

- クラスター ID `cluster-1234567`を使用して、クラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --cluster-
id cluster-1234567
```

`--region` と `--endpoint` のパラメータと `cluster-id` のパラメータを組み合わせて、システムが `DescribeClusters` の呼び出しを行う方法を指定することができます。例えば、クラスターのリージョンが AWS CLI のデフォルトとして設定されているものと異なる場合、そのリージョンを使用するように `--region` パラメータを使用する必要があります。さらに、呼び出しに使用する AWS CloudHSM API エンドポイントを指定することもできます。これは、 のデフォルトの DNS ホスト名を使用しない VPC インターフェイスエンドポイントを使用するなど、さまざまなネットワーク設定で必要になる場合があります AWS CloudHSM。

PKCS #11 library

カスタムエンドポイントとリージョンを使用して Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id cluster-1234567 --
region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

カスタムエンドポイントとリージョンを使用して Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe --cluster-
id cluster-1234567--region us-east-1 --endpoint https://cloudhsmv2.us-
east-1.amazonaws.com
```

OpenSSL Dynamic Engine

カスタムエンドポイントとリージョンを使用して Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

JCE provider

カスタムエンドポイントとリージョンを使用して Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

カスタムエンドポイントとリージョンを使用して Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-jce.exe" --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

CloudHSM CLI

カスタムエンドポイントとリージョンを使用して Linux EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

カスタムエンドポイントとリージョンを使用して Windows EC2 インスタンスをブートストラップするには

- 設定ツールを使用して、カスタムリージョンとエンドポイントを持つクラスター内の HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-cli.exe" --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

`--cluster-id`、`--region`、`--endpoint` パラメータの詳細については、the section called “パラメータ”を参照してください。

TLS クライアントサーバー相互認証のためのクライアント証明書とキーの更新

Example

この例では、 のカスタムキー`server-client-cert-file`と SSL 証明書を指定して、 および `--server-client-key-file`パラメータを使用して SSL を再設定する方法を示します。 AWS CloudHSM

PKCS #11 library

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" `
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

OpenSSL Dynamic Engine

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

JCE provider

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" `
```

```
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

CloudHSM CLI

Linux のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
$ sudo cp ssl-client.crt /opt/cloudhsm/etc
sudo cp ssl-client.key /opt/cloudhsm/etc
```

2. 構成ツールで ssl-client.crt、ssl-client.key を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli \
            --server-client-cert-file /opt/cloudhsm/etc/ssl-client.crt \
            --server-client-key-file /opt/cloudhsm/etc/ssl-client.key
```

Windows のクライアント SDK 5 で TLS クライアントサーバーの相互認証にカスタム証明書とキーを使用するには

1. キーと証明書を適切なディレクトリにコピーします。

```
cp ssl-client.crt C:\ProgramData\Amazon\CloudHSM\ssl-client.crt
cp ssl-client.key C:\ProgramData\Amazon\CloudHSM\ssl-client.key
```

2. PowerShell インタープリタでは、設定ツールを使用して ssl-client.crtと を指定しますssl-client.key。

```
& "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" `
            --server-client-cert-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.crt `
            --server-client-key-file C:\ProgramData\Amazon\CloudHSM\ssl-
client.key
```

server-client-cert-file、および --server-client-key-file、パラメータの詳細については、the section called “パラメータ” を参照してください。

クライアントキーの耐久性設定を無効にする

Example

この例では --disable-key-availability-check パラメータを使用して、クライアントキーの耐久性設定を無効にします。単一の HSM でクラスターを実行するには、クライアントキーの耐久性設定を無効にする必要があります。

PKCS #11 library

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --disable-key-
availability-check
```

OpenSSL Dynamic Engine

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

JCE provider

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --disable-key-
availability-check
```

CloudHSM CLI

Linux でクライアント SDK 5 のクライアントキーの耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
$ sudo /opt/cloudhsm/bin/configure-cli --disable-key-availability-check
```

Windows でクライアント SDK 5 のクライアントキー耐久性を無効にするには

- 構成ツールを使用して、クライアントキーの耐久性設定を無効にします。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --disable-key-
availability-check
```

`--disable-key-availability-check` パラメータの詳細については、「the section called “パラメータ”」をご参照ください。

ログ記録オプションの管理

Example

Client SDK 5 では、`log-file`、`log-level`、`log-rotation`、および `log-type` パラメータを使用して、ログを管理します。

> **Note**
>
> AWS Fargate や AWS Lambda などのサーバーレス環境に SDK を設定するには、 AWS CloudHSM ログタイプを に設定することをお勧めしますterm。クライアントログは に出力stderrされ、その環境に設定された CloudWatch ロググループにキャプチャされます。

PKCS #11 library

デフォルトのログ記録の場所

- ファイルの場所を指定しない場合、システムはログを以下のデフォルトの場所に書き込みます。

  Linux

  ```
  /opt/cloudhsm/run/cloudhsm-pkcs11.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-pkcs11.log
  ```

ログ記録レベルを設定し、他のログ記録オプションはデフォルトのままにしておくには

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level info
  ```
  ```

ファイルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-type file --log-file <file name with path> --log-rotation daily --log-level info
  ```

ターミナルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-type term --log-level info
  ```

OpenSSL Dynamic Engine

デフォルトのログ記録の場所

- ファイルの場所を指定しない場合、システムはログを以下のデフォルトの場所に書き込みます。

  Linux

  ```
  stderr
  ```

ログ記録レベルを設定し、他のログ記録オプションはデフォルトのままにしておくには

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-level info
  ```

ファイルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-type <file name> --log-file file --log-rotation daily --log-level info
  ```

ターミナルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-type term --log-level info
  ```
  ```

JCE provider

デフォルトのログ記録の場所

- ファイルの場所を指定しない場合、システムはログを以下のデフォルトの場所に書き込みます。

  Linux

  ```
  /opt/cloudhsm/run/cloudhsm-jce.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-jce.log
  ```

ログ記録レベルを設定し、他のログ記録オプションはデフォルトのままにしておくには

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-level info
  ```

ファイルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-type file --log-file <file name> --
  log-rotation daily --log-level info
  ```

ターミナルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-type term --log-level info
  ```

CloudHSM CLI

デフォルトのログ記録の場所

- ファイルの場所を指定しない場合、システムはログを以下のデフォルトの場所に書き込みます。

  Linux
  ```

```
/opt/cloudhsm/run/cloudhsm-cli.log
```

Windows

```
C:\Program Files\Amazon\CloudHSM\cloudhsm-cli.log
```

ログ記録レベルを設定し、他のログ記録オプションはデフォルトのままにしておくには

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-level info
  ```

ファイルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-type file --log-file <file name> --
  log-rotation daily --log-level info
  ```

ターミナルのログ記録オプションを設定するには

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-type term --log-level info
  ```

`log-file`、`log-level`、`log-rotation`、`log-type` のパラメータの詳細については、「the section called “パラメータ”」を参照してください。

Client SDK 5 の発行証明書を配置する

Example

この例では `--hsm-ca-cert` パラメータを使用して、クライアント SDK 5 の発行証明書の場所を更新します。

PKCS #11 library

Linux クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。
  ```

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe" --hsm-ca-
cert <customerCA certificate file>
```

OpenSSL Dynamic Engine

Linux クライアント SDK 5 の発行証明書を配置する

- 構成ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --hsm-ca-cert <customerCA certificate
 file>
```

JCE provider

Linux クライアント SDK 5 の発行証明書を配置する

- 設定ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-jce --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-jce.exe" --hsm-ca-cert <customerCA
 certificate file>
```

CloudHSM CLI

Linux クライアント SDK 5 の発行証明書を配置する

- 設定ツールを使用して、発行証明書の場所を指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli --hsm-ca-cert <customerCA certificate
 file>
```

Windows クライアント SDK 5 の発行証明書を配置します。

- 設定ツールを使用して、発行証明書の場所を指定します。

```
"C:\Program Files\Amazon\CloudHSM\configure-cli.exe" --hsm-ca-cert <customerCA
 certificate file>
```

`--hsm-ca-cert` パラメータの詳細については、「the section called “パラメータ”」をご参照ください。

パラメータ

-a ***<ENI IP address>***

指定した IP アドレスをクライアント SDK 5 設定ファイルに追加します。クラスターから HSM の ENI IP アドレスを入力します。このオプションの使用方法の詳細については、「クライアント SDK 5でブートストラップ」 を参照してください。

必須：はい

--hsm-ca-cert ***<customerCA ##########***

EC2クライアントインスタンスをクラスターに接続するために使用する認証局（CA）証明書を格納するディレクトリへのパス。このファイルは、クラスターを初期化するときに作成します。デフォルトでは、システムはこのファイルを次の場所で検索します。

Linux

```
/opt/cloudhsm/etc/customerCA.crt
```

Windows

```
C:\ProgramData\Amazon\CloudHSM\customerCA.crt
```

クラスターの初期化または証明書の配置の詳細については、「???」 および 「???」 を参照してください。

必須：いいえ

--cluster-id ***<cluster ID>***

`DescribeClusters` を呼び出して、クラスターIDに関連付けられたクラスターのすべての HSM Elastic Network Interface（ENI）IPアドレスを検索します。システムは ENI IP アドレスを設定 AWS CloudHSM ファイルに追加します。

> **Note**
>
> パブリックインターネットにアクセスできない VPC 内の EC2 インスタンスから `--cluster-id`パラメータを使用する場合は、インターフェイス VPC エンドポイントを作成して に接続する必要があります AWS CloudHSM。VPCエンドポイントの詳細については、??? を参照してください。

必須：いいえ

--endpoint ***<endpoint>***

`DescribeClusters` 呼び出しに使用する AWS CloudHSM API エンドポイントを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

必須：いいえ

--region ***<region>***

クラスターのリージョンを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

この `--region` パラメータを指定しない場合、システムは `AWS_DEFAULT_REGION` または `AWS_REGION` の環境変数の読み取りを試みてリージョンを選択します。これらの変数が設定されていない場合、環境変数で別のファイルを指定しない限り、AWS Config (通常は `~/.aws/config`) のプロファイルに関連付けられたリージョンをチェックします`AWS_CONFIG_FILE`。いずれも設定されていない場合は、`us-east-1` デフォルトでリージョンが設定されます。

必須：いいえ

--server-client-cert-file ***##################***

TLS クライアント・サーバー相互認証に使用するクライアント証明書へのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは `--server-client-key-file` と組み合わせて設定する必要があります。

必須：いいえ

--server-client-key-file ***##################***

TLS クライアントとサーバーの相互認証に使用されるクライアントキーへのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは `--server-client-cert-file` と組み合わせて設定する必要があります。

必須：いいえ

--log-level ***<error | warn | info | debug | trace>***

システムがログファイルに書き込むべき最小のログレベルを指定します。各レベルは前のレベルを含み、最小レベルはエラー、最大レベルはトレースとなります。つまり、エラーを指定する

と、システムはログにエラーのみを書き込みます。トレースを指定すると、システムはエラー、警告、情報 (info)、およびデバッグメッセージをログに書き込みます。詳細については、「クライアント SDK 5 のログの記録」を参照してください。

必須：いいえ

--log-rotation ***<daily | weekly>***

システムがログをローテートする頻度を指定します。詳細については、「クライアント SDK 5 のログの記録」を参照してください。

必須：いいえ

--log-file ***<file name with path>***

システムがログファイルを書き込む場所を指定します。詳細については、「クライアント SDK 5 のログの記録」を参照してください。

必須：いいえ

--log-type ***<term | file>***

システムがログをファイルまたはターミナルのどちらに書き込むかを指定します。詳細については、「クライアント SDK 5 のログの記録」を参照してください。

必須：いいえ

-h | --help

ヘルプを表示します。

必須：いいえ

-v | --version

バージョンを表示します。

必須：いいえ

--disable-key-availability-check

キーの可用性クォーラムを無効にするためのフラグ。このフラグを使用して、 AWS CloudHSM がキー可用性クォーラムを無効にし、クラスター内の 1 つの HSM にのみ存在するキーを使用できることを示します。このフラグを使用してキーの可用性クォーラムを設定する方法については、「???」を参照してください。

必須：いいえ

--enable-key-availability-check

キーの可用性クォーラムを有効にするためのフラグ。このフラグを使用して、 AWS CloudHSM がキー可用性クォーラムを使用し、それらのキーがクラスター内の 2 つの HSMs に存在するまでキーを使用できないことを示します。このフラグを使用してキーの可用性クォーラムを設定する方法については、「???」を参照してください。

デフォルトでは有効になっています。

必須：いいえ

-disable-validate-key-at--init

このフラグを指定すると、その後の呼び出しでキーのパーミッションを確認するための初期化呼び出しをスキップできるため、パフォーマンスが向上します。注意して使用してください。

背景: PKCS #11 ライブラリの一部のメカニズムでは、初期化コールで後続のコールでキーを使用できるかどうかを検証するマルチパートオペレーションをサポートしています。これには HSM への検証呼び出しが必要で、オペレーション全体にレイテンシーが追加されます。このオプションを使用すると、後続の呼び出しを無効にし、パフォーマンスを向上させる可能性があります。

必須：いいえ

-enable-validate-key-at--init

初期化呼び出しを使用して、後続の呼び出しでキーに対する許可を検証するように指定します。これがデフォルトのオプションです。`enable-validate-key-at-init` を使用して、これらの初期化呼び出しを再開するには `disable-validate-key-at-init` を一時停止します。

必須：いいえ

関連トピック

- DescribeClusters API オペレーション
- describe-clusters AWS CLI
- Get-HSM2Cluster PowerShell コマンドレット
- クライアント SDK 5 のブートストラップ
- AWS CloudHSM VPC エンドポイント
- クライアント SDK 5 キーの耐久性設定の管理
- クライアント SDK 5 ログ記録

Client SDK 5 設定ツールの詳細設定

Client SDK 5 設定ツールには、ほとんどのユーザーが使用する一般的な設定には含まれない詳細設定があります。詳細設定には追加機能があります。

- PKCS #11 の詳細設定
  - PKCS #11 を使用した複数のスロットへの接続
  - PKCS #11 の再試行コマンド
- JCE の詳細設定
  - JCE プロバイダーによる複数のクラスターへの接続
  - JCE の再試行コマンド
  - JCE を使用したキー抽出
- OpenSSL の詳細設定
  - OpenSSL の再試行コマンド
- AWS CloudHSM コマンドラインインターフェイス (CLI) の詳細設定
  - CloudHSM CLI を使用した複数のクラスターへの接続

Client SDK 3 設定ツール

クライアント SDK 3 設定ツールを使用して、クライアントデーモンをブートストラップし、CloudHSM 管理ユーティリティを構成します。

構文

```
configure -h | --help
          -a <ENI IP address>
          -m [-i <daemon_id>]
          --ssl --pkey <private key file> --cert <certificate file>
          --cmu <ENI IP address>
```

例

以下の例では、configure ツールの使用方法を示します。

Example : AWS CloudHSM クライアントと key_mgmt_util の HSM データを更新する

この例では、 の -aパラメータを使用してconfigure、 AWS CloudHSM クライアントと key_mgmt_util の HSM データを更新します。-a パラメータの場合は、クラスターのいずれかの HSM の IP アドレスが必要です。IP アドレスを取得するには、コンソールまたは AWS CLI を使用します。

HSM の IP アドレスを取得するには (コンソール)

1. https://console.aws.amazon.com/cloudhsm/home で AWS CloudHSM コンソールを開きます。
2. AWS リージョンを変更するには、ページの右上隅にあるリージョンセレクターを使用します。
3. クラスターの詳細ページを開くには、クラスターテーブルでクラスター ID を選択します。
4. IP アドレスを取得するには、[HSM] タブで、[ENI IP アドレス] にリストされている IP アドレスのいずれかを選択します。

HSM の IP アドレスを取得するには (AWS CLI)

- describe-clusters から AWS CLIコマンドを実行して、HSM の IP アドレスを取得します。コマンドの出力では、HSM の IP アドレスは `EniIp` の値です。

```
$  aws cloudhsmv2 describe-clusters


{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
...
```

: HSM のデータを更新するには

1. -a パラメータを更新する前に、 AWS CloudHSM クライアントを停止します。これにより、configure がクライアントの設定ファイルを編集する間に発生する可能性がある競合を防ぎます。クライアントがすでに停止している場合は、このコマンドによる影響はないので、スクリプトで使用できます。

   Amazon Linux

   ```
   $ sudo stop cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 8

   ```
   $ sudo service cloudhsm-client stop
   ```

   RHEL 7

   ```
   $ sudo service cloudhsm-client stop
   ```

   RHEL 8

   ```
   $ sudo service cloudhsm-client stop
   ```

   Ubuntu 16.04 LTS

   ```
   $ sudo service cloudhsm-client stop
   ```

   Ubuntu 18.04 LTS

   ```
   $ sudo service cloudhsm-client stop
   ```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

  AWS CloudHSM クライアントを起動したコマンドウィンドウで Ctrl +C を使用します。

2. このステップでは、configure -aconfigure の -a パラメータを使用して 10.0.0.9 ENI IP アドレスを設定ファイルに追加します。

Amazon Linux

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Amazon Linux 2

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 16.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 18.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\ configure.exe -a 10.0.0.9
```

3. 次に、 AWS CloudHSM クライアントを再起動します。起動した クライアントでは、設定ファイルの ENI IP アドレスを使用してクラスターにクエリを実行します。次に、クラスター内のすべての HSM の ENI IP アドレスを、`cluster.info` ファイルに書き込みます。

   Amazon Linux

   ```
   $ sudo start cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 8

   ```
   $ sudo service cloudhsm-client start
   ```

   RHEL 7

   ```
   $ sudo service cloudhsm-client start
   ```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

コマンドが完了すると、 AWS CloudHSM クライアントと key_mgmt_util が使用する HSM データは完全かつ正確になります。

Example : Client SDK 3.2.1 以前から CMU の HSM データを更新

この例では、-mconfigure コマンドを使用して、更新された HSM データを `cluster.info` ファイルから cloudhsm_mgmt_util が使用する `cloudhsm_mgmt_util.cfg` ファイルにコピーしています。クライアント SDK 3.2.1 以前と同梱されている CMU でこれを使用します。

- を実行する前に-m、前の例 に示すように、 AWS CloudHSM クライアントを停止し、-a コマンドを実行してから AWS CloudHSM クライアントを再起動します。これにより、`cluster.info` ファイルから `cloudhsm_mgmt_util.cfg` ファイルにコピーされたデータが完全で正確であることを確認できます。

Linux

```
$ sudo /opt/cloudhsm/bin/configure -m
```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\ configure.exe -m
```

Example : Client SDK 3.3.0 以降に同梱されている CMU の HSM データを更新

この例では、--cmu パラメータの configure コマンドにして CMU の HSM データを更新しています。クライアント SDK 3.3.0 以降と同梱されている CMU で使用します。CMU の使用方法の詳細については、「CloudHSM 管理ユーティリティ (CMU) を使用したユーザーの管理」そして「Client SDK 3.2.1 以前での CMU の使用」を参照してください。

- --cmu パラメータを使用して、クラスター内の HSM の IP アドレスを渡します。

  Linux

  ```
  $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
  ```

パラメータ

-h | --help

コマンド構文を表示します。

必須：はい

-a *<ENI IP address>*

指定した HSM の Elastic Network Interface (ENI) IP アドレスを AWS CloudHSM 設定ファイルに追加します。クラスターのいずれかの HSM の ENI IP アドレスを入力します。どれを選択してもかまいません。

クラスター内の HSMs の ENI IP アドレスを取得するには、 DescribeClustersオペレーション、describe-clusters AWS CLI コマンド、または Get-HSM2Cluster PowerShell コマンドレットを使用します。

> **Note**
>
> -a configure コマンドを実行する前に、 AWS CloudHSM クライアントを停止します。その後、-aコマンドが完了したら、 AWS CloudHSM クライアントを再起動します。詳細については、例を参照してください。

このパラメータは、次の設定ファイルを編集します。

- `/opt/cloudhsm/etc/cloudhsm_client.cfg`: client AWS CloudHSM および key_mgmt_util によって使用されます。
- `/opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg`: 使用者 cloudhsm_mgmt_util。

AWS CloudHSM クライアントは起動時に、設定ファイル内の ENI IP アドレスを使用してクラスターをクエリし、クラスター内のすべての HSMs の正しい ENI IP アドレスで`cluster.info`ファイル (`/opt/cloudhsm/daemon/1/cluster.info`) を更新します。

必須：はい

-m

CMU が使用する構成ファイルで HSM ENI IP アドレスを更新します。

> **Note**
>
> -m パラメータは、Client SDK 3.2.1 以前の CMU で使用するためのものです。クライアント SDK 3.3.0 以降の CMU については、CMU の HSM データ更新プロセスを簡略化する「`--cmu`パラメータ」を参照してください。

の -aパラメータを更新configureして AWS CloudHSM クライアントを起動すると、クライアントデーモンはクラスターをクエリし、クラスター内のすべての HSM の正しい HSMs IP アドレスで`cluster.info`ファイルを更新します。-mconfigure コマンドを実行すると、Cloudhsm_mgmt_util が使用する `cluster.info` 構成ファイルから `cloudhsm_mgmt_util.cfg` 構成ファイルに HSM IP アドレスがコピーされ、更新が完了します。

-a configure コマンドを実行する前に、必ず -m コマンドを実行し、 AWS CloudHSM クライアントを再起動してください。これにより、cluster.info から cloudhsm_mgmt_util.cfg にコピーされたデータが完全で正確であることを確認できます。

必須：はい

-i

代替クライアントデーモンを指定します。デフォルト値は AWS CloudHSM クライアントを表します。

デフォルト: 1

必須：いいえ

--ssl

クラスターの SSL キーおよび証明書を指定するプライベートキーおよび証明書に置き換えます。このパラメータを使用する場合には、--pkey および --cert パラメータが必要となります。

必須：いいえ

--pkey

新しいプライベートキーを指定します。プライベートキーが含まれているファイルのファイル名を入力します。

必須: はい (--ssl が指定されている場合) それ以外の場合は、使用しないでください。

--cert

新しい証明書を指定します。証明書が含まれているファイルのファイル名を入力します。証明書は、クラスターを初期化するために使用される自己署名証明書である customerCA.crt 証明書に連鎖する必要があります。詳細については、「クラスターの初期化」を参照してください。

必須: はい (--ssl が指定されている場合) それ以外の場合は、使用しないでください。

--cmu ***<ENI IP address>***

-a と -m のパラメータを 1 つのパラメータにまとめます。指定された HSM Elastic Network Interface (ENI) IP アドレスを設定 AWS CloudHSM ファイルに追加し、CMU 設定ファイルを更新します。クラスター内の任意の HSM から IP アドレスを入力します。クライアント SDK 3.2.1 以前については、「クライアント SDK 3.2.1 以前での CMU の使用」 を参照してください。

必須: はい

関連トピック

- key_mgmt_util のセットアップ

CloudHSM コマンドラインインターフェイス (CLI)

CloudHSM CLI は、管理者がユーザーを管理し、暗号ユーザーがクラスター内のキーを管理するのに役立ちます。これには、ユーザーの作成、削除、一覧表示、ユーザーパスワードの変更、ユーザー多要素認証 (MFA) の更新に使用できるツールが含まれています。また、キーの生成、削除、インポート、エクスポート、属性の取得と設定、キーの検索、暗号化オペレーションの実行を行うコマンドも含まれています。

CloudHSM CLI ユーザーの定義済みリストについては、「CloudHSM CLI による HSM ユーザーの管理 」を参照してください。CloudHSM CLI のキー属性の定義済みリストについては、「」を参照してくださいCloudHSM CLI のキー属性。CloudHSM CLI を使用してキーを管理する方法については、「」を参照してくださいCloudHSM CLI によるキーの管理。

クイックスタートについては、「CloudHSM コマンドラインインターフェイス (CLI) の使用開始」を参照してください。CloudHSM CLI コマンドとコマンドの使用例の詳細については、「CloudHSM CLI コマンドのリファレンス」を参照してください。

トピック

- CloudHSMのコマンドラインインターフェイス (CLI)がサポートされているプラットフォーム
- CloudHSM コマンドラインインターフェイス (CLI) の使用開始
- インタラクティブモードおよびシングルコマンドモード
- CloudHSM CLI のキー属性
- Client SDK 3 CMU および KMU から Client SDK 5 CloudHSM CLI に移行する
- CLI の詳細設定
- CloudHSM CLI コマンドのリファレンス

CloudHSMのコマンドラインインターフェイス (CLI)がサポートされているプラットフォーム

Linux サポート

サポートされているプラットフォーム	x86_64 アーキテクチャ	ARM アーキテクチャ
Amazon Linux 2	はい	あり
Amazon Linux 2023	はい	あり
CentOS 7 (7.8 以降)	あり	なし
Red Hat Enterprise Linux 7 (7.8 以降)	あり	なし
Red Hat Enterprise Linux 8 (8.3 以降)	あり	なし
Red Hat Enterprise Linux 9 (9.2 以降)	はい	あり
Ubuntu 20.04 LTS	はい	いいえ
Ubuntu 22.04 LTS	はい	あり

注: SDK 5.4.2 は、CentOS 8 プラットフォームのサポートを提供する最後のリリースです。詳細については、「CentOS のウェブサイト」を参照してください。

Windows サポート

- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

CloudHSM コマンドラインインターフェイス (CLI) の使用開始

CloudHSM コマンドラインインターフェイス (CLI) を使用すると、 AWS CloudHSM クラスター内のユーザーを管理できます。このトピックを使用して、ユーザーの作成、ユーザーのリスト、CloudHSM CLI のクラスターへの接続など、基本的な HSM ユーザー管理タスクを開始します。

CloudHSM CLI のインストール

次のコマンドを使用して、CloudHSM CLI をダウンロードしてインストールします。

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

ARM64 の Amazon Linux 2 アーキテクチャ:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.aarch64.rpm
```

Amazon Linux 2023

x86_64 アーキテクチャの Amazon Linux 2023:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

ARM64 アーキテクチャの Amazon Linux 2023:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

RHEL 7 (7.8+)

x86_64 アーキテクチャの RHEL 7:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

x86_64 アーキテクチャの RHEL 8:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-cli-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el8.x86_64.rpm
```

RHEL 9 (9.2+)

x86_64 アーキテクチャの RHEL 9:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.x86_64.rpm
```

ARM64 アーキテクチャの RHEL 9:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.aarch64.rpm
```

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/cloudhsm-cli_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u20.04_amd64.deb
```

Ubuntu 22.04 LTS

x86_64 アーキテクチャの Ubuntu 22.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_amd64.deb
```

ARM64 アーキテクチャの Ubuntu 22.04 LTS:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_arm64.deb
```

Windows Server 2016

x86_64 アーキテクチャの Windows Server 2016 の場合は、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2019

x86_64 アーキテクチャの Windows Server 2019 の場合は、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

次のコマンドを使用して CloudHSM CLI を設定します。

クライアント SDK 5 の Linux EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IP addresses of the HSMs>
```

クライアント SDK 5 の Windows EC2 インスタンスをブートストラップするには

- 構成ツールを使用して、クラスターの HSM の IP アドレスを指定します。

```
"C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI IP addresses
 of the HSMs>
```

CloudHSM CLI を使用する

1. CloudHSM CLI を起動するには、次のコマンドを使用します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
```

2. login コマンドを使用して、クラスターにログインします。このコマンドはすべてのユーザーが使用できます。

   次のコマンド例では管理者でログインしています。これは、デフォルトの管理者アカウントです。このユーザーのパスワードは、「クラスターのアクティブ化」を行う場合に設定します。

   ```
   aws-cloudhsm > login --username admin --role admin
   ```

   システムからパスワードの入力を求められます。パスワードを入力するとコマンドが正常に実行されたことが出力で示されます。

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin",
       "role": "admin"
     }
   }
   ```

3. user list コマンドを実行して、クラスター上のすべてのユーザーを一覧表示します。

   ```
   aws-cloudhsm > user list
   {
     "error_code": 0,
     "data": {
       "users": [
         {
           "username": "admin",
           "role": "admin",
           "locked": "false",
           "mfa": [],
           "cluster-coverage": "full"
         },
         {
           "username": "app_user",
           "role": "internal(APPLIANCE_USER)",
   ```

```
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

4. user create を使用して、**example_user** という名前の CU ユーザーを作成します。

   前のステップで管理者ユーザーとしてログインしたので、CU を作成できます。ユーザーの作成および削除や、他のユーザーのパスワード変更などのユーザー管理作業を行うことができるのは、管理者ユーザーのみです。

```
aws-cloudhsm > user create --username example_user --role crypto-user
Enter password:
Confirm password:
{
 "error_code": 0,
 "data": {
   "username": "example_user",
   "role": "crypto-user"
 }
}
```

5. user list を使用して、クラスター上のすべてのユーザーを一覧表示します。

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "example_user",
        "role": "crypto_user",
```

```
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

6. logout コマンドを使用して AWS CloudHSM クラスターからログアウトします。

```
aws-cloudhsm > logout
{
  "error_code": 0,
  "data": "Logout successful"
}
```

7. quit コマンドを使用して CLI を停止します。

```
aws-cloudhsm > quit
```

インタラクティブモードおよびシングルコマンドモード

CloudHSM CLI では、シングルコマンドモードおよびインタラクティブモードの 2 つの方法でコマンドを実行できます。インタラクティブモードはユーザー向けに設計され、シングルコマンドモードはスクリプト向けに設計されています。

> **Note**
>
> すべてのコマンドはインタラクティブモードおよびシングルコマンドモードで動作します。

インタラクティブモード

次のコマンドを使用して CloudHSM CLI インタラクティブモードを起動

Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

Windows

```
C:\Program Files\Amazon\CloudHSM\bin\> .\cloudhsm-cli.exe interactive
```

CLI をインタラクティブモードで使用する場合、login コマンドを使用してユーザーアカウントにログインできます。

CloudHSM CLI コマンドをすべて一覧表示するには、次のコマンドを実行します。

```
aws-cloudhsm > help
```

CloudHSM CLI コマンドの構文を取得するには、次のコマンドを実行します。

```
aws-cloudhsm >  help <command-name>
```

HSM 上のユーザーのリストを取得するには、「user list」と入力します。

```
aws-cloudhsm > user list
```

CloudHSM CLI のセッションを終了するには、次のコマンドを実行します。

```
aws-cloudhsm > quit
```

シングルコマンドモード

シングルコマンドモードを使用して CloudHSM CLI を実行する場合、認証情報を提供するために CLOUDHSM_PIN と CLOUDHSM_ROLE という 2 つの環境変数を設定する必要があります。

```
$ export CLOUDHSM_ROLE=admin
```

```
$ export CLOUDHSM_PIN=admin_username:admin_password
```

これを実行すると、環境に保存されている認証情報を使用してコマンドを実行できます。

```
$ cloudhsm-cli user change-password --username alice --role crypto-user
Enter password:
Confirm password:
{
    "error_code": 0,
    "data": {
      "username": "alice",
      "role": "crypto-user"
    }
}
```

CloudHSM CLI のキー属性

このトピックでは、CloudHSM CLI を使用してキー属性を設定する方法について説明します。CloudHSM CLI のキー属性では、キーのタイプ、キーの機能、またはキーのラベル付け方法を定義できます。一部の属性は固有の特性 (キーのタイプなど) を定義します。その他の属性は「true」または「false」に設定できます。これらの属性を変更すると、キーの機能の一部が有効または無効になります。

キー属性の使用方法を示す例については、親コマンド キー の下にリストされているコマンドを参照してください。

サポートされている属性

ベストプラクティスとして、制限する属性の値のみを設定してください。値を指定しない場合、CloudHSM CLI は次の表で指定されたデフォルト値を使用します。

次の表は、キー属性、指定できる値、デフォルト、および関連する注意事項を示しています。Value 列のセルが空の場合は、属性に割り当てられている特定のデフォルト値がないことを示します。

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
always-sensitive	sensitive が常に True に設定されており、変更されたことがない場合、値は True となります。	いいえ	いいえ

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
check-value	キーのチェック値。詳細については、「その他の詳細」を参照してください。	いいえ	いいえ
class	想定される値: secret-key 、public-key 、private-key 。	いいえ	はい
curve	EC キーペア生成に使用される楕円曲線。 有効な値: secp224r1 、secp256r1 、prime256v1 、secp384r1 、secp256k1 、secp521r1	いいえ	RSA では設定可能、EC では設定不可
decrypt	デフォルト: False	はい	はい
derive	デフォルト: False	はい	はい
destroyable	デフォルト: True	はい	はい

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
ec-point	EC キーの場合、ANSI X9.62 ECPoint 値「Q」の DER エンコーディングを 16 進数形式で表します。 他のキータイプの場合、この属性は存在しません。	いいえ	いいえ
encrypt	デフォルト: False	はい	はい
extractable	デフォルト: True	いいえ	はい
id	デフォルト: 空	いいえ	はい
key-length-bytes	AES キーを生成するために必要です。 有効な値: 16 バイト、24 バイト、32 バイト。	いいえ	いいえ
key-type	想定される値: aes、rsa、ec。	いいえ	はい
label	デフォルト: 空	はい	はい
local	デフォルト: HSM で生成されたキー向け True、HSM にインポートされたキー向け False。	いいえ	いいえ
modifiable	デフォルト: True	いいえ	いいえ

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
modulus	RSA キーペアを生成するために使用されたモジュラス。他のキータイプの場合、この属性は存在しません。	いいえ	いいえ
modulus-size-bits	RSA キーペアを生成するために必要です。 最小値は 2048 です。	いいえ	RSA では設定可能、EC では設定不可
never-extractable	この値は、抽出可能が False に設定されたことがない場合、True となります。 この値は、抽出可能が True に設定されたことがある場合、False となります。	いいえ	いいえ
private	デフォルト: True	いいえ	はい
public-exponent	RSA キーペアを生成するために必要です。 有効な値: 値は、65537 以上の奇数にする必要があります	いいえ	RSA では設定可能、EC では設定不可

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
sensitive	デフォルト: • この値は AES キー、EC および RSA プライベートキーでは True となります。 • この値は EC および RSA パブリックキー用では False となります。	いいえ	プライベートキーでは設定可能で、パブリックキーでは設定できません。
sign	デフォルト: • 値は AES キーでは True となります。 • 値は RSA キーと EC キーでは False となります。	はい	はい
token	デフォルト: False	いいえ	はい
trusted	デフォルト: False	はい	いいえ
unwrap	デフォルト: False	はい	はい
unwrap-template	値は、このラッピングキーを使用してラップ解除されたキーに適用される属性テンプレートを使用する必要があります。	はい	いいえ

CloudHSM CLI 属性	値	key set-attributeで変更可能	キー作成時に設定可能
verify	デフォルト: • 値は AES キーでは True となります。 • 値は RSA キーと EC キーでは False となります。	はい	はい
wrap	デフォルト: False	はい	はい
wrap-template	値は、属性テンプレートを使用し、このラッピングキーでラップされたキーと一致させる必要があります。	はい	いいえ
wrap-with-trusted	デフォルト: False	はい	はい

その他の詳細

値の確認

チェック値は、HSM がキーをインポートまたは生成するときに生成されるキーの 3 バイトのハッシュまたはチェックサムです。キーを エクスポートした後など、HSM の外部でチェック値を計算することもできます。次に、チェック値を比較して、キーのアイデンティティと整合性を確認できます。キーのチェック値を取得するには、キーリストに Verbose (詳細) フラグを付けて使用します。

AWS CloudHSM は、次の標準メソッドを使用してチェック値を生成します。

- 対称キー: ゼロブロックをキーで暗号化した結果の最初の 3 バイト。
- 非対称キーペア: 公開キーの SHA-1 ハッシュの最初の 3 バイト。
- HMAC キー : 現時点では、HMAC キーの KCV はサポートされていません。

関連トピック

- キー
- CloudHSM CLI コマンドのリファレンス

Client SDK 3 CMU および KMU から Client SDK 5 CloudHSM CLI に移行する

このトピックでは、Client SDK 3 コマンドラインツール、CloudHSM 管理ユーティリティ (CMU)、およびキー管理ユーティリティ (KMU) を使用するワークフローを移行し、代わりに Client SDK 5 コマンドラインツール、CloudHSM CLI を使用します。

では AWS CloudHSM、顧客アプリケーションは AWS CloudHSM クライアントソフトウェア開発キット (SDK) を使用して暗号化オペレーションを実行します。Client SDK 5 は、引き続き新機能とプラットフォームサポートが追加されているプライマリ SDK です。このトピックでは、コマンドラインツールの Client SDK 3 から Client SDK 5 への移行に固有の詳細について説明します。

Client SDK 3 には、ユーザーを管理するための CMU と、キーを管理し、キーでオペレーションを実行するための KMU の 2 つの個別のコマンドラインツールが含まれています。Client SDK 5 は、CMU と KMU (Client SDK 3 で提供されたツール) の関数を 1 つのツールである に統合しますCloudHSM コマンドラインインターフェイス (CLI)。ユーザー管理オペレーションは、 サブコマンドユーザーおよび にありますクォーラム。キー管理オペレーションはキーサブコマンド にあり、暗号化オペレーションは暗号化サブコマンド にあります。コマンドの完全なリストCloudHSM CLI コマンドのリファレンスについては、「」を参照してください。

> **Note**
>
> クライアント SDK 3 で、クラスター間の同期に syncKeyおよび syncUser機能に依存している場合は、引き続き CMU を使用します。Client SDK 5 の CloudHSM CLI は現在、この機能をサポートしていません。

Client SDK 5 への移行手順については、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。移行のメリットについては、「」を参照してくださいClient SDK 5 の利点。

CLI の詳細設定

AWS CloudHSM コマンドラインインターフェイス (CLI) には、次の高度な設定が含まれていますが、これはほとんどのお客様が使用する一般的な設定の一部ではありません。これらの設定には追加機能があります。

- 複数のクラスターへの接続

CloudHSM CLI を使用した複数のクラスターへの接続

Client SDK 5 では、単一の CLI インスタンスから複数の CloudHSM クラスターへの接続を許可するように CloudHSM CLI を設定できます。

このトピックの手順を使用して、CloudHSM CLI でマルチクラスター機能を使用して複数のクラスターに接続します。

トピック

- マルチクラスターの前提条件
- マルチクラスター機能用に CloudHSM CLI を設定する
- configure-cli クラスターの追加
- configure-cli 削除クラスター
- 複数のクラスターの使用

マルチクラスターの前提条件

- 接続先の 2 つ以上の AWS CloudHSM クラスターとそのクラスター証明書。
- セキュリティグループが上記のすべてのクラスターに接続するように正しく設定された EC2 インスタンス。クラスターとクライアントインスタンスのセットアップ方法の詳細については、「 の開始方法 AWS CloudHSM」を参照してください。
- マルチクラスター機能を設定するには、CloudHSM CLI を既にダウンロードしてインストールしておく必要があります。これをまだ確認していない場合は、「???」の手順を参照してください。
- で設定されたクラスターには、 に関連付けられ`./configure-cli[.exe] -a`ないためアクセスできません`cluster-id`。このガイドで説明`config-cli add-cluster`されているように、 に従って再設定できます。

マルチクラスター機能用に CloudHSM CLI を設定する

マルチクラスター機能用に CloudHSM CLI を設定するには、次の手順に従います。

1. 接続するクラスターを特定します。
2. 以下で説明add-clusterするように、configure-cli サブコマンドを使用して、これらのクラスターを CloudHSM CLI 設定に追加します。
3. 新しい設定を有効にするには、CloudHSM CLI プロセスを再起動します。

configure-cli クラスターの追加

複数のクラスターに接続する場合は、 `configure-cli add-cluster` コマンドを使用してクラスターを設定に追加します。

構文

```
configure-cli add-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [--region <REGION>]
        [--endpoint <ENDPOINT>]
        [--hsm-ca-cert <HSM CA CERTIFICATE FILE>]
        [--server-client-cert-file <CLIENT CERTIFICATE FILE>]
        [--server-client-key-file <CLIENT KEY FILE>]
        [-h, --help]
```

例

`cluster-id` パラメータを使用してクラスターを追加する

Example

`configure-cli add-cluster` とともに `cluster-id` パラメータを使用して、クラスター (`cluster-1234567` の ID) を設定に追加します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli add-cluster --cluster-id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-cli.exe add-cluster --cluster-
id cluster-1234567
```

> **Tip**
>
> configure-cli add-cluster を cluster-id パラメータと一緒に使用してもクラスターが追加されない場合は、以下の例を参照して、追加するクラスターを識別するための --region と --endpoint パラメータも必要な、より長いバージョンのこのコマンドを参照してください。例えば、クラスターのリージョンが AWS CLI のデフォルトとして設定されているものと異なる場合、適切なリージョンを使用するように --region パラメータを使用する必要があります。さらに、呼び出しに使用する AWS CloudHSM API エンドポイントを指定することもできます。これは、 のデフォルトの DNS ホスト名を使用しない VPC インターフェイスエンドポイントを使用するなど、さまざまなネットワーク設定で必要になる場合があります AWS CloudHSM。

cluster-id、endpoint、および region パラメータを使用してクラスターを追加する

Example

configure-cli add-cluster とともに cluster-id、endpoint、region のパラメータを使用して、クラスター (cluster-1234567 の ID) を設定に追加します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli add-cluster --cluster-id cluster-1234567 --
region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-cli.exe add-cluster --cluster-
id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-
east-1.amazonaws.com
```

--cluster-id、--region、--endpoint パラメータの詳細については、the section called “パラメータ”を参照してください。

パラメータ

--cluster-id ***<Cluster ID>***

`DescribeClusters` を呼び出して、クラスターIDに関連付けられたクラスターのすべての HSM Elastic Network Interface (ENI) IPアドレスを検索します。システムは ENI IP アドレスを設定 AWS CloudHSM ファイルに追加します。

> **Note**
>
> パブリックインターネットにアクセスできない VPC 内の EC2 インスタンスから `--cluster-id`パラメータを使用する場合は、インターフェイス VPC エンドポイントを作成して に接続する必要があります AWS CloudHSM。VPC エンドポイントの詳細については、「???」を参照してください。

必須：はい

--endpoint ***<Endpoint>***

`DescribeClusters` 呼び出しに使用する AWS CloudHSM API エンドポイントを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

必須：いいえ

--hsm-ca-cert ***<HsmCA ##########***

HSM CA 証明書へのファイルパスを指定します。

必須：いいえ

--region ***<Region>***

クラスターのリージョンを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

この `--region` パラメータを指定しない場合、システムは `AWS_DEFAULT_REGION` または `AWS_REGION` の環境変数の読み取りを試みてリージョンを選択します。これらの変数が設定されていない場合、環境変数で別のファイルを指定しない限り、AWS Config (通常は `~/.aws/config`) のプロファイルに関連付けられたリージョンをチェックします`AWS_CONFIG_FILE`。いずれも設定されていない場合は、`us-east-1` デフォルトでリージョンが設定されます。

必須：いいえ

--server-client-cert-file ##################

TLS クライアント・サーバー相互認証に使用するクライアント証明書へのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは `--server-client-key-file` と組み合わせて設定する必要があります。

必須：いいえ

--server-client-key-file ##################

TLS クライアントとサーバーの相互認証に使用されるクライアントキーへのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは `--server-client-cert-file` と組み合わせて設定する必要があります。

必須：いいえ

configure-cli 削除クラスター

CloudHSM CLI を使用して複数のクラスターに接続する場合は、 `configure-cli remove-cluster` コマンドを使用して設定からクラスターを削除します。

構文

```
configure-cli remove-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [-h, --help]
```

例

`cluster-id` パラメータを使用してクラスターを削除します

Example

`configure-cli remove-cluster` とともに `cluster-id` パラメータを使用して、クラスター (`cluster-1234567` の ID) を設定から削除します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli remove-cluster --cluster-id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-cli.exe remove-cluster --cluster-
id cluster-1234567
```

`--cluster-id` パラメータの詳細については、「the section called “パラメータ”」をご参照ください。

パラメータ

--cluster-id *<Cluster ID>*

設定から削除するクラスターの ID。

必須：はい

複数のクラスターの使用

CloudHSM CLI で複数のクラスターを設定したら、 `cloudhsm-cli` コマンドを使用してクラスターを操作します。

例

インタラクティブモード**cluster-id**を使用する場合のデフォルトの設定

Example

`cluster-id` パラメータ???とともに を使用して、設定からデフォルトクラスター (ID が `cluster-1234567`) を設定します。

Linux

```
$ cloudhsm-cli interactive --cluster-id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\cloudhsm-cli.exe interactive --cluster-
id cluster-1234567
```

1 つのコマンドを実行する**cluster-id**ときに を設定する

Example

`cluster-id` パラメータを使用して、クラスター (ID が `cluster-1234567`) の取得???元を設定します。

Linux

```
$ cloudhsm-cli cluster hsm-info --cluster-id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\cloudhsm-cli.exe cluster hsm-info --cluster-
id cluster-1234567
```

CloudHSM CLI コマンドのリファレンス

CloudHSM CLI は、管理者が AWS CloudHSM クラスター内のユーザーを管理するのに役立ちます。CloudHSM CLI は、インタラクティブモードとシングルコマンドモードの 2 つのモードで実行できます。クイックスタートについては、「CloudHSM コマンドラインインターフェイス (CLI) の使用開始」を参照してください。

多くの CloudHSM CLI コマンドを実行するには、CloudHSM CLI を起動し、HSM にログインする必要があります。HSM を追加または削除する場合は、CloudHSM CLI の構成ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

以下のトピックでは、CloudHSM CLI のコマンドについて説明します。

コマンド	説明	ユーザータイプ
クラスターアクティブ化	CloudHSM クラスターをアクティブ化し、クラスターが新しいものであることを確認します。これは他のオペレーションを実行する前に行う必要があります。	非アクティブ管理者

コマンド	説明	ユーザータイプ
cluster hsm-info	クラスター内の HSMsを一覧表示します。	認証されていないユーザーを含むすべての[1]。ログインは必須ではありません。
暗号署名 ECDSA	EC プライベートキーと ECDSA 署名メカニズムを使用して署名を生成します。	Crypto User (CU)
暗号署名 rsa-pkcs	RSA プライベートキーと RSA-PKCS 署名メカニズムを使用して署名を生成します。	CU
暗号記号 rsa-pkcs-pss	RSA プライベートキーと RSA-PKCS-PSS 署名メカニズムを使用して署名を生成します。	CU
ECDSA の暗号化検証	ファイルが特定のパブリックキーによって HSM で署名されていることを確認します。ECDSA 署名メカニズムを使用して署名が生成されたことを確認します。署名付きファイルをソースファイルと比較し、指定された ecdsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。	CU

コマンド	説明	ユーザータイプ
rsa-pkcs の暗号化検証	ファイルが特定のパブリックキーによって HSM で署名されていることを確認します。署名が RSA-PKCS 署名メカニズムを使用して生成されたことを確認します。署名付きファイルをソースファイルと比較し、指定された rsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。	CU
暗号検証 rsa-pkcs-pss	ファイルが特定のパブリックキーによって HSM で署名されていることを確認します。署名が RSA-PKCS-PSS 署名メカニズムを使用して生成されたことを確認します。署名付きファイルをソースファイルと比較し、指定された rsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。	CU
キー削除	AWS CloudHSM クラスターからキーを削除します。	CU
key generate-file	AWS CloudHSM クラスターにキーファイルを生成します。	CU
キー generate-asymmetric-pair rsa	AWS CloudHSM クラスターに非対称 RSA キーペアを生成します。	CU

コマンド	説明	ユーザータイプ
キー generate-asymmetric-pair EC	AWS CloudHSM クラスターに非対称楕円曲線 (EC) キーペアを生成します。	CU
key generate-symmetric aes	AWS CloudHSM クラスターに対称 AES キーを生成します。	CU
key generate-symmetric generic-secret	AWS CloudHSM クラスターに対称汎用シークレットキーを生成します。	CU
キーインポート pem	PEM 形式キーを HSM にインポートします。このコマンドを使用すると、HSM の外部で生成されたパブリックキーをインポートできます。	CU
キーリスト	AWS CloudHSM クラスターに存在する現在のユーザーのすべてのキーを検索します。	CU
キーレプリケート	ソースクラスターからクローンの宛先クラスターにキーをレプリケートします。	CU
key set-attribute	AWS CloudHSM クラスター内のキーの属性を設定します。	CU はこのコマンドを実行でき、管理者は信頼できる属性を設定できます。
キーシェア	AWS CloudHSM クラスター内の他の CUs とキーを共有します。	CU
キー共有解除	AWS CloudHSM クラスター内の他の CUs とキーの共有を解除します。	CU

コマンド	説明	ユーザータイプ
aes-gcm のキーアンラップ	AES ラッピングキーと AES-GCM アンラップメカニズムを使用して、ペイロードキーをクラスターにアンラップします。	CU
キーアンラップ aes-no-pad	AES ラッピングキーと AES-NO-PAD アンラップメカニズムを使用して、ペイロードキーをクラスターにアンラップします。	CU
aes-pkcs5-pad のキーアンラップ	AES ラッピングキーと AES-PKCS5-PAD アンラップメカニズムを使用してペイロードキーをアンラップします。	CU
キーアンラップ aes-zero-pad	AES ラッピングキーと AES-ZERO-PAD アンラップメカニズムを使用して、ペイロードキーをクラスターにアンラップします。	CU
キーアンラップ cloudhsm-aes-gcm	AES ラッピングキーと CLOUDHSM-AES-GCM アンラップメカニズムを使用して、ペイロードキーをクラスターにアンラップします。	CU
キーが rsa-aes をラップ解除する	RSA プライベートキーと RSA-AES アンラップメカニズムを使用してペイロードキーをアンラップします。	CU

コマンド	説明	ユーザータイプ
rsa-oaep のキーアンラップ	RSA プライベートキーと RSA-OAEP アンラップメカニズムを使用してペイロードキーをアンラップします。	CU
rsa-pkcs のキーアンラップ	RSA プライベートキーと RSA-PKCS アンラップメカニズムを使用してペイロードキーをアンラップします。	CU
キーラップ aes-gcm	HSM の AES キーと AES-GCM ラップメカニズムを使用してペイロードキーをラップします。	CU
キーラップ aes-no-pad	HSM の AES キーと AES-NO-PAD ラップメカニズムを使用してペイロードキーをラップします。	CU
キーラップ aes-pkcs5-pad	HSM の AES キーと AES-PKCS5-PAD ラップメカニズムを使用してペイロードキーをラップします。	CU
キーラップ aes-zero-pad	HSM の AES キーと AES-ZERO-PAD ラップメカニズムを使用してペイロードキーをラップします。	CU
キーラップ cloudhsm-aes-gcm	HSM の AES キーと CLOUDHSM-AES-GCM ラップメカニズムを使用してペイロードキーをラップします。	CUs

コマンド	説明	ユーザータイプ
キーラップ rsa-aes	HSM の RSA パブリックキーと RSA-AES ラップメカニズムを使用してペイロードキーをラップします。	CU
キーラップ rsa-oaep	HSM の RSA パブリックキーと RSA-OAEP ラップメカニズムを使用してペイロードキーをラップします。	CU

コマンド	説明	ユーザータイプ
key wrap rsa-pkcs コマンドは、HSM の RSA パブリッククキーとラッピングメカニズムを使用してペイロードキーをRSA-PKCSラップします。ペイロードキーの extractable 属性は に設定する必要がありますtrue。 キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。 key wrap rsa-pkcs コマンドを使用するには、まず AWS CloudHSM クラスターに RSA キーが必要です。キー generate-asymmetric-pair コマンドと wrap 属性を に設定して、RSA キーペアを生成できますtrue。 ユーザーのタイプ このコマンドは、次のタイプのユーザーが実行できます。 • Crypto User (CU) 要件 • このコマンドを実行するには、CU としてログインする必要があります。	HSM の RSA パブリックキーと RSA-PKCS ラップメカニズムを使用してペイロードキーをラップします。	CU

Syntax

```
aws-cloudhsm > help key
```

コマンド	説明	ユーザータイプ
login (ログイン)	AWS CloudHSM クラスターにログインします。	Admin、Crypto User (CU)、および Appliance User (AU)
logout (サインアウト)	AWS CloudHSM クラスターからログアウトします。	Admin、CU、および Appliance User (AU)
quorum token-sign delete	クォーラム承認サービスのトークンを 1 つ以上削除します。	管理
quorum token-sign generate	クォーラム承認サービスのトークンを生成します。	管理
quorum token-sign list	CloudHSM クラスターに存在するすべてのトークン署名のクォーラムトークンを一覧表示します。	認証されていないユーザーを含むすべての [1]。ログインは必須ではありません。
クォーラムトークン署名 list-quorum-values	CloudHSM クラスターに設定されているクォーラム値を一覧表示します。	認証されていないユーザーを含むすべての [1]。ログインは必須ではありません。
quorum token-sign list-timeouts	すべてのトークンタイプのトークンタイムアウト時間を秒単位で取得します。	管理者と暗号化ユーザー
クォーラムトークン署名 set-quorum-value	クォーラム認定サービスの新しいクォーラム値を設定します。	管理
quorum token-sign set-timeout	トークンの種類ごとにトークンのタイムアウト時間を秒単位で設定します。	管理
user change-mfa	ユーザーの多要素認証 (MFA) 戦略を変更します。	Admin、CU

コマンド	説明	ユーザータイプ
user change-password	HSM 上のユーザーのパスワードを変更します。どのユーザーも自分のパスワードを変更できます。管理者は誰でもパスワードを変更できます。	Admin、CU
user create	AWS CloudHSM クラスターにユーザーを作成します。	管理
user delete	AWS CloudHSM クラスター内のユーザーを削除します。	管理
user list	AWS CloudHSM クラスター内のユーザーを一覧表示します。	認証されていないユーザーを含むすべての [1]。ログインは必須ではありません。
ユーザー変更クォーラムトークン署名登録	ユーザーのクォーラムトークン署名クォーラム戦略を登録します。	管理

注釈

- [1] すべてのユーザーには、リストされているすべてのロールとログインしていないユーザーが含まれます。

クラスター

cluster はコマンドグループの親カテゴリであり、親カテゴリと組み合わせるとユーザー固有のコマンドが作成されます。現在、このユーザーカテゴリは次のコマンドで構成されています。

- クラスターアクティブ化
- cluster hsm-info

クラスターアクティブ化

CloudHSM CLI の cluster activate コマンドを使用して新しいクラスターをアクティブ化します。クラスターを使用して暗号化オペレーションを実行するには、まずこのコマンドを実行する必要があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- 非アクティブ管理者

構文

このコマンドにはパラメータはありません。

```
aws-cloudhsm > help cluster activate
Activate a cluster

This command will set the initial Admin password. This process will cause your CloudHSM
 cluster to
move into the ACTIVE state.

USAGE:
    cloudhsm-cli cluster activate [OPTIONS] [--password <PASSWORD>]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --password <PASSWORD>
          Optional: Plaintext activation password If you do not include this argument
 you will be prompted for it

  -h, --help
          Print help (see a summary with '-h')
```

例

このコマンドは、管理者ユーザーの初期パスワードを設定してクラスターをアクティブ化します。

```
aws-cloudhsm > cluster activate
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": "Cluster activation successful"
}
```

関連トピック

- user create
- user delete
- user change-password

cluster hsm-info

CloudHSM CLI の cluster hsm-info コマンドを使用して、クラスター内の HSM を一覧表示します。このコマンドは、CloudHSM CLI にログインしていなくても実行できます。

> **Note**
>
> HSMs を追加または削除する場合は、 AWS CloudHSM クライアントとコマンドラインツールが使用する設定ファイルを更新します。そうしないと、クラスタ内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

Syntax

```
aws-cloudhsm > help cluster hsm-info
List info about each HSM in the cluster

Usage: cloudhsm-cli cluster hsm-info [OPTIONS]
```

```
Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

例

このコマンドは、 AWS CloudHSM クラスターに存在する HSMsを一覧表示します。

```
aws-cloudhsm > cluster hsm-info
{
  "error_code": 0,
  "data": {
    "hsms": [
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
        "serial-number": "5.3G1941-ICM000590",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      },
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
        "serial-number": "5.3G1941-ICM000625",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      },
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
```

```
        "serial-number": "5.3G1941-ICM000663",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      }
    ]
  }
}
```

出力には以下の属性があります。

- Vendor: HSM のベンダー名。
- Model: HSM のモデル番号。
- Serial-number: デバイスのシリアル番号。置換により変更される場合があります。
- H ardware-version-major: ハードウェアのメジャーバージョン。
- H ardware-version-minor: マイナーハードウェアバージョン。
- F irmware-version-major: ファームウェアのメジャーバージョン。
- F irmware-version-minor: マイナーファームウェアのバージョン。
- F irmware-build-number: ファームウェアのビルド番号。
- Firmware-id: ファームウェア ID。ビルドに加えてメジャーバージョンとマイナーバージョンが含まれます。
- FIPS 状態 : クラスターとその中の HSMs FIPS モード。FIPS モードの場合、出力は「2 [FIPS モード、単一要素認証〕」です。非 FIPS モードの場合、出力は「0 [単一要素認証による非 FIPS モード〕」です。

関連トピック

- クラスターアクティブ化

crypto

crypto は、コマンドグループの親カテゴリであり、親カテゴリと組み合わせると、暗号化オペレーションに固有のコマンドが作成されます。現在、このカテゴリは次のコマンドで構成されています。

- 暗号記号
  - 暗号署名 ECDSA
  - 暗号署名 rsa-pkcs
  - 暗号記号 rsa-pkcs-pss
- 暗号検証
  - ECDSA の暗号化検証
  - rsa-pkcs の暗号化検証
  - 暗号検証 rsa-pkcs-pss

暗号記号

crypto sign は、コマンドグループの親カテゴリであり、親カテゴリと組み合わせると、 AWS CloudHSM クラスター内の選択したプライベートキーを使用して署名を生成します。 crypto sign には次のサブコマンドがあります。

- 暗号署名 ECDSA
- 暗号署名 rsa-pkcs
- 暗号記号 rsa-pkcs-pss

を使用するにはcrypto sign、HSM にプライベートキーが必要です。プライベートキーは、次のコマンドで生成できます。

- キー generate-asymmetric-pair EC
- キー generate-asymmetric-pair rsa

暗号署名 ECDSA

crypto sign ecdsa コマンドは、EC プライベートキーと ECDSA 署名メカニズムを使用して署名を生成します。

crypto sign ecdsa コマンドを使用するには、まず AWS CloudHSM クラスターに EC プライベートキーが必要です。`sign` 属性を に設定して、 キー generate-asymmetric-pair EC コマンドを使用して EC プライベートキーを生成できます`true`。

> **Note**
>
> 署名は、 暗号検証 サブコマンド AWS CloudHSM を使用して で検証できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto sign ecdsa
Sign with the ECDSA mechanism

Usage: crypto sign ecdsa --key-filter [<KEY_FILTER>>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be signed
      --data <DATA>
          Base64 Encoded data to be signed
  -h, --help
          Print help
```

例

これらの例は、 crypto sign ecdsaを使用して ECDSA 署名メカニズムとSHA256ハッシュ関数を使用して署名を生成する方法を示しています。このコマンドは HSM でプライベートキーを使用します。

Example 例: 64 でエンコードされたベースデータの署名を生成する

```
aws-cloudhsm > crypto sign ecdsa --key-filter attr.label=ec-private --hash-function
 sha256 --data YWJjMTIz
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007808dd",
    "signature": "4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw=="
  }
}
```

Example 例: データファイルの署名を生成する

```
aws-cloudhsm > crypto sign ecdsa --key-filter attr.label=ec-private --hash-function
 sha256 --data-path data.txt
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007808dd",
    "signature": "4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw=="
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<DATA>

署名する Base64 エンコードデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データパスで指定される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<KEY_FILTER>

キーリファレンス (key-reference=0xabc など) または attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式のキー属性のスペース区切りリストを使用して、一致するキーを選択します。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

関連トピック

- 暗号記号
- 暗号検証

暗号署名 rsa-pkcs

crypto sign rsa-pkcs コマンドは、RSA プライベートキーと RSA-PKCS 署名メカニズムを使用して署名を生成します。

crypto sign rsa-pkcs コマンドを使用するには、まず AWS CloudHSM クラスターに RSA プライベートキーが必要です。`sign` 属性を `true` に設定して key generate-asymmetric-pair rsa コマンドを使用して RSA プライベートキーを生成できます。

> **Note**
>
> 署名は、 verify サブコマンド AWS CloudHSM を使用して で検証できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto sign rsa-pkcs
Sign with the RSA-PKCS mechanism

Usage: crypto sign rsa-pkcs --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be signed
      --data <DATA>
```

```
          Base64 Encoded data to be signed
  -h, --help
          Print help
```

例

これらの例は、 crypto sign rsa-pkcsを使用して RSA-PKCS 署名メカニズムとSHA256ハッシュ関数を使用して署名を生成する方法を示しています。このコマンドは HSM でプライベートキーを使用します。

Example 例: 64 でエンコードされたベースデータの署名を生成する

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private --hash-function
 sha256 --data YWJjMTIz
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ=="
  }
}
```

Example 例: データファイルの署名を生成する

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private --hash-function
 sha256 --data-path data.txt
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ=="
  }
```

```
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<DATA>

署名する Base64 エンコードデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データを通じて提供される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<KEY_FILTER>

キーリファレンス (例: `key-reference=0xabc`) 、または一致するキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するための の形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

関連トピック

- 暗号記号
- 暗号検証

暗号記号 rsa-pkcs-pss

crypto sign rsa-pkcs-pss コマンドは、RSA プライベートキーと署名メカニズムを使用してRSA-PKCS-PSS署名を生成します。

crypto sign rsa-pkcs-pss コマンドを使用するには、まず AWS CloudHSM クラスターに RSA プライベートキーが必要です。RSA プライベートキーは、 `sign` 属性を に設定して キー generate-asymmetric-pair rsa コマンドを使用して生成できますtrue。

> **Note**
>
> 署名は、 暗号検証 サブコマンド AWS CloudHSM を使用して で検証できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto sign rsa-pkcs-pss
Sign with the RSA-PKCS-PSS mechanism

Usage: crypto sign rsa-pkcs-pss [OPTIONS] --key-filter [<KEY_FILTER>...] --
hash-function <HASH_FUNCTION> --mgf <MGF> --salt-length <SALT_LENGTH> <--data-
path <DATA_PATH>|--data <DATA>>

Options:
```

```
      --cluster-id <CLUSTER_ID>        Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]   Key reference (e.g. key-
reference=0xabc) or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key
      --hash-function <HASH_FUNCTION>  [possible values: sha1, sha224, sha256, sha384,
 sha512]
      --data-path <DATA_PATH>          The path to the file containing the data to be
 signed
      --data <DATA>                    Base64 Encoded data to be signed
      --mgf <MGF>                      The mask generation function [possible values:
 mgf1-sha1, mgf1-sha224, mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --salt-length <SALT_LENGTH>      The salt length
  -h, --help                           Print help
```

例

これらの例は、 crypto sign rsa-pkcs-pssを使用して署名メカニズムとSHA256ハッシュ関数を使用してRSA-PKCS-PSS署名を生成する方法を示しています。このコマンドは HSM でプライベートキーを使用します。

Example 例: ベース 64 エンコードデータの署名を生成する

```
aws-cloudhsm > crypto sign rsa-pkcs-pss --key-filter attr.label=rsa-private --hash-
function sha256 --data YWJjMTIz --salt-length 10 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg=="
  }
}
```

Example 例: データファイルの署名を生成する

```
aws-cloudhsm > crypto sign rsa-pkcs-pss --key-filter attr.label=rsa-private --hash-
function sha256 --data-path data.txt --salt-length 10 --mgf mgf1-sha256
```

```
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg=="
  }
}
```


引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<DATA>

署名する Base64 エンコードされたデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データを通じて提供される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<KEY_FILTER>

キーリファレンス (例: `key-reference=0xabc`) 、または一致するキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するための の形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

<MGF>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

必須: はい

<SALT_LENGTH>

ソルトの長さを指定します。

必須: はい

関連トピック

- 暗号記号

- 暗号検証

関連トピック

- 暗号検証

暗号検証

crypto verify は、コマンドグループの親カテゴリであり、親カテゴリと組み合わせると、ファイルが特定のキーによって署名されたかどうかが確認されます。 crypto verify には次のサブコマンドがあります。

- ECDSA の暗号化検証
- rsa-pkcs の暗号化検証
- 暗号検証 rsa-pkcs-pss

crypto verify コマンドは、署名付きファイルをソースファイルと比較し、特定のパブリックキーと署名メカニズムに基づいて、それらが暗号的に関連しているかどうかを分析します。

> **Note**
>
> ファイルは 暗号記号オペレーション AWS CloudHSM でサインインできます。

ECDSA の暗号化検証

crypto verify ecdsa コマンドは、次のオペレーションを完了するために使用されます。

- 特定のパブリックキーによって HSM でファイルが署名されていることを確認します。
- ECDSA 署名メカニズムを使用して署名が生成されたことを確認します。
- 署名付きファイルをソースファイルと比較し、特定の ecdsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。

crypto verify ecdsa コマンドを使用するには、まず AWS CloudHSM クラスターに EC パブリックキーが必要です。`verify` 属性を に設定して キーインポート pem コマンドを使用して EC パブリックキーをインポートできます`true`。

> **Note**
>
> 暗号記号 サブコマンドを使用して CloudHSM CLI で署名を生成できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto verify ecdsa
Verify with the ECDSA mechanism

Usage: crypto verify ecdsa --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>> <--signature-
path <SIGNATURE_PATH>|--signature <SIGNATURE>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be verified
      --data <DATA>
          Base64 encoded data to be verified
      --signature-path <SIGNATURE_PATH>
          The path to where the signature is located
```

```
      --signature <SIGNATURE>
          Base64 encoded signature to be verified
  -h, --help
          Print help
```

例

これらの例は、 crypto verify ecdsaを使用して ECDSA 署名メカニズムとSHA256ハッシュ関数を使用して生成された署名を検証する方法を示しています。このコマンドは HSM でパブリックキーを使用します。

Example 例: Base64 でエンコードされた署名を Base64 でエンコードされたデータで検証する

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --key-filter attr.label=ec-
public --data YWJjMTIz --signature 4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw==
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: データファイルを使用して署名ファイルを検証する

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --key-filter attr.label=ec-
public --data-path data.txt --signature-path signature-file
{
   "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: 偽の署名関係の証明

このコマンドは、 にある署名を生成する ECDSA 署名メカニズムecdsa-publicを使用して、 にあるデータが ラベルを持つパブリックキーによって署名/home/dataされたかどうかを確認します/home/signature。指定された引数は真の署名関係を構成しないため、コマンドはエラーメッセージを返します。

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --
key-filter attr.label=ec-public --data aW52YWxpZA== --signature
 +ogk7M7S3iTqFg3SndJfd91dZFr5Qo6YixJl8JwcvqqVgsVuO6o+VKvTRjz0/V05kf3JJbBLr87Q
+wLWcMAJfA==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<DATA>

署名する Base64 エンコードデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データパスで指定される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<KEY_FILTER>

キーリファレンス (例: `key-reference=0xabc`) 、または一致するキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するための の形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

####

Base64 でエンコードされた署名。

必須: はい (署名パスで指定される場合を除く)

<SIGNATURE_PATH>

署名の場所を指定します。

必須: はい (署名パスで指定される場合を除く)

関連トピック

- 暗号記号
- 暗号検証

rsa-pkcs の暗号化検証

crypto verify rsa-pkcs コマンドは、次のオペレーションを完了するために使用されます。

- ファイルが特定のパブリックキーによって HSM で署名されていることを確認します。
- 署名メカニズムを使用して`RSA-PKCS`署名が生成されたことを確認します。
- 署名付きファイルをソースファイルと比較し、指定された rsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。

crypto verify rsa-pkcs コマンドを使用するには、まず AWS CloudHSM クラスターに RSA パブリックキーが必要です。

> **Note**
>
> 暗号記号 サブコマンドで CloudHSM CLI を使用して署名を生成できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto verify rsa-pkcs
Verify with the RSA-PKCS mechanism

Usage: crypto verify rsa-pkcs --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>> <--signature-
path <SIGNATURE_PATH>|--signature <SIGNATURE>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be verified
      --data <DATA>
          Base64 encoded data to be verified
      --signature-path <SIGNATURE_PATH>
          The path to where the signature is located
```

```
      --signature <SIGNATURE>
          Base64 encoded signature to be verified
  -h, --help
          Print help
```

例

これらの例は、 crypto verify rsa-pkcsを使用して、RSA-PKCS 署名メカニズムとSHA256ハッシュ関数を使用して生成された署名を検証する方法を示しています。このコマンドは HSM でパブリックキーを使用します。

Example 例: Base64 でエンコードされた署名を Base64 でエンコードされたデータで検証する

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data YWJjMTIz --signature XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ==
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: データファイルを使用して署名ファイルを検証する

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data-path data.txt --signature-path signature-file
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: 偽の署名関係の証明

このコマンドは、RSAPKCS 署名メカニズムrsa-publicを使用して にある署名を生成し、無効なデータが ラベル付きのパブリックキーによって署名されたかどうかを確認します/home/

signature。指定された引数は真の署名関係を構成しないため、コマンドはエラーメッセージを返します。

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data aW52YWxpZA== --signature XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#####

署名する Base64 エンコードデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データパスで指定される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256

- sha384
- sha512

必須: はい

<KEY_FILTER>

一致するキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

####

Base64 でエンコードされた署名。

必須: はい (署名パスで指定されない限り)

<SIGNATURE_PATH>

署名の場所を指定します。

必須: はい (署名パスで指定されない限り)

関連トピック

- 暗号記号
- 暗号検証

暗号検証 rsa-pkcs-pss

crypto sign rsa-pkcs-pss コマンドは、次のオペレーションを完了するために使用されます。

- ファイルが特定のパブリックキーによって HSM で署名されていることを確認します。
- RSA-PKCS-PSS 署名メカニズムを使用して署名が生成されたことを確認します。
- 署名付きファイルをソースファイルと比較し、指定された rsa パブリックキーと署名メカニズムに基づいて、その 2 つが暗号的に関連しているかどうかを判断します。

crypto verify rsa-pkcs-pss コマンドを使用するには、まず AWS CloudHSM クラスターに RSA パブリックキーが必要です。`verify` 属性を に設定して、キーインポート pem コマンド ADD UNWRAP LINK HERE) を使用して RSA パブリックキーをインポートできます`true`。

> **Note**
>
> 暗号記号 サブコマンドで CloudHSM CLI を使用して署名を生成できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help crypto verify rsa-pkcs-pss
Verify with the RSA-PKCS-PSS mechanism

Usage: crypto verify rsa-pkcs-pss --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --salt-length >SALT_LENGTH< <--data-
path <DATA_PATH>|--data <DATA> <--signature-path <SIGNATURE_PATH>|--
signature <SIGNATURE>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
```

```
          The path to the file containing the data to be verified
      --data <DATA>
          Base64 encoded data to be verified
      --signature-path <SIGNATURE_PATH>
          The path to where the signature is located
      --signature <SIGNATURE>
          Base64 encoded signature to be verified
      --mgf <MGF>
          The mask generation function [possible values: mgf1-sha1, mgf1-sha224, mgf1-
sha256, mgf1-sha384, mgf1-sha512]
      --salt-length <SALT_LENGTH>
          The salt length
  -h, --help
          Print help
```

例

これらの例は、 crypto verify rsa-pkcs-pssを使用して、RSA-PKCS-PSS 署名メカニズムとSHA256ハッシュ関数を使用して生成された署名を検証する方法を示しています。このコマンドはHSM でパブリックキーを使用します。

Example 例: Base64 でエンコードされた署名を Base64 でエンコードされたデータで検証する

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public
 --hash-function sha256 --data YWJjMTIz --salt-length 10 --mgf mgf1-sha256
 --signature H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg==
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: データファイルを使用して署名ファイルを検証する

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public --hash-
function sha256 --data-path data.txt --salt-length 10 --mgf mgf1-sha256 --signature
 signature-file
```

```
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example 例: 偽の署名関係の証明

このコマンドは、RSAPKCSPSS 署名メカニズムrsa-publicを使用して、無効なデータが にある署名を生成するラベル付きのパブリックキーによって署名されたかどうかを確認します/home/signature。指定された引数は真の署名関係を構成しないため、コマンドはエラーメッセージを返します。

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public
 --hash-function sha256 --data aW52YWxpZA== --salt-length 10 --mgf mgf1-sha256
 --signature H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<DATA>

署名する Base64 エンコードされたデータ。

必須: はい (データパスで指定される場合を除く)

<DATA_PATH>

署名するデータの場所を指定します。

必須: はい (データパスで指定される場合を除く)

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<KEY_FILTER>

キーリファレンス (例: `key-reference=0xabc`) 、または一致するキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するための の形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、CloudHSM CLI のキー属性」を参照してください。

必須: はい

<MFG>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256

- mgf1-sha384
- mgf1-sha512

必須: はい

####

Base64 でエンコードされた署名。

必須: はい (署名パスで指定されない限り)

<SIGNATURE_PATH>

署名の場所を指定します。

必須: はい (署名パスで指定されない限り)

関連トピック

- 暗号記号
- 暗号検証

キー

key はコマンドグループの親カテゴリであり、親カテゴリと組み合わせるとキー固有のコマンドが作成されます。現在、このカテゴリは次のコマンドで構成されています。

- キー削除
- key generate-file
- キー generate-asymmetric-pair
  - キー generate-asymmetric-pair rsa
  - キー generate-asymmetric-pair EC
- key generate-symmetric
  - key generate-symmetric aes
  - key generate-symmetric generic-secret
- キーインポート pem
- キーリスト

- キーレプリケート
- key set-attribute
- キーシェア
- キー共有解除
- キーアンラップ
- キーラップ

キー削除

CloudHSM CLI の key delete コマンドを使用して、 AWS CloudHSM クラスターからキーを削除します。一度に削除できるキーは 1 つだけです。キーペアの一方のキーを削除しても、ペアの他方のキーには影響がありません。

キーを作成し、そのキーを所有している CU のみがキーを削除できます。キーを共有しているが所有者ではないユーザーは、暗号化オペレーションでキーを使用できますが、削除することはできません。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key delete
Delete a key in the HSM cluster

Usage: key delete [OPTIONS] --filter [<FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
```

```
      --filter [<FILTER>...]     Key reference (e.g. key-reference=0xabc)
 or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key for deletion
  -h, --help                     Print help
```

例

```
aws-cloudhsm > key delete --filter attr.label="ec-test-public-key"
{
  "error_code": 0,
  "data": {
    "message": "Key deleted successfully"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

キーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`を使用して、一致するキーを選択して削除します。

サポートされている CloudHSM CLI キー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください

必須: はい

関連トピック

- キーリスト
- key generate-file
- キー共有解除
- CloudHSM CLI のキー属性

- CloudHSM CLI を使用してキーをフィルタリングする

key generate-file

key generate-file コマンドは HSM から非対称キーをエクスポートします。ターゲットがプライベートキーの場合、プライベートキーへの参照はフェイク PEM 形式でエクスポートされます。ターゲットがパブリックキーの場合、パブリックキーのバイトは PEM 形式でエクスポートされます。

実際のプライベートキーマテリアルは含まず、HSM のプライベートキーを参照するフェイク PEM ファイルを使用して、ウェブサーバーから への SSL/TLS オフロードを確立できます AWS CloudHSM。詳細については、「SSL/TLS オフロード」を参照してください。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key generate-file
Generate a key file from a key in the HSM cluster. This command does not export any
 private key data from the HSM

Usage: key generate-file --encoding <ENCODING> --path <PATH> --filter [<FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --encoding <ENCODING>
          Encoding format for the key file

          Possible values:
          - reference-pem: PEM formatted key reference (supports private keys)
          - pem:           PEM format (supports public keys)
```

```
      --path <PATH>
          Filepath where the key file will be written

      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for file generation

  -h, --help
          Print help (see a summary with '-h')
```

例

この例では、 key generate-fileを使用して AWS CloudHSM クラスターにキーファイルを生成する方法を示します。

Example

```
aws-cloudhsm > key generate-file --encoding reference-pem --path /tmp/ec-private-
key.pem --filter attr.label="ec-test-private-key"
{
  "error_code": 0,
  "data": {
    "message": "Successfully generated key file"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

キーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`を使用して、一致するキーを選択して削除します。

サポートされている CloudHSM CLI キー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください

必須: いいえ

<ENCODING>

キーファイルのエンコード形式を指定します

必須: はい

<PATH>

キーファイルが書き込まれるファイルパスを指定します

必須: はい

関連トピック

- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする
- キー generate-asymmetric-pair
- key generate-symmetric

キー generate-asymmetric-pair

key generate-asymmetric-pair はコマンドグループの親カテゴリで、親カテゴリと組み合わせると非対称キーペアを生成するコマンドが作成されます。現在、このカテゴリは次のコマンドで構成されています。

- キー generate-asymmetric-pair EC
- キー generate-asymmetric-pair rsa

キー generate-asymmetric-pair EC

CloudHSM CLI の key asymmetric-pair ec コマンドを使用して、 AWS CloudHSM クラスターに非対称楕円曲線 (EC) キーペアを生成します。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key generate-asymmetric-pair ec
Generate an Elliptic-Curve Cryptography (ECC) key pair

Usage: key generate-asymmetric-pair ec [OPTIONS] --public-label <PUBLIC_LABEL> --
private-label <PRIVATE_LABEL> --curve <CURVE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --public-label <PUBLIC_LABEL>
          Label for the public key
      --private-label <PRIVATE_LABEL>
          Label for the private key
      --session
          Creates a session key pair that exists only in the current session. The key
 cannot be recovered after the session ends
      --curve <CURVE>
          Elliptic curve used to generate the key pair [possible values: prime256v1,
 secp256r1, secp224r1, secp384r1, secp256k1, secp521r1]
      --public-attributes [<PUBLIC_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated EC public key
 in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --private-attributes [<PRIVATE_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated EC private
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
  -h, --help
          Print help
```

例

以下の例では、key generate-asymmetric-pair ec コマンドを使用して EC キーペアを作成する方法を示します。

Example 例: EC キーペアの作成

```
aws-cloudhsm > key generate-asymmetric-pair ec \
```

```
    --curve secp224r1 \
    --public-label ec-public-key-example \
    --private-label ec-private-key-example
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x000000000012000b",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-public-key-example",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
```

```
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f38a
        "curve": "secp224r1"
      }
    },
"private_key": {
      "key-reference": "0x000000000012000c",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-private-key-example",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f38a
```

```
        "curve": "secp224r1"
      }
    }
  }
}
```

Example 例: オプションの属性を持つ EC キーペア作成

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve secp224r1 \
    --public-label ec-public-key-example \
    --private-label ec-private-key-example \
    --public-attributes token=true encrypt=true \
    --private-attributes token=true decrypt=true
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x00000000002806eb",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-public-key-example",
        "id": "",
        "check-value": "0xedef86",
        "class": "public-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
```

```
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
        "ec-point":
"0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280c82",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-private-key-example",
        "id": "",
        "check-value": "0xedef86",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
```

```
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<CURVE>

楕円曲線の識別子を指定します。

- prime256v1
- secp256r1
- secp224r1
- secp384r1
- secp256k1
- secp521r1

必須: はい

<PUBLIC_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された EC パブリックキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<PUBLIC_LABEL>

パブリックキーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<PRIVATE_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された EC プライベートキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<PRIVATE_LABEL>

プライベートキーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

デフォルトでは、生成されるキーは永続 (トークン) キーです。<SESSION> で渡すことでこれが変わり、この引数で生成されたキーがセッション (エフェメラル) キーであることが保証されます。

必須: いいえ

関連トピック

- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする

キー generate-asymmetric-pair rsa

key generate-asymmetric-pair rsa コマンドを使用すると、 AWS CloudHSM クラスターに非対称 RSA キーペアが生成されます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key generate-asymmetric-pair rsa
Generate an RSA key pair

Usage: key generate-asymmetric-pair rsa [OPTIONS] --public-label <PUBLIC_LABEL>
 --private-label <PRIVATE_LABEL> --modulus-size-bits <MODULUS_SIZE_BITS> --public-
exponent <PUBLIC_EXPONENT>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --public-label <PUBLIC_LABEL>
          Label for the public key
      --private-label <PRIVATE_LABEL>
          Label for the private key
      --session
          Creates a session key pair that exists only in the current session. The key
 cannot be recovered after the session ends
      --modulus-size-bits <MODULUS_SIZE_BITS>
```

```
          Modulus size in bits used to generate the RSA key pair
      --public-exponent <PUBLIC_EXPONENT>
          Public exponent used to generate the RSA key pair
      --public-attributes [<PUBLIC_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated RSA public
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --private-attributes [<PRIVATE_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated RSA private
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
  -h, --help
          Print help
```

例

以下の例では、key generate-asymmetric-pair rsa を使用して RSA キーペアを作成する方法を示します。

Example 例: RSA キーペアの作成

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000160010",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "",
```

```
        "check-value": "0x498e1f",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xdfca0669dc8288ed3bad99509bd21c7e6192661407021b3f4cdf4a593d939dd24f4d641af8e4e73b04c847731c6
e89a065e7d1a46ced96b46b909db2ab6be871ee700fd0a448b6e975bb64cae77c49008749212463e37a577baa57ce3e
bcebb7d20bd6df1948ae336ae23b52d73b7f3b6acc2543edb6358e08d326d280ce489571f4d34e316a2ea1904d513ca
        "modulus-size-bits": 2048
      }
    },
"private_key": {
      "key-reference": "0x0000000000160011",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
```

```
        "id": "",
        "check-value": "0x498e1f",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
 "0xdfca0669dc8288ed3bad99509bd21c7e6192661407021b3f4cdf4a593d939dd24f4d641af8e4e73b04c847731c6
        "modulus-size-bits": 2048
      }
    }
  }
}
```

Example 例: オプションの属性を含む RSA キーペアの作成

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example \
--public-attributes token=true encrypt=true \
--private-attributes token=true decrypt=true
{
  "error_code": 0,
  "data": {
```

```
    "public_key": {
      "key-reference": "0x0000000000280cc8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "public-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
73a80fdb457aa7b20cd61e486c326e2cfd5e124a7f6a996437437812b542e3caf85928aa866f0298580f7967ee6aa01
f6e6296d6c116d5744c6d60d14d3bf3cb978fe6b75ac67b7089bafd50d8687213b31abc7dc1bad422780d29c851d510
133022653225bd129f8491101725e9ea33e1ded83fb57af35f847e532eb30cd7e726f23910d2671c6364092e834697e
ac3160f0ca9725d38318b7",
```

```
        "modulus-size-bits": 2048
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280cc7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
  "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
        "modulus-size-bits": 2048
```

```
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<MODULUS_SIZE_BITS>

モジュラスの長さをビット単位で指定します。最小値は 2048 です。

必須: はい

<PRIVATE_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された RSA プライベートキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<PRIVATE_LABEL>

プライベートキーのユーザー定義ラベルを指定します。で使用できる最大サイズ`label`は、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<PUBLIC_EXPONENT>

パブリック指数を指定します。値は、65537 以上の奇数にする必要があります

必須: はい

<PUBLIC_KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された RSA パブリックキーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<PUBLIC_LABEL>

パブリックキーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

デフォルトでは、生成されるキーは永続 (トークン) キーです。<SESSION> で渡すことでこれが変わり、この引数で生成されたキーがセッション (エフェメラル) キーであることが保証されます。

必須: いいえ

関連トピック

- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする

key generate-symmetric

key generate-symmetric はコマンドグループの親カテゴリであり、親カテゴリと組み合わせると対称キーを生成するコマンドが作成されます。現在、このカテゴリは次のコマンドで構成されています。

- key generate-symmetric aes
- key generate-symmetric generic-secret

key generate-symmetric aes

key generate-symmetric aes コマンドは、 AWS CloudHSM クラスターに対称 AES キーを生成します。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key generate-symmetric aes
Generate an AES key

Usage: key generate-symmetric aes [OPTIONS] --label <LABEL> --key-length-
bytes <KEY_LENGTH_BYTES>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --label <LABEL>
          Label for the key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --key-length-bytes <KEY_LENGTH_BYTES>
          Key length in bytes
      --attributes [<KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated AES key in
 the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
  -h, --help
          Print help
```

例

以下の例では、key generate-symmetric aes コマンドを使って AES キーを作成する方法を示します。

Example 例: AES キーの作成

```
aws-cloudhsm > key generate-symmetric aes \
--label example-aes \
--key-length-bytes 24
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e06bf",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "aes",
        "label": "example-aes",
        "id": "",
        "check-value": "0x9b94bd",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
```

```
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 24
      }
    }
  }
}
```

Example 例: オプションの属性を含む AES キーペアの作成

```
aws-cloudhsm > key generate-symmetric aes \
--label example-aes \
--key-length-bytes 24 \
--attributes decrypt=true encrypt=true
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e06bf",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "aes",
        "label": "example-aes",
        "id": "",
        "check-value": "0x9b94bd",
        "class": "secret-key",
        "encrypt": true,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
```

```
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 24
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された AES キーに設定するキー属性のスペース区切りリストを指定します (例: token=true)。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<KEY-LENGTH-BYTES>

キーの長さをバイト単位で指定します。

有効値:

- 16、24、32

必須: はい

<LABEL>

AES キーのユーザー定義ラベルを指定します。で使用できる最大サイズlabelは、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

デフォルトでは、生成されるキーは永続 (トークン) キーです。<SESSION> で渡すことでこれが変わり、この引数で生成されたキーがセッション (エフェメラル) キーであることが保証されます。

必須: いいえ

関連トピック

- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする

key generate-symmetric generic-secret

key generate-asymmetric-pair コマンドは、AWS CloudHSM クラスターに対称汎用シークレットキーを生成します。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > key help generate-symmetric generic-secret
Generate a generic secret key

Usage: key generate-symmetric generic-secret [OPTIONS] --label <LABEL> --key-length-
bytes <KEY_LENGTH_BYTES>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --label <LABEL>
          Label for the key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --key-length-bytes <KEY_LENGTH_BYTES>
          Key length in bytes
      --attributes [<KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated generic
 secret key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
  -h, --help
          Print help
```

例

次の例は、key generate-symmetric generic-secret コマンドを使って汎用シークレットキーを作成する方法を示しています。

Example 例: 汎用シークレットキー作成

```
aws-cloudhsm > key generate-symmetric generic-secret \
--label example-generic-secret \
--key-length-bytes 256
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e08fd",
      "key-info": {
        "key-owners": [
```

```
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "generic-secret",
        "label": "example-generic-secret",
        "id": "",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 256
      }
    }
  }
}
```

Example 例: オプションの属性を持つ汎用シークレットキー作成

```
aws-cloudhsm > key generate-symmetric generic-secret \
--label example-generic-secret \
--key-length-bytes 256 \
--attributes token=true encrypt=true
```

```
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e08fd",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "generic-secret",
        "label": "example-generic-secret",
        "id": "",
        "class": "secret-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 256
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<KEY_ATTRIBUTES>

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE 形式で生成された AES キーに設定するキー属性のスペース区切りリストを指定します (例: `token=true`)。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

<KEY-LENGTH-BYTES>

キーの長さをバイト単位で指定します。

有効値:

- 1 ~ 800

必須: はい

<LABEL>

汎用シークレットキーのユーザー定義ラベルを指定します。で使用できる最大サイズ`label`は、クライアント SDK 5.11 以降では 127 文字です。Client SDK 5.10 以前では、126 文字に制限されています。

必須: はい

<SESSION>

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

デフォルトでは、生成されるキーは永続 (トークン) キーです。<SESSION> で渡すことでこれが変わり、この引数で生成されたキーがセッション (エフェメラル) キーであることが保証されます。

必須: いいえ

関連トピック

- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする

キーインポート pem

の key import pem コマンドは、PEM 形式キーを HSM に AWS CloudHSM インポートします。このコマンドを使用すると、HSM の外部で生成されたパブリックキーをインポートできます。

> **Note**
>
> key generate-file コマンドを使用して、パブリックキーから標準 PEM ファイルを作成するか、プライベートキーから参照 PEM ファイルを作成します。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key import pem
Import key from a PEM file

Usage: key import pem [OPTIONS] --path <PATH> --label <LABEL> --key-type-
class <KEY_TYPE_CLASS>
```

```
Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --path PATH>
          Path where the key is located in PEM format
      --label LABEL>
          Label for the imported key
      --key-type-class KEY_TYPE_CLASS>
          Key type and class of the imported key [possible values: ec-public, rsa-
public]
      --attributes [IMPORT_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the imported key
  -h, --help
          Print help
```

例

この例では、 key import pem コマンドを使用して、PEM 形式のファイルから RSA パブリックキーをインポートする方法を示します。

Example 例: RSA パブリックキーをインポートする

```
aws-cloudhsm > key import pem --path /home/example --label example-imported-key --key-
type-class rsa-public
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001e08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
```

```
        "key-type": "rsa",
        "label": "example-imported-key",
        "id": "0x",
        "check-value": "0x99fe93",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0x8e9c172c37aa22ed1ce25f7c3a7c936dadc532201400128b044ebb4b96#··3e4930ab910df5a2896eaeb8853cfe
        "modulus-size-bits": 2048
      }
    },
    "message": "Successfully imported key"
  }
}
```

Example 例: オプションの属性を持つ RSA パブリックキーをインポートする

```
aws-cloudhsm > key import pem --path /home/example --label example-imported-key-with-
attributes --key-type-class rsa-public --attributes verify=true
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001e08e3",
```

```
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "example-imported-key-with-attributes",
        "id": "0x",
        "check-value": "0x99fe93",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
"0x8e9c172c37aa22ed1ce25f7c3a7c936dadc532201400128b044ebb4b96#··3e4930ab910df5a2896eaeb8853cfe
        "modulus-size-bits": 2048
      }
    },
    "message": "Successfully imported key"
  }
```

```
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PATH>

キーファイルが配置されているファイルパスを指定します。

必須: はい

<LABEL>

インポートされたキーのユーザー定義ラベルを指定します。`label` の最大長は 126 文字です。

必須: はい

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス。

使用できる値:

- EC-パブリック
- rsa-public

必須: はい

<IMPORT_KEY_ATTRIBUTES>

インポートされたキーに設定するキー属性のスペース区切りリストを の形式で指定します `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (例: `token=true`) 。サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: いいえ

関連トピック

- 暗号記号

- 暗号検証

キーリスト

key list コマンドは、 AWS CloudHSM クラスターに存在する現在のユーザーのすべてのキーを検索します。出力には、そのユーザーが所有および共有しているキーと、CloudHSM クラスターのすべてのパブリックキーが含まれます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

構文

```
aws-cloudhsm > help key list
List the keys the current user owns, shares, and all public keys in the HSM cluster

Usage: key list [OPTIONS]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select
 matching key(s) to list
      --max-items <MAX_ITEMS>
          The total number of items to return in the command's output. If the total
 number of items available is more than the value specified, a next-token is provided
 in the command's output. To resume pagination, provide the next-token value in the
 starting-token argument of a subsequent command [default: 10]
      --starting-token <STARTING_TOKEN>
          A token to specify where to start paginating. This is the next-token from a
 previously truncated response
  -v, --verbose
          If included, prints all attributes and key information for each matched key.
 By default each matched key only displays its key-reference and label attribute
  -h, --help
```

```
          Print help
```

例

次の例は、key list コマンドを実行するためのさまざまな方法を示しています。

Example 例: すべてのキーを検索 - デフォルト

このコマンドは、 AWS CloudHSM クラスターに存在するログインしているユーザーのキーを一覧表示します。

> **Note**
>
> デフォルトでは、現在ログインしているユーザーのキーは 10 個だけ表示され、出力には `key-reference` と `label` のみ表示されます。適切なページ分割オプションを使用して、出力として表示するキーの数を増やしたり減らしたりします。

```
aws-cloudhsm > key list
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000000003d5",
        "attributes": {
          "label": "test_label_1"
        }
      },
      {
        "key-reference": "0x0000000000000626",
        "attributes": {
          "label": "test_label_2"
        }
      },.
      ...8 keys later...
    ],
    "total_key_count": 56,
    "returned_key_count": 10,
    "next_token": "10"
  }
}
```

Example 例: すべてのキーを検索 - 詳細度

出力には、そのユーザーが所有および共有しているキーと、HSM のすべてのパブリックキーが含まれます。

> **Note**
>
> 注: デフォルトでは、現在ログインしているユーザーのキーは 10 個だけ表示されます。適切なページ分割オプションを使用して、出力として表示するキーの数を増やしたり減らしたりします。

```
aws-cloudhsm > key list --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x000000000012000c",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "session"
        },
        "attributes": {
          "key-type": "ec",
          "label": "ec-test-private-key",
          "id": "",
          "check-value": "0x2a737d",
          "class": "private-key",
          "encrypt": false,
          "decrypt": false,
          "token": false,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
```

```
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": false,
          "trusted": false,
          "unwrap": false,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 122,
          "ec-point":
 "0x0442d53274a6c0ec1a23c165dcb9ccdd72c64e98ae1a9594bb5284e752c746280667e11f1e983493c1c605e0a80
          "curve": "secp224r1"
        }
      },
      {
        "key-reference": "0x000000000012000d",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "session"
        },
        "attributes": {
          "key-type": "ec",
          "label": "ec-test-public-key",
          "id": "",
          "check-value": "0x2a737d",
          "class": "public-key",
          "encrypt": false,
          "decrypt": false,
          "token": false,
          "always-sensitive": false,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
```

```
          "never-extractable": false,
          "private": true,
          "sensitive": false,
          "sign": false,
          "trusted": false,
          "unwrap": false,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 57,
          "ec-point":
 "0x0442d53274a6c0ec1a23c165dcb9ccdd72c64e98ae1a9594bb5284e752c746280667e11f1e983493c1c605e0a80
          "curve": "secp224r1"
        }
      }
    ],
      ...8 keys later...
    "total_key_count": 1580,
    "returned_key_count": 10
  }
}
```

Example 例: ページ分割されたリターン

次の例では、2 つのキーのみを表示するページ分割されたキーのサブセットを表示しています。次に、この例では次の 2 つのキーを表示する後続の呼び出しを行います。

```
aws-cloudhsm > key list --verbose --max-items 2
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000000030",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "full"
```

```
        },
        "attributes": {
          "key-type": "aes",
          "label": "98a6688d1d964ed7b45b9cec5c4b1909",
          "id": "",
          "check-value": "0xb28a46",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 32
        }
      },
      {
        "key-reference": "0x0000000000000042",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "4ad6cdcbc02044e09fa954143efde233",
          "id": "",
```

```
          "check-value": "0xc98104",
          "class": "secret-key",
          "encrypt": true,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": true,
          "wrap": true,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    ],
    "total_key_count": 1580,
    "returned_key_count": 2,
    "next_token": "2"
  }
}
```

次の 2 つのキーを表示するには、後続の呼び出しを行います。

```
aws-cloudhsm > key list --verbose --max-items 2 --starting-token 2
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000000081",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
```

```
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "6793b8439d044046982e5b895791e47f",
        "id": "",
        "check-value": "0x3f986f",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 32
      }
    },
    {
      "key-reference": "0x0000000000000089",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
```

```
        },
        "attributes": {
          "key-type": "aes",
          "label": "56b30fa05c6741faab8f606d3b7fe105",
          "id": "",
          "check-value": "0xe9201a",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 32
        }
      }
    ],
    "total_key_count": 1580,
    "returned_key_count": 2,
    "next_token": "4"
  }
}
```


CloudHSM CLI でキーフィルタリングメカニズムがどのように機能するかを示すその他の例については、CloudHSM CLI を使用してキーをフィルタリングする を参照してください。

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

キーリファレンス (例: `key-reference=0xabc`) または の形式のキー属性のスペース区切りリスト。一致するキー (複数可) `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` を選択して一覧表示します。

サポートされている CloudHSM CLI キー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください

必須: いいえ

<MAX_ITEMS>

コマンドの出力で返される項目の総数。使用可能な項目の総数が指定された値を上回る場合、コマンドの出力で next-token が提供されます。ページ分割を再開するには、後続コマンドの starting-token 引数で next-token 値を指定します。

必須: いいえ

<STARTING_TOKEN>

ページ分割を始める場所を指定するトークン。これは、以前に切り詰められたレスポンスからの next-token です。

必須: いいえ

<VERBOSE>

含まれている場合は、一致した各キーのすべての属性とキー情報を出力します。デフォルトでは、一致した各キーには key-reference とラベル属性のみが表示されます。

必須: いいえ

関連トピック

- キー削除
- key generate-file
- キー共有解除
- CloudHSM CLI のキー属性
- CloudHSM CLI を使用してキーをフィルタリングする

キーレプリケート

key replicate コマンドは、ソース AWS CloudHSM クラスターから宛先 AWS CloudHSM クラスターにキーをレプリケートします。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

> **Note**
>
> Crypto ユーザーは、このコマンドを使用するには キーを所有している必要があります。

要件

- 送信元クラスターと送信先クラスターはクローンである必要があります。つまり、一方はもう一方のバックアップから作成されたか、両方とも共通のバックアップから作成されたことを意味します。詳細については、「バックアップからクラスターを作成する」を参照してください。
- キーの所有者は、送信先クラスターに存在する必要があります。さらに、キーがユーザーと共有されている場合、それらのユーザーは送信先クラスターにも存在する必要があります。
- このコマンドを実行するには、送信元クラスターと送信先クラスターの両方で CU としてログインする必要があります。
  - シングルコマンドモードでは、 コマンドは CLOUDHSM_PIN および CLOUDHSM_ROLE 環境変数を使用してソースクラスターで認証します。詳細については、「シングルコマンドモード」を参照してください。送信先クラスターの認証情報を提供するには、DESTINATION_CLOUDHSM_PIN と DESTINATION_CLOUDHSM_ROLE の 2 つの追加の環境変数を設定する必要があります。

```
$ export DESTINATION_CLOUDHSM_ROLE=crypto-user
```

```
$ export DESTINATION_CLOUDHSM_PIN=username:password
```

  - インタラクティブモードでは、ユーザーは送信元クラスターと送信先クラスターの両方に明示的にログインする必要があります。

構文

```
aws-cloudhsm > help key replicate
Replicate a key from a source to a destination cluster

Usage: key replicate --filter [<FILTER>...] --source-cluster-id <SOURCE_CLUSTER_ID> --
destination-cluster-id <DESTINATION_CLUSTER_ID>

Options:
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select
 matching key on the source cluster
      --source-cluster-id <SOURCE_CLUSTER_ID>
          Source cluster ID
      --destination-cluster-id <DESTINATION_CLUSTER_ID>
          Destination cluster ID
  -h, --help
          Print help
```

例

Example 例: キーのレプリケート

このコマンドは、 を使用してソースクラスターからクローンの宛先クラスターにキーをレプリケートします。

```
crypto-user-1@cluster-1234abcdefg > key replicate \
      --filter attr.label=example-key \
      --source-cluster-id cluster-1234abcdefg \
      --destination-cluster-id cluster-2345bcdefgh
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x0000000000300006",
      "key-info": {
        "key-owners": [
          {
            "username": "crypto-user-1",
            "key-coverage": "full"
          }
        ],
```

```
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "example-key",
        "id": "0x",
        "check-value": "0x5e118e",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": true,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    },
    "message": "Successfully replicated key"
  }
}
```

引数

#######

ソースクラスターで一致するキーを選
択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするための、キーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

サポートされている CloudHSM CLI キー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください

必須: はい

<SOURCE_CLUSTER_ID>

ソースクラスター ID。

必須: はい

<DESTINATION_CLUSTER_ID>

送信先クラスター ID。

必須: はい

関連トピック

- CloudHSM CLI を使用した複数のクラスターへの接続

key set-attribute

key set-attribute コマンドを使用して、 AWS CloudHSM クラスター内のキーの属性を設定します。キーの属性を変更できるのは、キーを作成し、その結果キーを所有する CU のみです。

CloudHSM CLI で使用できる主な属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- このコマンドを実行できるのは Crypto User (CU) です。
- 管理者は信頼できる属性を設定できます。

要件

このコマンドを実行するには、CU としてログインする必要があります。信頼できる属性を設定するには、管理者ユーザーとしてログインする必要があります。

Syntax

```
aws-cloudhsm > help key set-attribute
Set an attribute for a key in the HSM cluster

Usage: cloudhsm-cli key set-attribute [OPTIONS] --filter [<FILTER>...] --
name <KEY_ATTRIBUTE> --value <KEY_ATTRIBUTE_VALUE>

Options:
      --cluster-id <CLUSTER_ID>         Unique Id to choose which of the clusters in
 the config file to run the operation against. If not provided, will fall back to the
 value provided when interactive mode was started, or error
      --filter [<FILTER>...]            Key reference (e.g. key-
reference=0xabc) or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key to modify
      --name <KEY_ATTRIBUTE>            Name of attribute to be set
      --value <KEY_ATTRIBUTE_VALUE>...  Attribute value to be set
  -h, --help                            Print help
```

例: キー属性の設定

次の例は、key set-attribute コマンドを使用してラベルを設定する方法を示しています。

Example

1. 次に示すように、ラベル my_key の付いたキーを使用してください。

```
aws-cloudhsm > key set-attribute --filter attr.label=my_key --name encrypt --value
 false
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

2. key list コマンドを使用して、encrypt 属性が変更されたことを確認します。

```
aws-cloudhsm > key list --filter attr.label=my_key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
```

```
      {
        "key-reference": "0x00000000006400ec",
        "key-info": {
          "key-owners": [
            {
              "username": "bob",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "my_key",
          "id": "",
          "check-value": "0x6bd9f7",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": true,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": true,
          "unwrap": true,
          "verify": true,
          "wrap": true,
          "wrap-with-trusted": false,
          "key-length-bytes": 32
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
}
```

```
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<KEY_ATTRIBUTE>

キーの属性の名前を指定します。

必須: はい

****

キーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`を使用して、一致するキーを選択して削除します。

サポートされている CloudHSM CLI キー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください

必須: いいえ

<KEY_ATTRIBUTE_VALUE>

キーの属性の値を指定します。

必須: はい

<KEY_REFERENCE>

キーの 16 進数または 10 進数表現。 (キーハンドルなど)。

必須: いいえ

関連トピック

- CloudHSM CLI を使用してキーをフィルタリングする
- CloudHSM CLI のキー属性

キーシェア

key share コマンドは、 AWS CloudHSM クラスター内の他の CUs とキーを共有します。

キーを共有できるのは、キーを作成し、その結果キーを所有する CU のみです。キーを共有しているユーザーは、暗号化オペレーションでキーを使用できますが、キーを削除、エクスポート、共有、または共有解除することはできません。さらに、これらのユーザーは キー属性 を変更できません。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key share
Share a key in the HSM cluster with another user

Usage: key share --filter [<FILTER>...] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for sharing

      --username <USERNAME>
          A username with which the key will be shared

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
```

```
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

  -h, --help
          Print help (see a summary with '-h')
```

例: キーを別の CU と共有する

以下の例は、key share コマンドを使用して CU `alice` とキーを共有する方法を示します。

Example

1. key share コマンドを実行して `alice` とキーを共有します。

```
aws-cloudhsm > key share --filter attr.label="rsa_key_to_share" attr.class=private-
key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key shared successfully"
  }
}
```

2. key list コマンドを実行します。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-
key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
```

```
            "key-coverage": "full"
          },
          {
            "username": "cu1",
            "key-coverage": "full"
          },
          {
            "username": "cu4",
            "key-coverage": "full"
          },
          {
            "username": "cu5",
            "key-coverage": "full"
          },
          {
            "username": "cu6",
            "key-coverage": "full"
          },
          {
            "username": "cu7",
            "key-coverage": "full"
          },
          {
            "username": "alice",
            "key-coverage": "full"
          }
        ],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa_key_to_share",
        "id": "",
        "check-value": "0xae8ff0",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
```

```
            "never-extractable": false,
            "private": true,
            "sensitive": true,
            "sign": true,
            "trusted": false,
            "unwrap": true,
            "verify": false,
            "wrap": false,
            "wrap-with-trusted": false,
            "key-length-bytes": 1219,
            "public-exponent": "0x010001",
            "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
            "modulus-size-bits": 2048
          }
        }
      ],
      "total_key_count": 1,
      "returned_key_count": 1
    }
}
```


3. 上記のリストで、alice が shared-users のリストに含まれている事を検証します

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

キーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリストattr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEを使用して、一致するキーを選択して削除します。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: はい

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。このコマンドではユーザー名の大文字と小文字は区別されません。ユーザー名は常に小文字で表示されます。

必須: はい

<ROLE>

このユーザーに割り当てられるロールを指定します。このパラメータは必須です。ユーザーのロールを取得するには、ユーザーリストコマンドを使用します。HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

必須: はい

関連トピック

- CloudHSM CLI を使用してキーをフィルタリングする
- CloudHSM CLI のキー属性

キー共有解除

key unshare コマンドは、 AWS CloudHSM クラスター内の他の CUs とキーの共有を解除します。

キーを共有解除できるのは、キーを作成し、その結果キーを所有する CU のみです。キーを共有しているユーザーは、暗号化オペレーションでキーを使用できますが、キーを削除、エクスポート、共有、または共有解除することはできません。さらに、これらのユーザーは キー属性 を変更できません。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key unshare
Unshare a key in the HSM cluster with another user

Usage: key unshare --filter [<FILTER>...] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for unsharing

      --username <USERNAME>
          A username with which the key will be unshared

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

  -h, --help
          Print help (see a summary with '-h')
```

例: 別の CU とのキーの共有を解除する

以下の例は、key unshare コマンドを使用して CU `alice` とのキーの共有を解除する方法を示しています。

Example

1. key list コマンドを実行し、`alice` と共有を解除したい特定のキーでフィルタリングします。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-
key --verbose
{
```

```
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
            {
              "username": "alice",
              "key-coverage": "full"
            }
          ],
          "cluster-coverage": "full"
        },
```

```
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

2. shared-users 出力に alice が含まれていることを確認し、次の key unshare コマンドを実行して alice とのキーの共有を解除します。

```
aws-cloudhsm > key unshare --filter attr.label="rsa_key_to_share"
 attr.class=private-key --username alice --role crypto-user
{
```

```
  "error_code": 0,
  "data": {
    "message": "Key unshared successfully"
  }
}
```

3. key list コマンドをもう一度実行して、alice とキーが共有解除されたことを確認します。

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-
key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
```

```
            {
              "username": "cu7",
              "key-coverage": "full"
            },
          ],
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

キーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`を使用して、一致するキーを選択して削除します。

サポートされているキー属性のリストについては、「CloudHSM CLI のキー属性」を参照してください。

必須: はい

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。このコマンドではユーザー名の大文字と小文字は区別されません。ユーザー名は常に小文字で表示されます。

必須: はい

<ROLE>

このユーザーに割り当てられるロールを指定します。このパラメータは必須です。ユーザーのロールを取得するには、ユーザーリストコマンドを使用します。HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

必須: はい

関連トピック

- CloudHSM CLI を使用してキーをフィルタリングする
- CloudHSM CLI のキー属性

キーアンラップ

CloudHSM CLI のkey unwrap親コマンドは、暗号化された (ラップされた) 対称または非対称プライベートキーをファイルから HSM にインポートします。このコマンドは、 キーラップ コマンドでラップされた暗号化キーをインポートするように設計されていますが、他のツールでラップされたキーをアンラップするためにも使用できます。ただし、このような場合は、PKCS#11 または JCE ソフトウェアライブラリを使用して、キーをラップ解除することをお勧めします。

- aes-gcm のキーアンラップ
- キーアンラップ aes-no-pad
- aes-pkcs5-pad のキーアンラップ
- キーアンラップ aes-zero-pad
- キーアンラップ cloudhsm-aes-gcm
- キーが rsa-aes をラップ解除する
- rsa-oaep のキーアンラップ
- rsa-pkcs のキーアンラップ

aes-gcm のキーアンラップ

key unwrap aes-gcm このコマンドは、AES ラッピングキーとアンラップメカニズムを使用して、ペイロードキーをクラスターにAES-GCMアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap aes-gcm コマンドを使用するには、 AWS CloudHSM クラスターに AES ラッピングキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap aes-gcm
Usage: key unwrap aes-gcm [OPTIONS] --filter [<FILTER>...] --tag-length-
bits <TAG_LENGTH_BITS> --key-type-class <KEY_TYPE_CLASS> --label <LABEL> --iv <IV> <--
data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --iv <IV>
          Initial value used to wrap the key, in hex
  -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して AES キーを使用して key unwrap aes-gcm コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --iv
 0xf90613bb8e337ec0339aad21 --data xvslgrtg8kHzrvekny97tLSIeokpPwV8
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e4",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
```

```
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --iv
 0xf90613bb8e337ec0339aad21 --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e4",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
```

```
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<AAD>

Aes GCM Additional Authenticated Data (AAD) 値を 16 進数で表します。

必須: いいえ

<TAG_LENGTH_BITS>

Aes GCM タグの長さをビット単位で指定します。

必須: はい

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

<IV>

キーを 16 進数でラップするために使用される初期値。

必須: いいえ

関連トピック

- キーラップ

- キーアンラップ

キーアンラップ aes-no-pad

key unwrap aes-no-pad このコマンドは、AES ラッピングキーとアンラップメカニズムを使用して、ペイロードキーをクラスターにAES-NO-PADアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap aes-no-pad コマンドを使用するには、 AWS CloudHSM クラスターに AES ラッピングキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap aes-no-pad
Usage: key unwrap aes-no-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
```

```
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
be recovered after the session ends
 -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して AES キーを使用して key unwrap aes-no-pad コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-no-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data eXK3PMAOnKM9y3YX6brbhtMoC060EOH9
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ec",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
```

```
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-no-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ec",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
```

```
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

ラップされたキーの 形式のキー属性KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEのスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

aes-pkcs5-pad のキーアンラップ

key unwrap aes-pkcs5-pad コマンドは、AES ラッピングキーとアンラップメカニズムを使用してペイロードキーをAES-PKCS5-PADアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap aes-pkcs5-pad コマンドを使用するには、 AWS CloudHSM クラスターに AES ラッピングキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap aes-pkcs5-pad
Usage: key unwrap aes-pkcs5-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
```

```
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
be recovered after the session ends
 -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して AES キーを使用して key unwrap aes-pkcs5-pad コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-pkcs5-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data MbuYNresfOKyGNnxKWen88nSfX+uUE/0qmGofSisicY=
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
```

```
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-pkcs5-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
```

```
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

キーアンラップ aes-zero-pad

key unwrap aes-zero-pad このコマンドは、AES ラッピングキーとアンラップメカニズムを使用して、ペイロードキーをクラスターにAES-ZERO-PADアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap aes-no-pad コマンドを使用するには、 AWS CloudHSM クラスターに AES ラッピングキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap aes-zero-pad
Usage: key unwrap aes-zero-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
```

```
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
be recovered after the session ends
 -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定した AES キーを使用して key unwrap aes-zero-pad コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-zero-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data L1wVlL/YeBNVAw6Mpk3owFJZXBzDLONt
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
```

```
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap aes-zero-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
```

```
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

キーアンラップ cloudhsm-aes-gcm

key unwrap cloudhsm-aes-gcm このコマンドは、AES ラッピングキーとアンラップメカニズムを使用して、ペイロードキーをクラスターにCLOUDHSM-AES-GCMアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap cloudhsm-aes-gcm コマンドを使用するには、 AWS CloudHSM クラスターに AES ラッピングキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap cloudhsm-aes-gcm
Usage: key unwrap cloudhsm-aes-gcm [OPTIONS] --filter [<FILTER>...] --tag-length-
bits <TAG_LENGTH_BITS> --key-type-class <KEY_TYPE_CLASS> --label <LABEL> <--data-
path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
```

```
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
  -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して AES キーを使用して key unwrap cloudhsm-aes-gcm コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap cloudhsm-aes-gcm --key-type-class aes --label aes-
unwrapped --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --data
 6Rn8nkjEriDYlnP3P8nPkYQ8hplOEJ899zsrF+aTB0i/fIlZ
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x0000000000001408e8",
      "key-info": {
        "key-owners": [
```

```
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap cloudhsm-aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --data-path payload-
key.pem
```

```
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001408e8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
```

```
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

****

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<AAD>

Aes GCM Additional Authenticated Data (AAD) 値を 16 進数で表します。

必須: いいえ

<TAG_LENGTH_BITS>

Aes GCM タグの長さをビット単位で指定します。

必須: はい

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

キーが rsa-aes をラップ解除する

key unwrap rsa-aes コマンドは、RSA プライベートキーとアンラップメカニズムを使用してペイロードキーをRSA-AESアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

を使用するにはkey unwrap rsa-aes、 AWS CloudHSM クラスターに RSA パブリックラッピングキーの RSA プライベートキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key unwrap rsa-aes
Usage: key unwrap rsa-aes [OPTIONS] --filter [<FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --key-type-class <KEY_TYPE_CLASS> --label <LABEL>
 <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
```

```
  -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して RSA プライベートキーを使用して key unwrap rsa-aes コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-aes --key-type-class aes --label aes-unwrapped
 --filter attr.label=rsa-private-key-example --hash-function sha256 --
mgf mgf1-sha256 --data HrSE1DEyLjIeyGdPa9R+ebiqB5TIJGyamPker31ZebPwRA
+NcerbAJO8DJ1lXPygZcI21vIFSZJuWMEiWpe1R9D/5WSYgxLVKex30xCFqebtEzxbKuv4DOmU4meSofqREYvtb3EoIKwjy
+RL5WGXKe4nAboAkC5GO7veI5yHL1SaKlssSJtTL/CFpbSLsAFuYbv/NUCWwMY5mwyVTCSlw+HlgKK
+5TH1MzBaSi8fpfyepLT8sHy2Q/VRl6ifb49p6m0KQFbRVvz/OWUd6l4d97BdgtaEz6ueg==
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e2",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
```

```
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-aes --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-key-example --hash-function sha256 --mgf mgf1-sha256 --data-
path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e2",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
```

```
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<MGF>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

rsa-oaep のキーアンラップ

key unwrap rsa-oaep コマンドは、RSA プライベートキーとアンラップメカニズムを使用してペイロードキーをRSA-OAEPアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap rsa-oaep コマンドを使用するには、 AWS CloudHSM クラスターに RSA パブリックラッピングキーの RSA プライベートキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap rsa-oaep
Usage: key unwrap rsa-oaep [OPTIONS] --filter [<FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --key-type-class <KEY_TYPE_CLASS> --label <LABEL>
 <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <<DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
```

```
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
  -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して RSA プライベートキーを使用して key unwrap rsa-oaep コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-oaep --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-example-key --hash-function sha256 --mgf mgf1-sha256 --data
 OjJe4msobPLz9TuSAdULEu17T5rMDWtSlLyBSkLbaZnYzzpdrhsbGLbwZJCtB/jGkDNdB4qyTAOQwEpggGf6v
+Yx6JcesNeKKNU8XZal/YBoHC8noTGUSDI2qr+u2tDc84NPv6d+F2KOONXsSxMhmxzzNG/
gzTVIJhOuy/B1yHjGP4mOXoDZf5+7f5M1CjxBmz4Vva/wrWHGCSG0yOaWblEvOiHAIt3UBdyKmU+/
My4xjfJv7WGGu3DFUUIZ06TihRtKQhUYU1M9u6NPf9riJJfHsk6QCuSZ9yWThDT9as6i7e3htnyDhIhGWaoK8JU855cN/
YNKAUqkNpC4FPL3iw==
{
  "data": {
    "key": {
      "key-reference": "0x00000000001808e9",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
```

```
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-oaep --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-example-key --hash-function sha256 --mgf mgf1-sha256 --data-
path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e9",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
```

```
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

#######

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE ラップされたキーの 形式のキー属性のスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<HASH_FUNCTION>

ハッシュ関数を指定します。

有効値:

- sha1
- sha224
- sha256
- sha384
- sha512

必須: はい

<MGF>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

rsa-pkcs のキーアンラップ

key unwrap rsa-pkcs コマンドは、RSA プライベートキーとアンラップメカニズムを使用してペイロードキーをRSA-PKCSアンラップします。

ラップされていないキーは、 によって生成されたキーと同じ方法で使用できます AWS CloudHSM。ローカルで生成されなかったことを示すために、そのlocal属性は に設定されますfalse。

key unwrap rsa-pkcs コマンドを使用するには、 AWS CloudHSM クラスターに RSA パブリックラッピングキーの RSA プライベートキーがあり、そのunwrap属性を に設定する必要がありますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

構文

```
aws-cloudhsm > help key unwrap rsa-pkcs
Usage: key unwrap rsa-pkcs [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
```

```
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
  -h, --help
          Print help
```

例

これらの例は、unwrap属性値を に設定して AES キーを使用して key unwrap rsa-oaep コマンドを使用する方法を示していますtrue。

Example 例: Base64 でエンコードされたラップされたキーデータからペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-pkcs --key-type-class aes --label
 aes-unwrapped --filter attr.label=rsa-private-key-example --data
 am0Nc7+YE8FWs+5HvU7sIBcXVb24QA0l65nbNAD+1bK+e18BpSfnaI3P+r8Dp+pLu1ofoUy/
vtzRjZoCiDofcz4EqCFnGl4GdcJ1/3W/5WRvMatCa2d7cx02swaeZcjKsermPXYRO1lGlfq6NskwMeeTkV8R7Rx9artFrs1
c3XdFJ2+0Bo94c6og/
yfPcpOOobJlITCoXhtMRepSdO4OggYq/6nUDuHCtJ86pPGnNahyr7+sAaSI3a5ECQLUjwaIARUCyoRh7EFK3qPXcg==
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ef",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
```

```
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example 例: データパスを介して提供されたペイロードキーをアンラップする

```
aws-cloudhsm > key unwrap rsa-pkcs --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-key-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ef",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
```

```
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

****

ラップ解除attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするキーを選択する の形式のキー属性のキーリファレンス (例: key-reference=0xabc) またはスペース区切りリスト。

必須: はい

<DATA_PATH>

ラップされたキーデータを含むバイナリファイルへのパス。

必須: はい (Base64 でエンコードされたデータを通じて提供される場合を除く)

<DATA>

Base64 でエンコードされたラップされたキーデータ。

必須: はい (データパスで指定される場合を除く)

####

ラップされたキーの 形式のキー属性KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEのスペース区切りリスト。

必須: いいえ

<KEY_TYPE_CLASS>

ラップされたキーのキータイプとクラス [指定できる値: aes、des3、ec-private、generic-secret、rsa-private〕。

必須: はい

<LABEL>

ラップされていないキーのラベル。

必須: はい

<SESSION>

現在のセッションにのみ存在するセッションキーを作成します。セッション終了後、キーをリカバリすることはできません。

必須: いいえ

関連トピック

- キーラップ
- キーアンラップ

キーラップ

CloudHSM CLI の key wrap コマンドは、対称または非対称プライベートキーの暗号化されたコピーを HSM からファイルにエクスポートします。を実行するときはkey wrap、エクスポートするキーと

出力ファイルの 2 つのことを指定します。エクスポートするキーは、エクスポートするキーを暗号化 (ラップ) する HSM 上のキーです。

key wrap コマンドは、HSM からキーを削除したり、暗号化オペレーションで使用したりしません。同じキーを複数回エクスポートできます。暗号化されたキーを HSM にインポートし直すには、 を使用しますキーアンラップ。キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでのみキーを使用できます。

key wrap コマンドは、次のサブコマンドで構成されます。

- キーラップ aes-gcm
- キーラップ aes-no-pad
- キーラップ aes-pkcs5-pad
- キーラップ aes-zero-pad
- キーラップ cloudhsm-aes-gcm
- キーラップ rsa-aes
- キーラップ rsa-oaep
- キーラップ rsa-pkcs

キーラップ aes-gcm

key wrap aes-gcm コマンドは、HSM の AES キーとラッピングメカニズムを使用してペイロードキーをAES-GCMラップします。ペイロードキーの `extractable` 属性は に設定する必要がありますtrue。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap aes-gcm コマンドを使用するには、まず AWS CloudHSM クラスターに AES キーが必要です。key generate-symmetric aes コマンドと `wrap` 属性を に設定してラップするための AES キーを生成できますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap aes-gcm
Usage: key wrap aes-gcm [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...] --tag-length-bits <TAG_LENGTH_BITS>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
  -h, --help
          Print help
```

例

この例では、AES キーを使用して key wrap aes-gcm コマンドを使用する方法を示します。

Example

```
aws-cloudhsm > key wrap aes-gcm --payload-filter attr.label=payload-key --wrapping-
filter attr.label=aes-example --tag-length-bits 64  --aad 0x10
{
  "error_code": 0,
```

```
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "iv": "0xf90613bb8e337ec0339aad21",
    "wrapped_key_data": "xvslgrtg8kHzrvekny97tLSIeokpPwV8"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

<AAD>

AES GCM Additional Authenticated Data (AAD) 値を 16 進数で表します。

必須: いいえ

<TAG_LENGTH_BITS>

AES GCM タグの長さをビット単位で指定します。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ aes-no-pad

key wrap aes-no-pad コマンドは、HSM の AES キーとラッピングメカニズムを使用してペイロードキーをAES-NO-PADラップします。ペイロードキーの extractable 属性は に設定する必要がありますtrue。

キーを作成した暗号ユーザー (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap aes-no-pad コマンドを使用するには、まず AWS CloudHSM クラスターに AES キーが必要です。key generate-symmetric aes コマンドと wrap 属性を に設定することで、ラップ用の AES キーを生成できますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap aes-no-pad
Usage: key wrap aes-no-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
```

```
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
  -h, --help
          Print help
```

例

この例では、wrap属性値を に設定して AES キーを使用して key wrap aes-no-pad コマンドを使用する方法を示しますtrue。

Example

```
aws-cloudhsm > key wrap aes-no-pad --payload-filter attr.label=payload-key --wrapping-
filter attr.label=aes-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "eXK3PMAOnKM9y3YX6brbhtMoC060EOH9"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ aes-pkcs5-pad

key wrap aes-pkcs5-pad コマンドは、HSM の AES キーとラッピングメカニズムを使用してペイロードキーを`AES-PKCS5-PAD`ラップします。ペイロードキーの `extractable` 属性は に設定する必要があります`true`。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap aes-pkcs5-pad コマンドを使用するには、まず AWS CloudHSM クラスターに AES キーが必要です。key generate-symmetric aes コマンドと `wrap` 属性を に設定することで、ラップ用の AES キーを生成できます`true`。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap aes-pkcs5-pad
Usage: key wrap aes-pkcs5-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
  -h, --help
          Print help
```

例

この例では、wrap属性値を に設定して AES キーを使用して key wrap aes-pkcs5-pad コマンドを使用する方法を示しますtrue。

Example

```
aws-cloudhsm > key wrap aes-pkcs5-pad --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "MbuYNresfOKyGNnxKWen88nSfX+uUE/0qmGofSisicY="
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ aes-zero-pad

key wrap aes-zero-pad コマンドは、HSM の AES キーとラッピングメカニズムを使用してペイロードキーを`AES-ZERO-PAD`ラップします。ペイロードキーの `extractable` 属性は に設定する必要があります`true`。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap aes-zero-pad コマンドを使用するには、まず AWS CloudHSM クラスターに AES キーが必要です。`wrap` 属性を `true` に設定して key generate-symmetric aes コマンドを使用して、ラップ用の AES キーを生成できます。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap aes-zero-pad
Usage: key wrap aes-zero-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
  -h, --help
          Print help
```

例

この例では、wrap属性値を に設定して AES キーを使用して key wrap aes-zero-pad コマンドを使用する方法を示しますtrue。

Example

```
aws-cloudhsm > key wrap aes-zero-pad --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "L1wVlL/YeBNVAw6Mpk3owFJZXBzDLONt"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ cloudhsm-aes-gcm

key wrap cloudhsm-aes-gcm コマンドは、HSM の AES キーとラCLOUDHSM-AES-GCMッピングメカニズムを使用してペイロードキーをラップします。ペイロードキーの extractable 属性は に設定する必要がありますtrue。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap cloudhsm-aes-gcm コマンドを使用するには、まず AWS CloudHSM クラスターに AES キーが必要です。key generate-symmetric aes コマンドと wrap 属性を に設定してラップするための AES キーを生成できますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap cloudhsm-aes-gcm
Usage: key wrap cloudhsm-aes-gcm [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...] --tag-length-bits <TAG_LENGTH_BITS>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
```

```
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
  attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
  payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
  attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
  wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
  -h, --help
          Print help
```

例

この例では、AES キーを使用して key wrap cloudhsm-aes-gcm コマンドを使用する方法を示します。

Example

```
aws-cloudhsm > key wrap cloudhsm-aes-gcm --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example --tag-length-bits 64 --aad 0x10
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "6Rn8nkjEriDYlnP3P8nPkYQ8hplOEJ899zsrF+aTB0i/fIlZ"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<AAD>

AES GCM Additional Authenticated Data (AAD) 値を 16 進数で表します。

必須: いいえ

<TAG_LENGTH_BITS>

AES GCM タグの長さをビット単位で指定します。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ rsa-aes

key wrap rsa-aes コマンドは、HSM の RSA パブリックキーと RSA-AES ラップメカニズムを使用してペイロードキーをラップします。ペイロードキーの extractable 属性は に設定する必要がありますtrue。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap rsa-aes コマンドを使用するには、まず AWS CloudHSM クラスターに RSA キーが必要です。キー generate-asymmetric-pair コマンドと `wrap` 属性を に設定して、RSA キーペアを生成できますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap rsa-aes
Usage: key wrap rsa-aes [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...] --hash-function <HASH_FUNCTION> --mgf <MGF>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
```

```
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
  -h, --help
          Print help
```

例

この例では、wrap属性値が に設定された RSA パブリックキーを使用して key wrap rsa-ae コマンドを使用する方法を示しますtrue。

Example

```
aws-cloudhsm > key wrap rsa-aes --payload-filter attr.label=payload-key --wrapping-
filter attr.label=rsa-public-key-example --hash-function sha256 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "payload-key-reference": "0x00000000001c08f1",
    "wrapping-key-reference": "0x00000000007008da",
    "wrapped-key-data": "HrSE1DEyLjIeyGdPa9R+ebiqB5TIJGyamPker31ZebPwRA
+NcerbAJO8DJ1lXPygZcI21vIFSZJuWMEiWpe1R9D/5WSYgxLVKex30xCFqebtEzxbKuv4DOmU4meSofqREYvtb3EoIKwjy
+RL5WGXKe4nAboAkC5GO7veI5yHL1SaKlssSJtTL/CFpbSLsAFuYbv/NUCWwMY5mwyVTCSlw+HlgKK
+5TH1MzBaSi8fpfyepLT8sHy2Q/VRl6ifb49p6m0KQFbRVvz/OWUd6l4d97BdgtaEz6ueg=="
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<MGF>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ rsa-oaep

key wrap rsa-oaep コマンドは、HSM の RSA パブリックキーとラッピングメカニズムを使用してペイロードキーをRSA-OAEPラップします。ペイロードキーの extractable 属性は に設定する必要がありますtrue。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap rsa-oaep コマンドを使用するには、まず AWS CloudHSM クラスターに RSA キーが必要です。キー generate-asymmetric-pair コマンドと `wrap` 属性を に設定して、RSA キーペアを生成できます`true`。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap rsa-oaep
Usage: key wrap rsa-oaep [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...] --hash-function <HASH_FUNCTION> --mgf <MGF>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
```

```
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
  -h, --help
          Print help
```

例

この例では、wrap属性値が に設定された RSA パブリックキーを使用して key wrap rsa-oaep コマンドを使用する方法を示しますtrue。

Example

```
aws-cloudhsm > key wrap rsa-oaep --payload-filter attr.label=payload-key --wrapping-
filter attr.label=rsa-public-key-example --hash-function sha256 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "payload-key-reference": "0x00000000001c08f1",
    "wrapping-key-reference": "0x00000000007008da",
    "wrapped-key-data": "OjJe4msobPLz9TuSAdULEu17T5rMDWtSlLyBSkLbaZnYzzpdrhsbGLbwZJCtB/
jGkDNdB4qyTAOQwEpggGf6v+Yx6JcesNeKKNU8XZal/YBoHC8noTGUSDI2qr+u2tDc84NPv6d
+F2KOONXsSxMhmxzzNG/gzTVIJhOuy/B1yHjGP4mOXoDZf5+7f5M1CjxBmz4Vva/
wrWHGCSG0yOaWblEvOiHAIt3UBdyKmU+/
My4xjfJv7WGGu3DFUUIZ06TihRtKQhUYU1M9u6NPf9riJJfHsk6QCuSZ9yWThDT9as6i7e3htnyDhIhGWaoK8JU855cN/
YNKAUqkNpC4FPL3iw=="
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

<MGF>

マスク生成関数を指定します。

> **Note**
>
> マスク生成関数のハッシュ関数は、署名メカニズムのハッシュ関数と一致する必要があります。

有効値

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

必須: はい

関連トピック

- キーラップ
- キーアンラップ

キーラップ rsa-pkcs

key wrap rsa-pkcs コマンドは、HSM の RSA パブリックキーとラッピングメカニズムを使用してペイロードキーをRSA-PKCSラップします。ペイロードキーの `extractable` 属性は に設定する必要がありますtrue。

キーを作成した Crypto User (CU) であるキーの所有者のみがキーをラップできます。キーを共有するユーザーは、暗号化オペレーションでキーを使用できます。

key wrap rsa-pkcs コマンドを使用するには、まず AWS CloudHSM クラスターに RSA キーが必要です。キー generate-asymmetric-pair コマンドと `wrap` 属性を に設定して、RSA キーペアを生成できますtrue。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

要件

- このコマンドを実行するには、CU としてログインする必要があります。

Syntax

```
aws-cloudhsm > help key wrap rsa-pkcs
Usage: key wrap rsa-pkcs [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
```

```
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
  -h, --help
          Print help
```

例

この例では、RSA パブリックキーを使用して key wrap rsa-pkcs コマンドを使用する方法を示します。

Example

```
aws-cloudhsm > key wrap rsa-pkcs --payload-filter attr.label=payload-key --wrapping-
filter attr.label=rsa-public-key-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000007008da",
    "wrapped_key_data": "am0Nc7+YE8FWs+5HvU7sIBcXVb24QA0l65nbNAD+1bK+e18BpSfnaI3P+r8Dp
+pLu1ofoUy/
vtzRjZoCiDofcz4EqCFnGl4GdcJ1/3W/5WRvMatCa2d7cx02swaeZcjKsermPXYRO1lGlfq6NskwMeeTkV8R7Rx9artFrs1
c3XdFJ2+0Bo94c6og/
yfPcpOOobJlITCoXhtMRepSdO4OggYq/6nUDuHCtJ86pPGnNahyr7+sAaSI3a5ECQLUjwaIARUCyoRh7EFK3qPXcg=="
  }
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PAYLOAD_FILTER>

ペイロードキーを選択attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUEするためのキーリファレンス (例: key-reference=0xabc) または 形式のキー属性のスペース区切りリスト。

必須: はい

<PATH>

ラップされたキーデータが保存されるバイナリファイルへのパス。

必須: いいえ

<WRAPPING_FILTER>

ラッピングキーを選択`attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`するためのキーリファレンス (例: `key-reference=0xabc`) または 形式のキー属性のスペース区切りリスト。

必須: はい

関連トピック

- キーラップ
- キーアンラップ

login (ログイン)

クラスターの各 HSM でログインおよびログアウトを行うには、CloudHSM CLI の login コマンドを使用します。

> **Note**
>
> ログイン試行回数が 5 回を超えると、アカウントがロックアウトされます。アカウントのロックを解除するには、管理者は、cloudhsm_cli の user change-password コマンドを使用してパスワードをリセットする必要があります。

ログインとログアウトのトラブルシューティングを行うには

クラスター内に複数の HSM がある場合は、アカウントがロックアウトされるまでのログイン試行回数の上限が増える可能性があります。これは、CloudHSM クライアントがさまざまな HSM 間で負荷を分散するためです。したがって、ログイン試行は毎回同じ HSM で開始されない場合があります。この機能をテストしている場合は、アクティブな HSM が1つだけのクラスターでテストすることをお勧めします。

2018 年 2 月より前にクラスターを作成した場合、ロックアウトされるまでのログイン試行回数は 20 回です。

ユーザーのタイプ

これらのコマンドは、次のユーザーが実行できます。

- 非アクティブ管理者
- 管理
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help login
Login to your cluster

USAGE:
    cloudhsm-cli login [OPTIONS] --username <USERNAME> --role <ROLE> [COMMAND]

Commands:
  mfa-token-sign  Login with token-sign mfa
  help            Print this message or the help of the given subcommand(s)

OPTIONS:
        --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

        --username <USERNAME>
            Username to access the Cluster

        --role <ROLE>
            Role the user has in the Cluster

           Possible values:
           - crypto-user: A CryptoUser has the ability to manage and use keys
           - admin:       An Admin has the ability to manage user accounts

        --password <PASSWORD>
           Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

  -h, --help
          Print help (see a summary with '-h')
```

例

Example

このコマンドは、admin1 という名前の管理者ユーザーの認証情報を使用して、クラスター内のすべての HSM にログインします。

```
aws-cloudhsm > login --username admin1 --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。このコマンドではユーザー名の大文字と小文字は区別されません。ユーザー名は常に小文字で表示されます。

必須: はい

<ROLE>

このユーザーに割り当てられるロールを指定します。このパラメータは必須です。有効な値は admin、crypto-user です。

ユーザーのロールを取得するには、user list コマンドを使用します。HSM のユーザー タイプの詳細については、「HSM ユーザーについて」を参照してください。

<PASSWORD>

HSM にログインしているユーザーのパスワードを指定します。

関連トピック

- CloudHSM CLI の使用開始
- クラスターのアクティブ化

ログイン mfa-token-sign

AWS CloudHSM CloudHSM CLI の login mfa-token-sign コマンドを使用し、多要素認証を使用してログインします。このコマンドを使用するには、まず CloudHSM CLI の MFA を設定する必要があります。

ユーザーのタイプ

これらのコマンドは、次のユーザーが実行できます。

- 管理
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help login mfa-token-sign
Login with token-sign mfa

USAGE:
    login --username <USERNAME> --role <ROLE> mfa-token-sign --token <TOKEN>

OPTIONS:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --token <TOKEN>            Filepath where the unsigned token file will be written
  -h, --help                     Print help
```

例

Example

```
aws-cloudhsm > login --username test_user --role admin mfa-token-sign --token /home/
valid.token
```

```
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
{
  "error_code": 0,
  "data": {
    "username": "test_user",
    "role": "admin"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<TOKEN>

署名されていないトークンファイルが書き込まれるファイルパス。

必須: はい

関連トピック

- CloudHSM CLI の使用開始
- クラスターのアクティブ化
- CloudHSM CLI を使用して MFA を管理する

ログアウト

クラスターの各 HSM でログアウトするには、CloudHSM CLI の logout コマンドを使用します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help logout
Logout of your cluster

USAGE:
    logout

OPTIONS:
        --cluster-id <CLUSTER_ID> Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
    -h, --help                    Print help information
    -V, --version                 Print version information
```

例

Example

このコマンドはクラスターの HSM からログアウトします。

```
aws-cloudhsm > logout
{
  "error_code": 0,
  "data": "Logout successful"
}
```

関連トピック

- CloudHSM CLI の使用開始
- クラスターのアクティブ化

ユーザー

user はコマンドグループの親カテゴリであり、親カテゴリと組み合わせるとユーザー固有のコマンドが作成されます。現在、このユーザーカテゴリは次のコマンドで構成されています。

- user change-mfa
- user change-password
- user create

- user delete
- user list

user change-mfa

現在、このカテゴリは次のサブコマンドで構成されています。

- user change-mfa token-sign

user change-mfa token-sign

CloudHSM CLI の user change-mfa コマンドを使用して、ユーザーアカウントの多要素認証 (MFA) 設定を更新します。このコマンドは、どのユーザーアカウントでも実行できます。Admin ロールを持つアカウントは、他のユーザーに対してこのコマンドを実行できます。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理
- Crypto User

Syntax

現在、ユーザーが利用できる多要素戦略は「トークン署名」だけです。

```
aws-cloudhsm > help user change-mfa
Change a user's Mfa Strategy

Usage:
    user change-mfa <COMMAND>

Commands:
  token-sign  Register or Deregister a public key using token-sign mfa strategy
  help        Print this message or the help of the given subcommand(s)
```

トークン署名戦略では、署名されていないトークンを書き込むためのトークンファイルを要求します。

```
aws-cloudhsm > help user change-mfa token-sign
Register or Deregister a public key using token-sign mfa strategy

Usage: user change-mfa token-sign [OPTIONS] --username <USERNAME> --role <ROLE> <--
token <TOKEN>|--deregister>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username of the user that will be modified

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --change-password <CHANGE_PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

      --token <TOKEN>
          Filepath where the unsigned token file will be written. Required for enabling
 MFA for a user

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

      --deregister
          Deregister the MFA public key, if present

      --change-quorum
          Change the Quorum public key along with the MFA key

  -h, --help
          Print help (see a summary with '-h')
```

例

このコマンドは、クラスター内の HSM ごとに 1 つの未署名トークンを token で指定されたファイルに書き込みます。プロンプトが表示されたら、ファイル内のトークンに署名します。

Example : クラスターの HSM ごとに 1 つの署名なしトークンを書き込みます

```
aws-cloudhsm > user change-mfa token-sign --username cu1 --change-password password --
role crypto-user --token /path/myfile
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:/path/mypemfile
{
  "error_code": 0,
  "data": {
    "username": "test_user",
    "role": "admin"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<ROLE>

ユーザーアカウントに付与されるロールを指定します。このパラメータは必須です。HSM のユーザータイプの詳細については、「HSM ユーザーについて」を参照してください。

有効な値

- Admin:管理者はユーザーを管理できますが、キーを管理することはできません。
- Crypto user: Crypto User は、管理キーを作成し、暗号化オペレーションでキーを使用できます。

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。

ユーザーの作成後にユーザー名を変更することはできません。CloudHSM CLI コマンドでは、ロールとパスワードでは大文字と小文字が区別されますが、ユーザー名では区別されません。

必須: はい

<CHANGE_PASSWORD>

MFA を登録/登録解除するユーザーのプレーンテキストの新しいパスワードを指定します。

必須: はい

<TOKEN>

署名なしトークンファイルが書き込まれるファイルパス。

必須: はい

<APPROVAL>

オペレーションを承認する署名付きクォーラムトークンファイルへのファイルパスを指定します。クォーラムユーザーサービスのクォーラム値が 1 より大きい場合にのみ必要です。

<DEREGISTER>

MFA パブリックキーが存在する場合、登録を解除します。

<CHANGE-QUORUM>

クォーラムパブリックキーを MFA キーと一緒に変更します。

関連トピック

- HSM ユーザー用 2FA について

user change-password

CloudHSM CLI の user change-password コマンドを使用して、 AWS CloudHSM クラスター内の既存のユーザーのパスワードを変更します。ユーザーの MFA を有効にするには、`user change-mfa` コマンドを使用します。

どのユーザーも自分のパスワードを変更できます。さらに、管理者ロールを持つユーザーは、クラスター内の別のユーザーのパスワードを変更できます。変更するために現在のパスワードを入力する必要はありません。

> **Note**
>
> 現在クラスターにログインしているユーザーのパスワードは変更できません。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理
- Crypto user (CU)

Syntax

> **Note**
>
> ユーザーの多要素認証 (MFA) を有効にするには、user change-mfa コマンドを使用します。

```
aws-cloudhsm > help user change-password
Change a user's password

    Usage:
        cloudhsm-cli user change-password [OPTIONS] --username <USERNAME> --role <ROLE>
 [--password <PASSWORD>]

    Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username of the user that will be modified

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --password <PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it
```

```
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

      --deregister-mfa <DEREGISTER-MFA>
          Deregister the user's mfa public key, if present

      --deregister-quorum <DEREGISTER-QUORUM>
          Deregister the user's quorum public key, if present
 -h, --help
          Print help (see a summary with '-h')
```

例

次の例は、user change-password を使用して、現在のユーザーまたはクラスター内の他のユーザーのパスワードをリセットする方法を示しています。

Example : パスワードの変更

クラスター内のすべてのユーザーは、user change-password を使用して自分のパスワードを変更できます。

次の出力は、Bob が現在 Crypto User (CU) としてログインしていることを示しています。

```
aws-cloudhsm > user change-password --username bob --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "bob",
    "role": "crypto-user"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<APPROVAL>

オペレーションを承認する署名付きクォーラムトークンファイルへのファイルパスを指定します。クォーラムユーザーサービスのクォーラム値が 1 より大きい場合にのみ必要です。

<DEREGISTER-MFA>

MFA パブリックキーが存在する場合、登録を解除します。

<DEREGISTER-QUORUM>

Quorum パブリックキーがある場合は、登録を解除します。

<PASSWORD>

ユーザーのプレーンテキストの新しいパスワードを指定します。

必須: はい

<ROLE>

ユーザーアカウントに付与されるロールを指定します。このパラメータは必須です。HSM のユーザータイプの詳細については、「HSM ユーザーについて」を参照してください。

有効な値

- Admin:管理者はユーザーを管理できますが、キーを管理することはできません。
- Crypto user: Crypto User は、管理キーを作成し、暗号化オペレーションでキーを使用できます。

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。

ユーザーの作成後にユーザー名を変更することはできません。CloudHSM CLI コマンドでは、ロールとパスワードでは大文字と小文字が区別されますが、ユーザー名では区別されません。

必須: はい

関連トピック

- user list
- user create

- user delete

user change-quorum

user change-quorum はコマンドグループの親カテゴリであり、これを親カテゴリと組み合わせると、ユーザーのクォーラム変更専用のコマンドが作成されます。

user change-quorum は指定されたクォーラム戦略を使用してユーザークォーラム認証を登録するために使用されます。SDK 5.8.0 では、次に示すように、ユーザーが利用できるクォーラム戦略は 1 つだけです。

現在、このカテゴリは次のカテゴリとサブコマンドで構成されています。

- token-sign
  - 登録

user change-quorum token-sign

user change-quorum token-sign は、コマンドの親カテゴリで、この親カテゴリと組み合わせるとトークン署名クォーラムオペレーション専用のコマンドが作成されます。

現在、このカテゴリは以下のコマンドで構成されています。

- 登録

ユーザー変更クォーラムトークン署名登録

CloudHSM CLI の user change-quorum token-sign register コマンドを使用して、管理者ユーザーのトークン署名クォーラム戦略を登録します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理

Syntax

```
aws-cloudhsm > help user change-quorum token-sign register
```

```
Register a user for quorum authentication with a public key

Usage: user change-quorum token-sign register --public-key <PUBLIC_KEY> --signed-
token <SIGNED_TOKEN>

Options:
      --cluster-id <CLUSTER_ID>      Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --public-key <PUBLIC_KEY>      Filepath to public key PEM file
      --signed-token <SIGNED_TOKEN>  Filepath with token signed by user private key
  -h, --help Print help (see a summary with '-h')
```

例

Example

このコマンドを実行するには、register quorum token-sign を実行するユーザーとしてログインする必要があります。

```
aws-cloudhsm > login --username admin1 --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
  }
}
```

user change-quorum token-sign register コマンドは HSM にパブリックキーを登録します。その結果、必要なクォーラム値のしきい値を満たすためにユーザーがクォーラム署名を取得する必要があるクォーラム必須オペレーションのクォーラム承認者としての資格が得られます。

```
aws-cloudhsm > user change-quorum token-sign register \
    --public-key /home/mypemfile \
    --signed-token /home/mysignedtoken
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
```

```
  }
}
```

これで、user list コマンドを実行して、このユーザーにクォーラムトークン署名が登録されていることを確認できます。

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "admin1",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<PUBLIC-KEY>

パブリックキー PEM ファイルへのファイルパス。

必須: はい

<SIGNED-TOKEN>

ユーザーのプライベートキーで署名されたトークンを含むファイルパス。

必須: はい

関連トピック

- CloudHSM CLI を使用してクォーラム認証を管理する
- 管理者用クォーラム認証を使用する: 初回セットアップ
- 管理者のクォーラム最小値を変更する
- クォーラム認証をサポートするサービス名とタイプ

user create

CloudHSM CLI の user create コマンドは、 AWS CloudHSM クラスターにユーザーを作成します。このコマンドを実行できるのは、管理者ロールを持つユーザーアカウントのみです。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- 管理

要件

このコマンドを実行するには、管理者ユーザーとしてログインする必要があります

Syntax

```
aws-cloudhsm > help user create
Create a new user
```

```
Usage: cloudhsm-cli user create [OPTIONS] --username <USERNAME> --role <ROLE> [--
password <PASSWORD>]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username to access the HSM cluster

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --password <PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

例

以下の例では、user create を使用して HSM に新しいユーザーを作成する方法を示します。

Example : Crypto User を作成する

この例では、 暗号化ユーザーロールを使用して AWS CloudHSM クラスターに アカウントを作成します。

```
aws-cloudhsm > user create --username alice --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 0,
```

```
  "data": {
    "username": "alice",
    "role": "crypto-user"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。このコマンドではユーザー名の大文字と小文字は区別されません。ユーザー名は常に小文字で表示されます。

必須: はい

<ROLE>

このユーザーに割り当てられるロールを指定します。このパラメータは必須です。有効な値は admin、crypto-user です。

ユーザーのロールを取得するには、user list コマンドを使用します。HSM のユーザー タイプの詳細については、「HSM ユーザーについて」を参照してください。

<PASSWORD>

HSM にログインしているユーザーのパスワードを指定します。

必須: はい

<APPROVAL>

オペレーションを承認する署名付きクォーラムトークンファイルへのファイルパスを指定します。クォーラムユーザーサービスのクォーラム値が 1 より大きい場合にのみ必要です。

関連トピック

- user list

- user delete
- user change-password

user delete

CloudHSM CLI の user delete コマンドは、 AWS CloudHSM クラスターからユーザーを削除します。このコマンドを実行できるのは、管理者ロールを持つユーザーアカウントのみです。現在 HSM にログインしているユーザーを削除することはできません。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- 管理

要件

- キーを所有しているユーザーアカウントを削除することはできません。
- このコマンドを実行するには、ユーザーアカウントに管理者ロールが必要です。

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
aws-cloudhsm > help user delete
Delete a user

Usage: user delete [OPTIONS] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username to access the HSM cluster

      --role <ROLE>
```

```
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
```

例

```
aws-cloudhsm > user delete --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "username": "alice",
    "role": "crypto-user"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<USERNAME>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。このコマンドではユーザー名の大文字と小文字は区別されません。ユーザー名は常に小文字で表示されます。

必須: はい

<ROLE>

このユーザーに割り当てられるロールを指定します。このパラメータは必須です。有効な値は admin、crypto-user です。

ユーザーのロールを取得するには、user list コマンドを使用します。HSM のユーザータイプの詳細については、「HSM ユーザーについて」を参照してください。

必須: はい

<APPROVAL>

オペレーションを承認する署名付きクォーラムトークンファイルへのファイルパスを指定します。クォーラムユーザーサービスのクォーラム値が 1 より大きい場合にのみ必要です。

必須: はい

関連トピック

- user list
- user create
- user change-password

user list

CloudHSM CLI の user list コマンドは、CloudHSM クラスターに存在するユーザーアカウントを一覧表示します。このコマンドは、CloudHSM CLI にログインしていなくても実行できます。

> **Note**
>
> HSMs を追加または削除する場合は、 AWS CloudHSM クライアントとコマンドラインツールが使用する設定ファイルを更新します。そうしないと、クラスタ内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

Syntax

```
aws-cloudhsm > help user list
List the users in your cluster
```

```
USAGE:
    user list

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

例

このコマンドは、CloudHSM クラスターに存在するユーザーを一覧表示します。

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "test_user",
        "role": "admin",
        "locked": "false",
        "mfa": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
```

```
    ]
  }
}
```

この出力が示すユーザー属性は以下のとおりです。

- Username: ユーザー定義のわかりやすいユーザー名を表示します。ユーザー名は常に小文字で表示されます。
- Role: HSM でユーザーが実行できるオペレーションを決定します。
- Locked: このユーザーアカウントがロックアウトされているかどうかを示します。
- MFA: このユーザーアカウントでサポートされている多要素認証メカニズムを示します。
- Cluster coverage: このユーザーアカウントのクラスター全体での可用性を示します。

関連トピック

- key_mgmt_util で listUsers
- user create
- user delete
- user change-password

クォーラム

quorum は、quorum と組み合わせた場合、クォーラム認証または M of N オペレーションに固有のコマンドを作成するコマンドグループの親カテゴリになります。現在、このカテゴリは独自のコマンドで構成される token-sign サブカテゴリで構成されています。詳細については、以下のリンクをクリックしてください。

- token-sign

管理サービス: クォーラム認証は、ユーザーの作成、ユーザーの削除、ユーザーパスワードの変更、クォーラム値の設定、クォーラム機能と MFA 機能の無効化などの管理者権限を持つサービスに使用されます。

各サービスタイプはさらに適格なサービス名に分類されます。このサービス名には、実行可能なクォーラムがサポートする特定のサービスオペレーションのセットが含まれます。

サービス名	サービスタイプ	サービスオペレーション
ユーザー	管理	• user create • user delete • user change-password • user change-mfa
クォーラム	管理	• クォーラムトークン署名 set-quorum-value

関連トピック

- 管理者用クォーラム認証を使用する: 初回セットアップ
- CloudHSM CLI を使用したクォーラム認証の管理 (M of N のアクセスコントロール)

quorum token-sign

quorum token-sign はコマンドグループのカテゴリで、quorum token-sign と組み合わせるとクォーラム認証または M of N オペレーションに特化したコマンドが作成されます。

現在、このカテゴリは次のコマンドで構成されています。

- 削除
- 生成
- list
- list-quorum-values
- list-timeouts
- set-quorum-value
- set-timeout

quorum token-sign delete

CloudHSM CLI の quorum token-sign delete のコマンドを使用して、クォーラム承認サービスの 1 つ以上のトークンを削除します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理

Syntax

```
aws-cloudhsm > help quorum token-sign delete
Delete one or more Quorum Tokens

Usage: quorum token-sign delete --scope <SCOPE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --scope <SCOPE>
          Scope of which token(s) will be deleted

          Possible values:
          - user: Deletes all token(s) of currently logged in user
          - all:  Deletes all token(s) on the HSM
  -h, --help
          Print help (see a summary with '-h')
```

例

次の例は、CloudHSM CLI の quorum token-sign delete コマンドを使用して、クォーラム認定サービスの 1 つ以上のトークンを削除する方法を示しています。

Example : クォーラム認定サービスの 1 つ以上のトークンを削除します

```
aws-cloudhsm > quorum token-sign delete --scope all
{
  "error_code": 0,
  "data": "Deletion of quorum token(s) successful"
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<SCOPE>

AWS CloudHSM クラスター内でトークン (複数可) が削除されるスコープ。

有効な値

- ユーザー: ログインしているユーザーが所有するトークンのみを削除する場合に使用します。
- すべて: AWS CloudHSM クラスター内のすべてのトークンを削除するために使用されます。

関連トピック

- user list
- user create
- user delete

quorum token-sign generate

CloudHSM CLI の quorum token-sign generate コマンドを使用して、クォーラム承認サービスのトークンを生成します。

サービスユーザーとクォーラムの HSM クラスターでは、サービスごとに 1 ユーザーにつき 1 つのアクティブトークンを取得することには制限があります。

> **Note**
>
> サービストークンを生成できるのは管理者のみです。

管理サービス: クォーラム認証は、ユーザーの作成、ユーザーの削除、ユーザーパスワードの変更、クォーラム値の設定、クォーラム機能と MFA 機能の無効化などの管理者権限を持つサービスに使用されます。

各サービスタイプはさらに適格なサービス名に分類されます。このサービス名には、実行可能なクォーラムがサポートする特定のサービスオペレーションのセットが含まれます。

サービス名	サービスタイプ	サービスオペレーション
ユーザー	管理	• user create • user delete • user change-password • user change-mfa
クォーラム	管理	• クォーラムトークン署名 set-quorum-value

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help quorum token-sign generate
Generate a token

Usage: quorum token-sign generate --service <SERVICE> --token <TOKEN>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --service <SERVICE>
          Service the token will be used for

          Possible values:
          - user:
```

```
            User management service is used for executing quorum authenticated user
 management operations
          - quorum:
            Quorum management service is used for setting quorum values for any quorum
 service
          - registration:
            Registration service is used for registering a public key for quorum
 authentication

      --token <TOKEN>
          Filepath where the unsigned token file will be written
  -h, --help                     Print help
```

例

このコマンドは、クラスター内の HSM ごとに 1 つの未署名トークンを token で指定されたファイルに書き込みます。

Example : クラスターの HSM ごとに 1 つの署名なしトークンを書き込みます

```
aws-cloudhsm > quorum token-sign generate --service user --token /home/tfile
{
  "error_code": 0,
  "data": {
    "filepath": "/home/tfile"
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<SERVICE>

トークンを生成するには、クォーラム承認サービスを指定します。このパラメータは必須です。

有効な値

- ユーザー:クォーラム承認ユーザー管理オペレーションの実行に使用されるユーザー管理サービス。

- クォーラム: すべてのクォーラム承認サービスのクォーラム承認クォーラム値を設定するために使用されるクォーラム管理サービス。
- 登録:クォーラム認証用のパブリックキーの登録に使用する署名なしトークンを生成します。

必須: はい

<TOKEN>

署名されていないトークンファイルが書き込まれるファイルパス。

必須: はい

関連トピック

- クォーラム認証をサポートするサービス名とタイプ

quorum token-sign list

CloudHSM CLI の quorum token-sign list コマンドを使用して、 AWS CloudHSM クラスターに存在するすべてのトークン署名クォーラムトークンを一覧表示します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help quorum token-sign list
List the token-sign tokens in your cluster

Usage: quorum token-sign list

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

例

このコマンドは、 AWS CloudHSM クラスターに存在するすべてのトークン署名トークンを一覧表示します。

Example

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": [
      {
        "username": "admin",
        "service": "quorum",
        "approvals-required": 2
        "number-of-approvals": 0
        "token-timeout-seconds": 397
        "cluster-coverage": "full"
      },
      {
        "username": "admin",
        "service": "user",
        "approvals-required": 2
        "number-of-approvals": 2
        "token-timeout-seconds": 588
        "cluster-coverage": "full"
      }
    ]
  }
}
```

関連トピック

- quorum token-sign generate

クォーラムトークン署名 list-quorum-values

CloudHSM CLI の quorum token-sign list-quorum-values コマンドを使用して、 AWS CloudHSM クラスターに設定されたクォーラム値を一覧表示します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

Syntax

```
aws-cloudhsm > help quorum token-sign list-quorum-values
List current quorum values

Usage: quorum token-sign list-quorum-values

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

例

このコマンドは、各サービスの AWS CloudHSM クラスターに設定されたクォーラム値を一覧表示します。

Example

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 1
  }
}
```

関連トピック

- クォーラム認証をサポートするサービス名とタイプ

quorum token-sign list-timeouts

CloudHSM CLI の quorum token-sign list-timeouts コマンドを使用して、すべてのトークンタイプのトークンタイムアウト時間を秒単位で取得します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

Syntax

```
aws-cloudhsm > help quorum token-sign list-timeouts
List timeout durations in seconds for token validity

Usage: quorum token-sign list-timeouts

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

例

Example

```
aws-cloudhsm > quorum token-sign list-timeouts
{
  "error_code": 0,
  "data": {
    "generated": 600,
    "approved": 600
  }
}
```

出力には以下が含まれます。

- 生成済み: 生成されたトークンが承認されるまでのタイムアウト時間 (秒単位)。
- 承認済み: 承認されたトークンを使用してクォーラム承認オペレーションを実行するまでのタイムアウト時間 (秒単位)。

関連トピック

- quorum token-sign set-timeout

クォーラムトークン署名 set-quorum-value

CloudHSM CLI の quorum token-sign set-quorum-value コマンドを使用して、クォーラム承認サービス用の新しいクォーラム値のトークンを設定します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理

Syntax

```
aws-cloudhsm > help quorum token-sign set-quorum-value
Set a quorum value

Usage: quorum token-sign set-quorum-value [OPTIONS] --service <SERVICE> --value <VALUE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --service <SERVICE>
          Service the token will be used for

          Possible values:
          - user:
            User management service is used for executing quorum authenticated user
 management operations
          - quorum:
            Quorum management service is used for setting quorum values for any quorum
 service

      --value <VALUE>
          Value to set for service
```

```
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

例

Example

次の例では、このコマンドは、クラスター内の HSM ごとに 1 つの署名なしトークンを、トークンで指定されたファイルに書き込みます。プロンプトが表示されたら、ファイル内のトークンに署名します。

```
aws-cloudhsm > quorum token-sign set-quorum-value --service quorum --value 2
{
  "error_code": 0,
  "data": "Set Quorum Value successful"
}
```

その後、list-quorum-values コマンドを実行して、クォーラム管理サービスのクォーラム値が設定されていることを確認できます。

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 2
  }
}
```

引数

<CLUSTER_ID>

このオペレーションを実行するクラスターの ID。

必須: 複数のクラスターが設定されている場合。

<APPROVAL>

HSM で承認される署名済みトークンファイルのファイルパス。

<SERVICE>

トークンを生成するには、クォーラム承認サービスを指定します。このパラメータは必須です。サービスのタイプと名前について詳しくは、「クォーラム認証をサポートするサービス名とタイプ」を参照してください。

有効な値

- ユーザー: ユーザー管理サービス。クォーラム承認ユーザー管理オペレーションの実行に使用されるサービス。
- クォーラム: クォーラム管理サービス。あらゆるクォーラム承認サービスのクォーラム承認クォーラム値を設定するために使用されるサービス。
- 登録: クォーラム認証用のパブリックキーの登録に使用する署名なしトークンを生成します。

必須: はい

<VALUE>

設定するクォーラム値を指定します。最大クォーラム値は 8 です。

必須: はい

関連トピック

- クォーラムトークン署名 list-quorum-values
- クォーラム認証をサポートするサービス名とタイプ

quorum token-sign set-timeout

CloudHSM CLI の quorum token-sign set-timeout コマンドを使用して、各トークンタイプのトークンのタイムアウト時間を秒単位で設定します。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- 管理

Syntax

```
aws-cloudhsm > help quorum token-sign set-timeout
```

```
Set timeout duration in seconds for token validity

Usage: quorum token-sign set-timeout <--generated <GENERATED> |--approved <APPROVED>>

Options:
      --cluster-id <CLUSTER_ID>   Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --generated <GENERATED>     Timeout period in seconds for a generated (non-
approved) token to be approved
      --approved <APPROVED>      Timeout period in seconds for an approved token to be
 used to execute a quorum operation
  -h, --help    Print help (see a summary with '-h')
```

例

次の例では、quorum token-sign set-timeout コマンドでトークンのタイムアウト期間を設定する方法を示します。

```
aws-cloudhsm > quorum token-sign set-timeout --generated 900
{
  "error_code": 0,
  "data": "Set token timeout successful"
}
```

関連トピック

- quorum token-sign list-timeouts

CloudHSM 管理ユーティリティ (CMU)

cloudhsm_mgmt_util と呼ばれるコマンドラインツールは、Crypto Officer が HSM 内のユーザーを管理するのに役立ちます。これには、ユーザーの作成、削除および一覧表示とユーザーパスワードの変更を行うツールが含まれています。

KMU と CMU は Client SDK 3 スイートの一部です。Client SDK 3 および関連するコマンドラインツール (キー管理ユーティリティおよび CloudHSM 管理ユーティリティ) は、HSM タイプ hsm1.medium でのみ使用できます。

また cloudhsm_mgmt_util は、Crypto User (CU) がキーを共有することを可能にするコマンド、およびキー属性を取得して設定することを可能にするコマンドも含まれています。これらのコマンドは、プライマリーキー管理ツールである key_mgmt_util のキー管理コマンドを補完するものです。

クイックスタートについては、「クローンされたクラスターを管理する」を参照してください。cloudhsm_mgmt_util コマンドの詳細情報とコマンドの使用例については、「cloudhsm_mgmt_util コマンドリファレンス」を参照してください。

トピック

- AWS CloudHSM 管理ユーティリティでサポートされているプラットフォーム
- CloudHSM 管理ユーティリティ (CMU) の使用開始
- AWS CloudHSM クライアントのインストールと設定 (Linux)
- AWS CloudHSM クライアントのインストールと設定 (Windows)
- cloudhsm_mgmt_util コマンドリファレンス

AWS CloudHSM 管理ユーティリティでサポートされているプラットフォーム

Linux サポート

- Amazon Linux
- Amazon Linux 2
- CentOS 6.10+
- CentOS 7.3+
- CentOS 8
- Red Hat Enterprise Linux (RHEL) 6.10+
- Red Hat Enterprise Linux (RHEL) 7.9+
- Red Hat Enterprise Linux (RHEL) 8
- Ubuntu 16.04 LTS
- Ubuntu 18.04 LTS

Windows サポート

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

CloudHSM 管理ユーティリティ (CMU) の使用開始

CloudHSM 管理ユーティリティ (CMU) を使用すると、ハードウェアセキュリティモジュール (HSM) ユーザーを管理できます。このトピックを使用して、ユーザーの作成、ユーザーのリスト、CMU のクラスターへの接続など、基本的な HSM ユーザー管理タスクを開始します。

1. CMU を使用するには、まず configure ツールを使用して、クラスタ内の HSM の 1 つから `--cmu` パラメータと IP アドレスを使用してローカルの CMU 設定を更新する必要があります。CMU を使用するたびにこれを実行して、クラスター内のすべての HSM で HSM ユーザーを管理していることを確認します。

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM\bin\ configure.exe --cmu <IP address>
   ```

2. 次のコマンドを使用して CLI をインタラクティブモードで起動します。

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   C:\Program Files\Amazon\CloudHSM> .\cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
   \CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

保有する HSM の数に応じて、出力は次のようになります。

```
Connecting to the server(s), it may take time
depending on the server(s) load, please wait...

Connecting to server '10.0.2.9': hostname '10.0.2.9', port 2225...
Connected to server '10.0.2.9': hostname '10.0.2.9', port 2225.

Connecting to server '10.0.3.11': hostname '10.0.3.11', port 2225...
Connected to server '10.0.3.11': hostname '10.0.3.11', port 2225.

Connecting to server '10.0.1.12': hostname '10.0.1.12', port 2225...
Connected to server '10.0.1.12': hostname '10.0.1.12', port 2225.
```

key_mgmt_util が実行されていると、プロンプトは aws-cloudhsm> に変わります。

3. loginHSM コマンドを使用して、クラスターにログインします。どのタイプのユーザーでも、このコマンドを使用してクラスターにログインすることができます。

   次のコマンド例では、admin でログインしています。これは、デフォルトの暗号担当者 (CO) です。このユーザーのパスワードは、「クラスターのアクティブ化」を行う場合に設定します。-hpswd パラメータを使用して、パスワードを非表示にします。

   ```
   aws-cloudhsm>loginHSM CO admin -hpswd
   ```

   システムからパスワードの入力を求められます。パスワードを入力するとシステムはパスワードを非表示にし、コマンドが正常に実行されたことと、クラスター上のすべての HSM に接続したことが出力で示されます。

   ```
   Enter password:


   loginHSM success on server 0(10.0.2.9)
   loginHSM success on server 1(10.0.3.11)
   loginHSM success on server 2(10.0.1.12)
   ```

4. listUsers を使用して、クラスター上のすべてのユーザーを一覧表示します。

   ```
   aws-cloudhsm>listUsers
   ```

CMU は、クラスター上のすべてのユーザーを一覧表示します。

```
Users on server 0(10.0.2.9):
Number of users found:2

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                   NO
               0               NO
         2              AU              app_user                                NO
               0               NO
Users on server 1(10.0.3.11):
Number of users found:2

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                   NO
               0               NO
         2              AU              app_user                                NO
               0               NO
Users on server 2(10.0.1.12):
Number of users found:2

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                   NO
               0               NO
         2              AU              app_user                                NO
               0               NO
```

5. createUser を使用して、 **example_user** という CU ユーザーをパスワードは **password1** で作成します。

   アプリケーションで CU ユーザーを使用して、暗号化およびキー管理操作を実行します。ステップ 3 で CO ユーザーとしてログインしたため、CU ユーザーを作成できます。CMU でユーザーの作成および削除や、他のユーザーのパスワード変更などのユーザー管理作業を行うことができるのは、 CO ユーザーのみです。

   ```
   aws-cloudhsm>createUser CU example_user password1
   ```

CMU はユーザーの作成操作についてプロンプトを表示します。

```
**********************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?
```

6. CU ユーザーを作成するには **example_user**、**y** と入力します。
7. listUsers を使用して、クラスター上のすべてのユーザーを一覧表示します。

```
aws-cloudhsm>listUsers
```

CMU は先ほど作成した新しい CU ユーザーを含む、クラスター上のすべてのユーザーを一覧表示します。

```
Users on server 0(10.0.2.9):
Number of users found:3

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                   NO
               0               NO
         2              AU              app_user                                NO
               0               NO
         3              CU              example_user                            NO
               0               NO
Users on server 1(10.0.3.11):
Number of users found:3

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                   NO
               0               NO
         2              AU              app_user                                NO
               0               NO
```

```
         3              CU              example_user                    NO
               0              NO
Users on server 2(10.0.1.12):
Number of users found:3

    User Id             User Type       User Name
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                           NO
               0              NO
         2              AU              app_user                        NO
               0              NO
         3              CU              example_user                    NO
               0              NO
```

8. HSM からログアウトするには、logoutHSM コマンドを使用します。

```
aws-cloudhsm>logoutHSM


logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
logoutHSM success on server 2(10.0.1.12)
```

9. cloudhsm_mgmt_util を停止するには quit コマンドを使用します。

```
aws-cloudhsm>quit


disconnecting from servers, please wait...
```

AWS CloudHSM クライアントのインストールと設定 (Linux)

AWS CloudHSM クラスター内の HSM を操作するには、Linux 用の AWS CloudHSM クライアントソフトウェアが必要です。このクライアントを以前に作成した Linux EC2 クライアントインスタンスにインストールする必要があります。Windows を使用している場合は、クライアントをインストールすることもできます。詳細については、「AWS CloudHSM クライアントのインストールと設定 (Windows)」を参照してください。

タスク

- AWS CloudHSM クライアントツールとコマンドラインツールをインストールする

- クライアント設定の編集

AWS CloudHSM クライアントツールとコマンドラインツールをインストールする

クライアントインスタンスに接続し、次のコマンドを実行して、 AWS CloudHSM クライアントおよびコマンドラインツールをダウンロードしてインストールします。

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb
```

クライアント設定の編集

AWS CloudHSM クライアントを使用してクラスターに接続する前に、クライアント設定を編集する必要があります。

クライアント設定を編集するには

1. cloudhsm_mgmt_util に Client SDK 3 をインストールする場合は、次の手順を実行して、クラスター内のすべてのノードが同期されていることを確認します。

   a. configure -a ***<IP of one of the HSMs>*** を実行します。

   b. クライアントサービスを再起動します。

   c. config -m を実行します。

2. 発行証明書 (クラスターの証明書に署名するために使用したもの) を、クライアントインスタンスの次の場所にコピーします：`/opt/cloudhsm/etc/customerCA.crt`。この場所に証明書をコピーするには、クライアントインスタンスにルートユーザーアクセス権限が必要です。

3. 次の configure コマンドを使用して、 AWS CloudHSM クライアントツールとコマンドラインツールの設定ファイルを更新し、クラスター内の HSM の IP アドレスを指定します。HSM の IP アドレスを取得するには、AWS CloudHSM コンソール でクラスターを表示するか、 describe-clusters AWS CLI コマンドを実行します。コマンドの出力では、HSM の IP アドレスは `EniIp` フィールドの値です。複数の HSM がある場合は、いずれかの HSM の IP アドレスを選択してください。どれでも構いません。

   ```
   sudo /opt/cloudhsm/bin/configure -a <IP address>

   Updating server config in /opt/cloudhsm/etc/cloudhsm_client.cfg
   Updating server config in /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

4. クラスターのアクティブ化 に移動します。

AWS CloudHSM クライアントのインストールと設定 (Windows)

Windows で AWS CloudHSM クラスター内の HSM を使用するには、Windows 用の AWS CloudHSM クライアントソフトウェアが必要です。このクライアントを以前に作成した Windows Server インスタンスにインストールする必要があります。

最新 Windows クライアントとコマンドラインツールをインストール (または更新) します。

1. Windows Server インスタンスに接続します。
2. AWSCloudHSMClient-latest.msi インストーラ をダウンロードします。
3. cloudhsm_mgmt_util に Client SDK 3 をインストールする場合は、次の手順を実行して、クラスター内のすべてのノードが同期されていることを確認します。

   a. configure -a ***<IP of one of the HSMs>*** を実行します。

   b. クライアントサービスを再起動します。

   c. config -m を実行します。
4. ダウンロード場所に移動し、管理者権限を持つインストーラ (AWSCloudHSMClient-latest.msi) を実行します。
5. インストーラの手順に従い、インストーラが終了したら 閉じる を選択します。
6. クラスターの証明書に署名するために使用した 自己署名発行証明書を `C:\ProgramData\Amazon\CloudHSM` フォルダにコピーします。
7. 以下のコマンドを実行して、設定ファイルを更新します。更新中は、再設定の間に必ずクライアントを停止してから再開します。

```
C:\Program Files\Amazon\CloudHSM\bin\ configure.exe -a <HSM IP address>
```

8. クラスターのアクティブ化 に移動します。

注意:

- クライアントを更新する場合、以前のインストールに存在する設定ファイルは上書き されません。
- Windows 用の AWS CloudHSM クライアントインストーラは、Cryptography API: Next Generation (CNG) とキーストレージプロバイダー (KSP) を自動的に登録します。クライアントをアンインストールするには、インストーラーを再度実行し、アンインストール手順に従います。
- Linux を使用している場合は、Linux クライアントをインストールすることもできます。詳細については、「AWS CloudHSM クライアントのインストールと設定 (Linux)」を参照してください。

cloudhsm_mgmt_util コマンドリファレンス

cloudhsm_mgmt_util コマンドラインツールは、Crypto Officer が HSM 内のユーザーを管理するのに役立ちます。また、Crypto User (CU) がキーを共有することを可能にするコマンド、およびキー属性を取得して設定することを可能にするコマンドも含まれています。これらのコマンドは、key_mgmt_util コマンドラインツールのプライマリキー管理コマンドを補完します。

クイックスタートについては、「クローンされたクラスターを管理する」を参照してください。

cloudhsm_mgmt_util コマンドを実行する前に、 cloudhsm_mgmt_util を起動し、HSM にログインする必要があります。ログインに使用するアカウントのユーザータイプで、使用するコマンドを実行できることを確認してください。

cloudhsm_mgmt_util コマンドをすべて一覧表示するには、次のコマンドを実行します。

```
aws-cloudhsm> help
```

cloudhsm_mgmt_util コマンドの構文を取得するには、次のコマンドを実行します。

```
aws-cloudhsm> help <command-name>
```

> **Note**
>
> 構文は、ドキュメントに従って使用してください。組み込みのソフトウェアヘルプによって追加のオプションが使用できる場合がありますが、これらはサポートされているとみなさず、本番稼働コードで使用しないことをお勧めします。

コマンドを実行するには、コマンド名、または他の cloudhsm_mgmt_util コマンドの名前と区別するのに十分な名前を入力します。

たとえば、HSM 上のユーザーのリストを取得するには、listUsers または listU と入力します。

```
aws-cloudhsm> listUsers
```

cloudhsm_mgmt_util のセッションを終了するには、次のコマンドを実行します。

```
aws-cloudhsm> quit
```

キー属性の解釈については、「キー属性リファレンス」を参照してください。

以下のトピックでは、cloudhsm_mgmt_util のコマンドについて説明します。

> **Note**
>
> key_mgmt_util と cloudhsm_mgmt_util のコマンドには、同じ名前のものがあります。ただし、コマンドは通常、構文が異なり、出力が異なり、機能がわずかに異なります。

コマンド	説明	ユーザータイプ
changePswd	HSM 上のユーザーのパスワードを変更します。どのユーザーも自分のパスワードを変更できます。CO は誰のパスワードでも変更できます。	CO
createUser	HSM 上のすべてのタイプのユーザーを作成します。	CO
deleteUser	HSM からすべてのタイプのユーザーを削除します。	CO
findAllKeys	ユーザーが所有または共有するキーを取得します。また、各 HSM のすべてのキーの、キー所有権と共有データのハッシュを取得します。	CO、AU
getAttribute	AWS CloudHSM キーの属性値を取得し、ファイルまたは stdout (標準出力) に書き込みます。	CU
getCert	特定の HSM の証明書を取得し、目的の証明書の形式で保存します。	すべて。

コマンド	説明	ユーザータイプ
getHSMInfo	HSM が実行されているハードウェアに関する情報を取得します。	すべて。ログインは必須ではありません。
getKeyInfo	所有者、共有ユーザー、およびキーのクォーラム認証ステータスを取得します。	すべて。ログインは必須ではありません。
info	IP アドレス、ホスト名、ポート、および現在のユーザーを含む、HSM に関する情報を取得します。	すべて。ログインは必須ではありません。
listUsers	各 HSM のユーザー、そのユーザータイプと ID、およびその他の属性を取得します。	すべて。ログインは必須ではありません。
loginHSM および logoutHSM	HSM へのログインとログアウト。	すべて。
quit	cloudhsm_mgmt_util を終了します。	すべて。ログインは必須ではありません。
サーバー	HSM のサーバーモードの起動と終了を行います。	すべて。
registerQuorumPubキー	HSM ユーザーを非対称 RSA-2048 キーペアに関連付けます。	CO
setAttribute	既存のキーのラベル、暗号化、復号、ラップ、およびラップ解除の属性の値を変更します。	CU
shareKey	既存のキーを他のユーザーと共有します。	CU

コマンド	説明	ユーザータイプ
syncKey	クローンされた AWS CloudHSM クラスター間でキーを同期します。	CU、CO
syncUser	クローンされた AWS CloudHSM クラスター間でユーザーを同期します。	CO

changePswd

cloudhsm_mgmt_util の changePswd コマンドは、クラスター内の HSM の既存のユーザーのパスワードを変更します。

どのユーザーも自分のパスワードを変更できます。さらに、Crypto Officer（COおよびPCO）は、別のCOまたは Crypto ユーザー（CU）のパスワードを変更することができます。変更するために現在のパスワードを入力する必要はありません。

> **Note**
>
> 現在 AWS CloudHSM クライアントまたは key_mgmt_util にログインしているユーザーのパスワードを変更することはできません。

changePswd のトラブルシューティングを行うには

CMU コマンドを実行する前に CMU を起動し、 HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- Crypto Officer (CO)
- Crypto User (CU)

構文

引数は構文の図表で指定された順序で入力します。-hpswd パラメータを使用して、パスワードをマスクします。CO ユーザーの 2 要素認証 (2FA) を有効にするには、-2fa パラメータを入力し、ファイルパスを含めます。詳細については、「the section called “引数”」を参照してください。

```
changePswd <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

例

次の例は、changePassword を使用して、現在のユーザーまたは HSM 内の他のユーザーのパスワードをリセットする方法を示しています。

Example : パスワードの変更

HSM のすべてのユーザーは changePswd を使用して自分のパスワードを変更できます。パスワードを変更する前に、info を使用して、ユーザー名やログインユーザーのユーザータイプなど、クラスター内の各 HSM に関する情報を取得します。

次の出力は、Bob が現在 Crypto User (CU) としてログインしていることを示しています。

```
        aws-cloudhsm> info server 0

Id      Name                    Hostname         Port   State           Partition
  LoginState
0       10.1.9.193              10.1.9.193      2225    Connected       hsm-jqici4covtv
  Logged in as 'bob(CU)'

aws-cloudhsm> info server 1

Id      Name                    Hostname         Port   State           Partition
  LoginState
1       10.1.10.7               10.1.10.7       2225    Connected       hsm-ogi3sywxbqx
  Logged in as 'bob(CU)'
```

パスワードを変更するために、Bob は changePswd に続けて、ユーザータイプ、ユーザー名、および新しいパスワードを指定して実行します。

```
aws-cloudhsm> changePswd CU bob newPassword
```

```
************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?y
Changing password for bob(CU) on 2 nodes
```

Example : 別のユーザーのパスワードを変更する

HSM 上の他の CO、または CU のパスワードを変更するには、CO または PCO である必要があります。他のユーザーのパスワードを変更する前に、info コマンドを使用して、自分のユーザタイプが CO または PCO であることを確認してください。

次の出力では、CO である Alice が現在ログインしていることが確認できます。

```
aws-cloudhsm>info server 0

Id      Name             Hostname         Port   State           Partition
 LoginState
0       10.1.9.193       10.1.9.193        2225   Connected      hsm-jqici4covtv
 Logged in as 'alice(CO)'


aws-cloudhsm>info server 1

Id      Name             Hostname         Port   State           Partition
 LoginState
0       10.1.10.7        10.1.10.7        2225   Connected       hsm-ogi3sywxbqx
 Logged in as 'alice(CO)'
```

Alice は別のユーザー、John のパスワードをリセットしようと考えています。パスワードを変更する前に、listUsers コマンドを使用して John のユーザータイプを確認します。

次の出力では、CO ユーザーとして John が表示されています。

```
aws-cloudhsm> listUsers
```

```
Users on server 0(10.1.9.193):
Number of users found:5

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                   YES              0
             NO
         2              AU              jane                     NO              0
             NO
         3              CU              bob                      NO              0
             NO
         4              CU              alice                    NO              0
             NO
         5              CO              john                     NO              0
             NO
Users on server 1(10.1.10.7):
Number of users found:5

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                   YES              0
             NO
         2              AU              jane                     NO              0
             NO
         3              CU              bob                      NO              0
             NO
         4              CO              alice                    NO              0
             NO
         5              CO              john                     NO              0
             NO
```

パスワードを変更するために、Alice は changePswd に続けて、John のユーザータイプ、ユーザー名、および新しいパスワードを指定して実行します。

```
aws-cloudhsm>changePswd CO john newPassword

*************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****************************************************************
```

```
Do you want to continue(y/n)?y
Changing password for john(CO) on 2 nodes
```

引数

引数は構文の図表で指定された順序で入力します。-hpswd パラメータを使用して、パスワードをマスクします。CO ユーザーに対して 2FA を有効にするには、-2fa パラメータを入力し、ファイルパスを含めます。2FA での作業の詳細については、「CMU を使用した 2FA の管理」を参照してください。

```
changePswd <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

<user-type>

パスワードを変更しようとしているユーザーの現在のタイプを指定します。changePswd を使用してユーザーのタイプを変更することはできません。

有効な値は、CO、CU、PCO、および PRECO です。

ユーザータイプを取得するには、listUsers を使用します。HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

必須: はい

<user-name>

ユーザーのわかりやすい名前を指定します。このパラメータは大文字と小文字が区別されません。changePswd を使用してユーザー名を変更することはできません。

必須: はい

<password |-hpswd >

ユーザーの新しいパスワードを指定します。7 ~ 32 文字の文字列を入力します。この値では、大文字と小文字が区別されます。パスワードは、入力するとプレーンテキストで表示されます。パスワードを非表示にするには、-hpswd パラメータをパスワードの代わりに入力し、プロンプトに従います。

必須: はい

[-2fa </path/to/authdata>]

この CO ユーザーに対して 2FA を有効にすることを指定します。2FA の設定に必要なデータを取得するには、`-2fa` パラメータの後にファイル名でファイルシステム内の場所へのパスを記述します。2FA の操作の詳細については、「CMU を使用した 2FA の管理」を参照してください。

必須: いいえ

関連トピック

- info
- listUsers
- createUser
- deleteUser

createUser

cloudhsm_mgmt_util の createUser コマンドは、HSM 上にユーザーを作成します。Crypto Officer (CO および PRECO) だけがこのコマンドを実行できます。コマンドが成功すると、クラスター内のすべての HSM にユーザーが作成されます。

createUser のトラブルシューティングを行うには

HSM 設定が不正確な場合、一部の HSM でユーザーが作成されない場合があります。ユーザーが欠落している HSM にユーザーを追加するには、そのユーザーがいない HSM でのみ syncUser コマンドまたは createUser コマンドを使用します。設定エラーを防ぐには、configureツールを `-m` オプション付きで実行します。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto Officer (CO、PRECO)

構文

引数は構文の図表で指定された順序で入力します。-hpswd パラメータを使用して、パスワードをマスクします。2 要素認証 (2FA) を使用して CO ユーザーを作成するには、-2fa パラメータを入力し、ファイルパスを含めます。詳細については、「the section called “引数”」を参照してください。

```
createUser <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

例

以下の例では、createUser を使用して HSM に新しいユーザーを作成する方法を示します。

Example : Crypto Officer を作成する

次の例では、クラスター内の HSM に Crypto Officer (CO) を作成します。最初のコマンドは、loginHSM を使用して、Crypto Officer として HSM にログインします。

```
aws-cloudhsm> loginHSM CO admin 735782961

loginHSM success on server 0(10.0.0.1)
loginHSM success on server 1(10.0.0.2)
loginHSM success on server 1(10.0.0.3)
```

2 番目のコマンドでは、createUser コマンドを使用して、HSM 上に新しい Crypto Officer である alice を作成します。

注意メッセージは、コマンドがクラスター内のすべての HSM でユーザーを作成することを説明しています。ただし、コマンドがいずれかの HSM で失敗した場合、その HSM にユーザーは存在しません。続行するには、y と入力します。

出力は、クラスター内の 3 つすべての HSM で新しいユーザーが作成されたことを示しています。

```
aws-cloudhsm> createUser CO alice 391019314

************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************
```

```
Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'

Do you want to continue(y/n)?y
Creating User alice(CO) on 3 nodes
```

コマンドが完了すると、alice には、HSM 上のすべてのユーザーのパスワードを変更するなど、admin CO ユーザーと同じ HSM のアクセス許可が与えられます。

最後のコマンドでは、listUsers コマンドを使用して、クラスターの 3 つの HSM すべてに alice が存在することを検証します。出力は、alice にユーザー ID 3 が割り当てられていることも示しています。.ユーザー ID を使用して、 などの他のコマンドaliceで を識別しますfindAllKeys。

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name                      MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                               YES
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              alice                                NO
           0               NO
Users on server 1(10.0.0.2):
Number of users found:3

    User Id             User Type       User Name                      MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                               YES
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              alice                                NO
           0               NO

Users on server 1(10.0.0.3):
Number of users found:3

    User Id             User Type       User Name                      MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                               YES
           0               NO
```

```
        2              AU              app_user                       NO
          0               NO
        3              CO              alice                          NO
          0               NO
```

Example : Crypto User を作成する

次の例では、HSM に Crypto User (CU)、bob を作成します。Crypto Users はキーを作成して管理することはできますが、ユーザーを管理することはできません。

警告メッセージに「y」と入力して応答すると、クラスター内の 3 つすべての HSM に bob が作成されたことが出力に表示されます。新しい CU は、HSM にログインしてキーを作成および管理できます。

このコマンドでは、パスワード値として defaultPassword を使用しています。その後、bob または他の CO は changePswd コマンドを使用してパスワードを変更できます。

```
aws-cloudhsm> createUser CU bob defaultPassword

**************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************

Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'

Do you want to continue(y/n)?y
Creating User bob(CU) on 3 nodes
```

引数

引数は構文の図表で指定された順序で入力します。-hpswd パラメータを使用して、パスワードをマスクします。2FA を有効にした CO ユーザーを作成するには、-2fa パラメータを入力し、ファイルパスを含めます。2FA の詳細については、「CMU を使用した 2FA の管理」を参照してください。

```
createUser <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

<user-type>

ユーザーのタイプを指定します。このパラメータは必須です。

HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

有効値:

- CO: Crypto officers はユーザーを管理できますが、キーを管理することはできません。
- CU: Crypto User は、管理キーを作成し、暗号化オペレーションでキーを使用できます。

HSM のアクティベーション 中にパスワードを割り当てると、PRECO は CO に変換されます。

必須: はい

<user-name>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。

ユーザーの作成後にユーザー名を変更することはできません。cloudhsm_mgmt_util コマンドでは、ユーザータイプとパスワードは大文字と小文字が区別されますが、ユーザー名は区別されません。

必須: はい

<password |-hpswd >

ユーザーのパスワードを指定します。7 ~ 32 文字の文字列を入力します。この値は大文字と小文字が区別されます。パスワードは、入力するとプレーンテキストで表示されます。パスワードを非表示にするには、-hpswd パラメータをパスワードの代わりに入力し、プロンプトに従います。

ユーザーパスワードを変更するには、changePswd を使用します。HSM ユーザーはすべて自分のパスワードを変更できますが、CO ユーザーは HSM のどのタイプのどのユーザーのパスワードでも変更できます。

必須: はい

[-2fa </path/to/authdata>]

2FA が有効な CO ユーザーの作成を指定します。2FA認証の設定に必要なデータを取得するには、-2fa パラメータの後にファイル名でファイルシステム内の場所へのパスを含めます。2FA のセットアップおよび使用の詳細については、「CMU を使用した 2FA の管理」を参照してください。

必須: いいえ

関連トピック

- listUsers
- deleteUser
- syncUser
- changePswd

deleteUser

cloudhsm_mgmt_utilの deleteUser コマンドは、ハードウェアセキュリティモジュール（HSM）からユーザーを削除します。このコマンドを実行できるのは、Crypto Officer（CO）のみです。現在 HSM にログインしているユーザーを削除することはできません。ユーザーの削除の詳細については、「HSM ユーザーを削除する方法」を参照してください。

> **Tip**
> キーを所有している暗号化ユーザー (CU) を削除することはできません。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- CO

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
deleteUser <user-type> <user-name>
```

例

次の例では、クラスター内の HSM から Crypto Officer (CO) を削除します。最初のコマンドでは、listUsers を使用して HSM のすべてのユーザーを一覧表示します。

出力は、ユーザー 3、alice が HSM の CO であることを示しています。

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                               YES
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              alice                                NO
           0               NO
Users on server 1(10.0.0.2):
Number of users found:3

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                               YES
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              alice                                NO
           0               NO

Users on server 1(10.0.0.3):
Number of users found:3

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                               YES
           0               NO
         2              AU              app_user                             NO
           0               NO
         3              CO              alice                                NO
           0               NO
```

2 番目のコマンドでは、deleteUser コマンドを使用して HSM から alice を削除します。

出力は、クラスター内の 3 つすべての HSM でコマンドが成功したことを示しています。

```
aws-cloudhsm> deleteUser CO alice
Deleting user alice(CO) on 3 nodes
deleteUser success on server 0(10.0.0.1)
```

```
deleteUser success on server 0(10.0.0.2)
deleteUser success on server 0(10.0.0.3)
```

最後のコマンドでは、listUsers コマンドを使用して alice がクラスターの 3 つの HSM すべてから削除されていることを検証します。

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:2

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
          1             PCO             admin                                YES
            0              NO
          2             AU              app_user                              NO
            0              NO
Users on server 1(10.0.0.2):
Number of users found:2

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
          1             PCO             admin                                YES
            0              NO
          2             AU              app_user                              NO
            0              NO
Users on server 1(10.0.0.3):
Number of users found:2

    User Id             User Type       User Name                       MofnPubKey
  LoginFailureCnt         2FA
          1             PCO             admin                                YES
            0              NO
          2             AU              app_user                              NO
            0              NO
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
deleteUser <user-type> <user-name>
```

<user-type>

ユーザーのタイプを指定します。このパラメータは必須です。

> **Tip**
>
> キーを所有している暗号化ユーザー (CU) を削除することはできません。

有効な値は CO、CU。

ユーザータイプを取得するには、listUsers を使用します。HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

必須: はい

<user-name>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。

ユーザーの作成後にユーザー名を変更することはできません。cloudhsm_mgmt_util コマンドでは、ユーザータイプとパスワードは大文字と小文字が区別されますが、ユーザー名は区別されません。

必須: はい

関連トピック

- listUsers
- createUser
- syncUser
- changePswd

findAllKeys

cloudhsm_mgmt_util の findAllKeys コマンドで、指定した Crypto User (CU) が所有または共有しているキーを取得します。HSM のそれぞれのユーザーデータのハッシュも返されます。ハッシュを使用して、ユーザー、キーの所有権、およびキー共有データがクラスター内のすべての HSM で同じ

かどうかを一目で判断できます。出力では、ユーザーが所有するキーは (o) によって注釈が付けられ、共有キーは (s) によって注釈が付けられます。

HSM のすべての CU が任意のパブリックキーを使用できますが、findAllKeys がパブリックキーを返すのは、指定した CU がそのキーを所有している場合のみです。この動作は、すべての CU ユーザーに公開キーを返す key_mgmt_util の findKey とは異なります。

Crypto Officer (CO および PCO) と Appliance User (AU) のみがこのコマンドを実行できます。Crypto User (CU) は、次のコマンドを実行することができます。

- listUsers – すべてのユーザーを検索する
- key_mgmt_util の findKey で、使用できるキーを見つけます。
- getKeyInfo key_mgmt_util で、所有または共有する特定のキーの所有者と共有ユーザーを検索する

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- Crypto Officer (CO、PCO)
- Appliance User (AU)

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

例

以下の例では、findAllKeys を使用してユーザーのすべてのキーを検索し、HSM のそれぞれのキーユーザー情報のハッシュを取得する方法を示します。

Example : CU のキーを検索する

次の例では、findAllKeys を使用して、ユーザー 4 が所有および共有している HSM のキーを検索します。このコマンドでは、2 番目の引数に値 0 を使用してハッシュ値を抑制します。オプションのファイル名が省略されるため、コマンドは stdout に書き込みを行います (標準出力)。

出力は、ユーザー 4 が 6 つのキー (8、9、17、262162、19、および 31) を使用できることを示しています。出力では、(s) を使用して、ユーザーが明示的に共有するキーが示されます。ユーザーが所有するキーは (o) によって示され、ユーザが共有しない対称キーとプライベートキー、およびすべての Crypto User が使用できる公開キーが含まれます。

```
aws-cloudhsm> findAllKeys 4 0
Keys on server 0(10.0.0.1):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 0(10.0.0.1)

Keys on server 1(10.0.0.2):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.2)

Keys on server 1(10.0.0.3):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.3)
```

Example : ユーザーデータが同期されていることを検証する

次の例では findAllKeys を使用して、クラスター内の HSM のすべてが同じユーザー、キーの所有者、およびキー共有の値を含んでいることを検証します。これを行うため、各 HSM のキーユーザーデータのハッシュを取得し、ハッシュ値を比較します。

キーハッシュを取得するため、このコマンドでは 2 番目の引数に値 1 を使用します。オプションのファイル名が省略されるため、コマンドはキーハッシュを stdout に書き込みます。

この例ではユーザー 6 を指定しますが、HSM のキーを所有または共有するどのユーザーに対してもハッシュ値は同じです。指定したユーザーが CO などのキーを所有または共有していない場合、コマンドはハッシュ値を返しません。

出力は、キーハッシュがクラスター内の両方の HSM と同じであることを示しています。HSM の 1 つに異なるユーザー、異なるキー所有者、または異なる共有ユーザーがいた場合、キーハッシュ値は同じにはなりません。

```
aws-cloudhsm> findAllKeys 6 1
Keys on server 0(10.0.0.1):
Number of keys found 3
number of keys matched from start index 0::3
8(s),9(s),11,17(s)
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 0(10.0.0.1)
Keys on server 1(10.0.0.2):
Number of keys found 3
number of keys matched from start index 0::3
8(s),9(s),11(o),17(s)
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 1(10.0.0.2)
```

次のコマンドは、ハッシュ値が HSM のすべてのキーのユーザーデータを表すことを示しています。このコマンドでは、ユーザー 3 に対して findAllKeys を使用します。3 つのキーだけを所有または共有しているユーザー 6 とは異なり、ユーザー 3 は 17 個のキーを所有または共有していますが、キーハッシュ値は同じです。

```
aws-cloudhsm> findAllKeys 3 1
Keys on server 0(10.0.0.1):
Number of keys found 17
number of keys matched from start index 0::17
6(o),7(o),8(s),11(o),12(o),14(o),262159(o),262160(o),17(s),262162(s),19(s),20(o),21(o),262177(o
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 0(10.0.0.1)
Keys on server 1(10.0.0.2):
Number of keys found 17
number of keys matched from start index 0::17
6(o),7(o),8(s),11(o),12(o),14(o),262159(o),262160(o),17(s),262162(s),19(s),20(o),21(o),262177(o
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49
```

```
findAllKeys success on server 1(10.0.0.2)
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

<user id>

指定したユーザーが所有または共有しているすべてのキーを取得します。HSM のユーザーのユーザー ID を入力します。すべてのユーザーのユーザー ID を検索するには、listUsers を使用します。

すべてのユーザー ID が有効ですが、`findAllKeys` は Crypto User (CU) のキーのみを返します。

必須: はい

<key hash>

各 HSM のすべてのキーのユーザーの所有権とデータ共有のハッシュを含める (1) か除外 (0) します。

`user id` 引数がキーを所有または共有しているユーザーを表す場合、キーハッシュが入力されます。異なるキーを所有または共有していても、キーハッシュ値は、HSM でキーを所有または共有しているすべてのユーザーで同じです。ただし、`user id` が CO などのキーを所有または共有していないユーザーを表す場合、ハッシュ値は入力されません。

必須: はい

<output file>

指定したファイルに出力を書き込みます。

必須: いいえ

デフォルト: Stdout

関連トピック

- changePswd
- deleteUser
- listUsers
- syncUser
- key_mgmt_util で findKey
- getKeyInfo key_mgmt_util の

getAttribute

cloudhsm_mgmt_util の getAttribute コマンドは、クラスター内のすべての HSM からキーの属性値を 1 つ取得し、stdout (標準出力) またはファイルに書き込みます。このコマンドを実行できるのは Crypto User (CU) のみです。

キー属性はキーのプロパティです。キー属性には、キータイプ、クラス、ラベル、ID などの特性と、キーに対して実行できるアクション (暗号化、復号、ラップ、署名、検証など) を表す値が含まれています。

getAttribute は、所有しているキーと共有しているキーに対してのみ使用できます。このコマンド、または、key_mgmt_util の getAttribute コマンドを実行し、キーの属性値の 1 つまたはすべてをファイルに書き込むことができます。

属性とそれを表す定数のリストを取得するには、listAttributes コマンドを使用します。既存のキーの属性値を変更するには、key_mgmt_util の setAttribute および cloudhsm_mgmt_util の setAttribute を使用します。キー属性の解釈については、「キー属性リファレンス」を参照してください。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- Crypto User (CU)

Syntax

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
getAttribute <key handle> <attribute id> [<filename>]
```

例

次の例では、HSM でキーの抽出可能な属性の値を取得します。次のようなコマンドを使用して HSM からキーをエクスポートできるかどうかを判断できます。

最初のコマンドでは、listAttributes を使用して抽出可能な属性を表す定数を見つけます。出力は、OBJ_ATTR_EXTRACTABLE の定数が 354 であることを示しています。この情報は、「キー属性リファレンス」の属性と値の説明を使用して検索することもできます。

```
aws-cloudhsm> listAttributes

Following are the possible attribute values for getAttribute:

      OBJ_ATTR_CLASS                = 0
      OBJ_ATTR_TOKEN                = 1
      OBJ_ATTR_PRIVATE              = 2
      OBJ_ATTR_LABEL                = 3
      OBJ_ATTR_TRUSTED              = 134
      OBJ_ATTR_KEY_TYPE             = 256
      OBJ_ATTR_ID                   = 258
      OBJ_ATTR_SENSITIVE            = 259
      OBJ_ATTR_ENCRYPT              = 260
      OBJ_ATTR_DECRYPT              = 261
      OBJ_ATTR_WRAP                 = 262
      OBJ_ATTR_UNWRAP               = 263
      OBJ_ATTR_SIGN                 = 264
      OBJ_ATTR_VERIFY               = 266
      OBJ_ATTR_DERIVE               = 268
      OBJ_ATTR_LOCAL                = 355
      OBJ_ATTR_MODULUS              = 288
      OBJ_ATTR_MODULUS_BITS         = 289
      OBJ_ATTR_PUBLIC_EXPONENT      = 290
      OBJ_ATTR_VALUE_LEN            = 353
      OBJ_ATTR_EXTRACTABLE          = 354
      OBJ_ATTR_NEVER_EXTRACTABLE    = 356
```

```
      OBJ_ATTR_ALWAYS_SENSITIVE       = 357
      OBJ_ATTR_DESTROYABLE            = 370
      OBJ_ATTR_KCV                    = 371
      OBJ_ATTR_WRAP_WITH_TRUSTED      = 528
      OBJ_ATTR_WRAP_TEMPLATE          = 1073742353
      OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
      OBJ_ATTR_ALL                    = 512
```

2 番目のコマンドは、getAttribute を使用して HSM でキーハンドルが 262170 であるキーの抽出可能な属性の値を取得します。抽出可能な属性を指定するために、コマンドは 354 (属性を表す定数) を使用します。このコマンドではファイル名が指定されないため、getAttribute は出力を stdout に書き込みます。

出力は、HSM のすべてにおいて抽出可能な属性の値が 1 であることを示しています。この値は、キーの所有者がキーをエクスポートできることを示します。値が 0 (0x0) であれば、HSM からキーをエクスポートすることはできません。抽出可能な属性の値は、キーの作成時に設定できますが、変更することはできません。

```
aws-cloudhsm> getAttribute 262170 354

Attribute Value on server 0(10.0.1.10):
OBJ_ATTR_EXTRACTABLE
0x00000001

Attribute Value on server 1(10.0.1.12):
OBJ_ATTR_EXTRACTABLE
0x00000001

Attribute Value on server 2(10.0.1.7):
OBJ_ATTR_EXTRACTABLE
0x00000001
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
getAttribute <key handle> <attribute id> [<filename>]
```

<key-handle>

ターゲットキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、key_mgmt_util の findKey を使用します。

指定するキーは所有しているか、共有している必要があります。キーのユーザーを検索するには、key_mgmt_util getKeyInfoで を使用します。

必須: はい

<attribute id>

属性を識別します。属性を表す定数を入力するか、すべての属性を表す 512 を入力します。たとえば、キーの種類を取得するには「256」と入力します。これは OBJ_ATTR_KEY_TYPE 属性を表す定数です。

属性とその定数のリストアップするには、listAttributes を使用します。キー属性の解釈については、キー属性リファレンス を参照してください。

必須: はい

<filename>

指定したファイルに出力を書き込みます。ファイルパスを入力します。

指定したファイルが既に存在する場合、getAttribute は警告なしにそのファイルを上書きします。

必須: いいえ

デフォルト: Stdout

関連トピック

- key_mgmt_util で getAttribute
- listAttributes
- cloudhsm_mgmt_util で setAttribute
- key_mgmt_util で setAttribute
- キー属性リファレンス

getCert

cloudhsm_mgmt_util の getCert コマンドを使用すると、クラスター内の特定の HSM の証明書を取得することができます。コマンドを実行する際、取得する証明書のタイプを指定します。そのためには、以下の「引数」セクションで説明されているように、いずれかの整数を使用します。これらの各証明書のロールについては、「HSM のアイデンティティの確認」を参照してください。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。

前提条件

始めるには、送信先 HSM のサーバーモードを入力する必要があります。詳細については、「サーバー」を参照してください。

構文

サーバーモードで、getCert コマンドを 1 回使用するには:

```
server> getCert <file-name> <certificate-type>
```

例

まず、サーバーモードを入力します。このコマンドでは、サーバー番号 `0` の HSM のサーバーモードを入力します。

```
aws-cloudhsm> server 0

Server is in 'E2' mode...
```

次に、getCert コマンドを使用します。この例では、証明書を保存するファイルの名前として `/tmp/P0.crt`、目的の証明書タイプとして `4` (お客様のルート証明書) を使用します。

```
server0> getCert /tmp/PO.crt 4
getCert Success
```

引数

```
getCert <file-name> <certificate-type>
```

<file-name>

証明書が保存されるファイルの名前を指定します。

必須: はい

<certificate-type>

取得する証明書のタイプを指定する整数。整数とその証明書タイプは次のとおりです。

- 1 – 製造元のルート証明書
- 2 – 製造元のハードウェア証明書
- 4 – お客様のルート証明書
- 8 – クラスターの証明書 (お客様のルート証明書で署名されている)
- 16 – クラスターの証明書 (製造元のルート証明書に連鎖されている)

必須: はい

関連トピック

- サーバー

getHSMInfo

cloudhsm_mgmt_util の getHSMInfo コマンドは、モデル、シリアル番号、FIPS 状態、メモリ、温度、ハードウェアとファームウェアのバージョン番号など、各 HSM が実行されているハードウェアに関する情報を取得します。この情報には、cloudhsm_mgmt_util が HSM を参照するために使用するサーバー ID も含まれます。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- すべてのユーザー。このコマンドを実行するのに、ログインする必要はありません。

構文

このコマンドにはパラメータはありません。

```
getHSMInfo
```

例

この例では、getHSMInfo を使用して、クラスター内の HSM に関する情報を取得します。

```
aws-cloudhsm> getHSMInfo
Getting HSM Info on 3 nodes
                *** Server 0 HSM Info ***

        Label                :cavium
        Model                :NITROX-III CNN35XX-NFBE

        Serial Number        :3.0A0101-ICM000001
        HSM Flags            :0
        FIPS state           :2 [FIPS mode with single factor authentication]

        Manufacturer ID      :
        Device ID            :10
        Class Code           :100000
        System vendor ID     :177D
        SubSystem ID         :10


        TotalPublicMemory    :560596
        FreePublicMemory     :294568
        TotalPrivateMemory   :0
        FreePrivateMemory    :0

        Hardware Major       :3
```

```
        Hardware Minor       :0

        Firmware Major       :2
        Firmware Minor       :03

        Temperature          :56 C

        Build Number         :13

        Firmware ID          :xxxxxxxxxxxxxxx

...
```

関連トピック

- info

getKeyInfo

key_mgmt_util の getKeyInfo コマンドは、キーを使用できるユーザーの HSM ユーザー ID を返します。これには、所有者およびキーを共有する暗号ユーザー (CU) も含まれます。キーに対するクォーラム認証が有効になっている場合、getKeyInfo はキーを使用する暗号化オペレーションを承認する必要があるユーザーの数も返します。getKeyInfo は、所有および共有しているキーに対してのみ実行できます。

パブリックキーに対して getKeyInfo を実行すると、HSM のすべてのユーザーがパブリックキーを使用できる場合でも、getKeyInfo はキー所有者のみを返します。HSM のユーザーの HSM ユーザー ID を確認するには、listUsers を使用します。特定のユーザーのキーを見つけるには、key_mgmt_util の findKey -u を使用します。Crypto Officer は cloudhsm_mgmt_util findAllKeysで を使用できます。

ユーザーは、自分で作成したキーを所有します。自分で作成したキーは、他のユーザーと共有できます。既存のキーを共有または共有解除するには、cloudhsm_mgmt_util の shareKey を使用します。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

構文

```
getKeyInfo -k <key-handle> [<output file>]
```

例

以下の例では、getKeyInfo を使用してキーのユーザーに関する情報を取得する方法を示します。

Example : 非対称キーのユーザーを取得する

次のコマンドでは、キーハンドルが 262162 の AES (非対称) キーを使用できるユーザーを取得します。出力は、キーの所有者がユーザー 3 であり、キーをユーザー 4 および 6 と共有していることを示しています。

ユーザー 3、4、および 6 のみが、キー 262162 に対して getKeyInfo を実行できます。

```
aws-cloudhsm>getKeyInfo 262162
Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 2 user(s):

                4
                6
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 2 user(s):

                4
                6
```

Example : 対称キーペアのユーザーを取得する

以下のコマンドでは、getKeyInfo を使用して ECC (対称) キーペアのキーを使用できるユーザーを取得します。パブリックキーのキーハンドルは 262179 です。プライベートキーのキーハンドルは 262177 です。

プライベートキー (262177) に対して getKeyInfo を実行すると、キー所有者 (3) とキーを共有している Crypto User (CU) 4 が返されます。

```
aws-cloudhsm>getKeyInfo -k 262177
Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                4
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                4
```

パブリックキー (262179) に対して getKeyInfo を実行すると、キー所有者であるユーザー 3 のみが返されます。

```
aws-cloudhsm>getKeyInfo -k 262179
Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

Key Info on server 1(10.0.3.6):

        Token/Flash Key,
```

```
        Owned by user 3
```

ユーザー 4 がパブリックキー (および HSM のすべての公開キー) を使用できることを確認するには、key_mgmt_util の findKey の -u パラメータを使用します。

出力は、ユーザー 4 がキーペアのパブリックキー (262179) とプライベートキー (262177) の両方を使用できることを示しています。ユーザー 4 は、他のすべてのパブリックキーと、自分で作成したプライベートキーまたは共有しているプライベートキーを使用することもできます。

```
Command:  findKey -u 4

Total number of keys present 8

 number of keys matched from start index 0::7
11, 12, 262159, 262161, 262162, 19, 20, 21, 262177, 262179

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : キーのクォーラム認証値 (m_value) を取得する

次の例では、キューの m_value を取得する方法を示します。m_value は、キーを使用する暗号化オペレーションとキーを共有/共有解除するオペレーションを承認するクォーラム内のユーザーの数です。

キーに対してクォーラム認証を有効にすると、ユーザーのクォーラムは、そのキーを使用する暗号化オペレーションを承認する必要があります。クォーラム認証を有効にしてクォーラムサイズを設定するには、キーの作成時に -m_value パラメータを使用します。

このコマンドは、 genSymKeyを使用して、ユーザー 4 と共有される 256 ビット AES キーを作成します。また、m_value パラメータを使用してクォーラム認証を有効にし、クォーラムサイズを 2 ユーザーに設定します。ユーザー数は必要な承認を提供できるだけの大きさが必要です。

出力は、このコマンドでキー 10 が作成されたことを示しています。

```
Command:  genSymKey -t 31 -s 32 -l aes256m2 -u 4 -m_value 2

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
```

```
        Symmetric Key Created.  Key Handle: 10

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

このコマンドは、cloudhsm_mgmt_util の getKeyInfo を使用してキーのユーザー 10 に関する情報を取得します。出力は、キーの所有者がユーザー 3 であり、キーがユーザー 4 と共有されていることを示しています。また、2 ユーザーのクォーラムが、このキーを使用するすべての暗号化オペレーションを承認する必要があることも示しています。

```
aws-cloudhsm>getKeyInfo 10

Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                4
         2 Users need to approve to use/manage this key
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                4
         2 Users need to approve to use/manage this key
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
getKeyInfo -k <key-handle> <output file>
```

<key-handle>

HSM で 1 つのキーのキーハンドルを指定します。所有または共有するキーのキーハンドルを入力します。このパラメータは必須です。

必須: はい

<output file>

出力の書き込み先を stdout ではなく、指定したファイルにします。既存のファイルがある場合は、警告なしに上書きされます。

必須: いいえ

デフォルト: stdout

関連トピック

- getKeyInfo key_mgmt_util の
- key_mgmt_util で findKey
- findAllKeys cloudhsm_mgmt_util の
- listUsers
- shareKey

info

cloudhsm_mgmt_util の info コマンドは、ホスト名、ポート、IPアドレス、および HSM 上の cloudhsm_mgmt_util にログインしているユーザーの名前とタイプを含むクラスタ内の各 HSM に関する情報を取得します。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- すべてのユーザー。このコマンドを実行するのに、ログインする必要はありません。

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
info server <server ID>
```

例

この例では、info を使用して、クラスター内の HSM に関する情報を取得します。コマンドは、0 を使用してクラスター内の最初の HSM を参照します。出力は、IP アドレス、ポート、および現在のユーザーのタイプと名前を表示します。

```
aws-cloudhsm> info server 0
Id      Name                    Hostname         Port   State           Partition
        LoginState
0       10.0.0.1                10.0.0.1        2225    Connected       hsm-udw0tkfg1ab
        Logged in as 'testuser(CU)'
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
info server <server ID>
```

<server id>

HSM のサーバー ID を指定します。HSM には、クラスターに追加された順番を示す序数 (0 から始まる) が割り当てられます。HSM のサーバー ID を見つけるには、getHSMInfo を使用します。

必須: はい

関連トピック

- getHSMInfo
- loginHSM および logoutHSM

listAttributes

cloudhsm_mgmt_util の listAttributes コマンドは、 AWS CloudHSM キーの属性とそれを表す定数を一覧表示します。これらの定数は、getAttribute コマンドおよび setAttribute コマンドの属性を特定するのに使用します。

キー属性の解釈については、キー属性リファレンス を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動 し、HSM に Crypto ユーザー (CU) として ログイン する必要があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。このコマンドを実行するのに、ログインする必要はありません。

Syntax

```
listAttributes [-h]
```

例

このコマンドは、key_mgmt_utilで取得および変更できるキー属性と、それらを表す定数を一覧表示します。キー属性の解釈については、「キー属性リファレンス」を参照してください。すべての属性を表すには、 512 を使用します

```
Command: listAttributes

    Description
===========
The following are all of the possible attribute values for getAttribute.

      OBJ_ATTR_CLASS                = 0
      OBJ_ATTR_TOKEN                = 1
      OBJ_ATTR_PRIVATE              = 2
      OBJ_ATTR_LABEL                = 3
      OBJ_ATTR_TRUSTED              = 134
      OBJ_ATTR_KEY_TYPE             = 256
      OBJ_ATTR_ID                   = 258
      OBJ_ATTR_SENSITIVE            = 259
      OBJ_ATTR_ENCRYPT              = 260
```

```
        OBJ_ATTR_DECRYPT                = 261
        OBJ_ATTR_WRAP                   = 262
        OBJ_ATTR_UNWRAP                 = 263
        OBJ_ATTR_SIGN                   = 264
        OBJ_ATTR_VERIFY                 = 266
        OBJ_ATTR_DERIVE                 = 268
        OBJ_ATTR_LOCAL                  = 355
        OBJ_ATTR_MODULUS                = 288
        OBJ_ATTR_MODULUS_BITS           = 289
        OBJ_ATTR_PUBLIC_EXPONENT        = 290
        OBJ_ATTR_VALUE_LEN              = 353
        OBJ_ATTR_EXTRACTABLE            = 354
        OBJ_ATTR_NEVER_EXTRACTABLE      = 356
        OBJ_ATTR_ALWAYS_SENSITIVE       = 357
        OBJ_ATTR_DESTROYABLE            = 370
        OBJ_ATTR_KCV                     = 371
        OBJ_ATTR_WRAP_WITH_TRUSTED       = 528
        OBJ_ATTR_WRAP_TEMPLATE           = 1073742353
        OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
        OBJ_ATTR_ALL                    = 512
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

関連トピック

- getAttribute
- setAttribute
- キー属性リファレンス

listUsers

cloudhsm_mgmt_util の listUsers コマンドは、各 HSM のユーザーを、そのユーザータイプや他の属性とともに取得します。このコマンドは、すべてのユーザータイプで実行できます。このコマンドを実行するのに、ログインする必要はありません。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

構文

このコマンドにはパラメータはありません。

```
listUsers
```

例

このコマンドは、クラスター内の HSM ごとにユーザーとその属性を一覧表示します。User ID 属性を使用して、deleteUser、changePswd、findAllKeys などの他のコマンドでユーザーを特定できます。

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:6

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
        1               PCO             admin                   YES              0
            NO
        2               AU              app_user                 NO              0
            NO
        3               CU              crypto_user1             NO              0
            NO
        4               CU              crypto_user2             NO              0
            NO
        5               CO              officer1                YES              0
            NO
        6               CO              officer2                 NO              0
            NO
```

```
Users on server 1(10.0.0.2):
Number of users found:5

   User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
        1              PCO             admin                   YES              0
            NO
        2              AU              app_user                 NO              0
            NO
        3              CU              crypto_user1             NO              0
            NO
        4              CU              crypto_user2             NO              0
            NO
        5              CO              officer1                YES              0
            NO
```

この出力が示すユーザー属性は以下のとおりです。

- ユーザー ID: key_mgmt_util および cloudhsm_mgmt_util のコマンドでユーザーを識別します。
- User type: HSM でユーザーが実行できるオペレーションを決定します。
- User Name: ユーザー定義のわかりやすいユーザー名を表示します。
- MofnPubKey: ユーザーがクォーラム認証トークン に署名するためのキーペアを登録したかどうかを示します。
- LoginFailureCnt: ユーザーがログインに失敗した回数を示します。
- 2FA: ユーザーが多要素認証を有効にしていることを示します。

関連トピック

- key_mgmt_util で listUsers
- createUser
- deleteUser
- changePswd

loginHSM および logoutHSM

クラスターの各 HSM でログインおよびログアウトを行うには、cloudhsm_mgmt_util の loginHSM および logoutHSM コマンドを使用します。タイプに関係なく、すべてのユーザーがこのコマンドを使用できます。

> **Note**
>
> ログイン試行回数が 5 回を超えると、アカウントがロックアウトされます。アカウントのロックを解除するには、暗号化オフィサー (CO) が cloudhsm_mgmt_util で changePswd コマンドを使用してパスワードをリセットする必要があります。

loginHSM および logoutHSM のトラブルシューティングを行う場合

これらの cloudhsm_mgmt_util コマンドを実行する前に、cloudhsm_mgmt_util を起動する必要があります。

HSMs を追加または削除する場合は、 AWS CloudHSM クライアントとコマンドラインツールが使用する設定ファイルを更新します。そうしないと、クラスタ内のすべての HSM で変更が有効にならない場合があります。

クラスター内に複数の HSM がある場合は、アカウントがロックアウトされるまでのログイン試行回数の上限が増える可能性があります。これは、CloudHSM クライアントがさまざまな HSM 間で負荷を分散するためです。したがって、ログイン試行は毎回同じ HSM で開始されない場合があります。この機能をテストしている場合は、アクティブな HSM が1つだけのクラスターでテストすることをお勧めします。

2018 年 2 月より前にクラスターを作成した場合、ロックアウトされるまでのログイン試行回数は 20 回です。

ユーザーのタイプ

これらのコマンドは、次のユーザーが実行できます。

- PreCrypto Officer (PRECO)
- Crypto Officer (CO)
- Crypto user (CU)

構文

引数は構文の図表で指定された順序で入力します。`-hpswd` パラメータを使用して、パスワードをマスクします。2 要素認証 (2FA) でログインするには、`-2fa` パラメータを入力し、ファイルパスを含めます。詳細については、「the section called “引数”」を参照してください。

```
loginHSM <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

```
logoutHSM
```

例

これらの例では、loginHSM および logoutHSM を使用して、クラスターのすべての HSM のログインおよびログアウトを行う方法を示します。

Example : クラスター内の HSM にログインする

このコマンドでは、CO ユーザー admin とパスワード co12345 の認証情報を使用して、クラスターのすべての HSM にログインします。出力には、コマンドが成功し、HSM(この場合は server 0 と server 1) に接続したことが示されています。

```
aws-cloudhsm>loginHSM CO admin co12345

loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
```

Example : 隠しパスワードでログインします

このコマンドは上記の例と同じですが、今回はシステムがパスワードを隠すように指定することを除きます。

```
aws-cloudhsm>loginHSM CO admin -hpswd
```

システムからパスワードの入力を求められます。パスワードを入力すると、システムはパスワードを非表示にし、コマンドが正常に実行されたことと HSM に接続したことが出力 (と) で示されます。

```
Enter password:

loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)

aws-cloudhsm>
```

Example : HSM からログアウトする

このコマンドは、現在ログインしている HSM (この場合は `server 0` と `server 1`) からログアウトします。出力は、コマンドが成功し、HSM から切断されたことを示します。

```
aws-cloudhsm>logoutHSM

logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
```

引数

引数は構文の図表で指定された順序で入力します。-hpswd パラメータを使用して、パスワードをマスクします。2 要素認証 (2FA) でログインするには、-2fa パラメータを入力し、ファイルパスを含めます。2FA での作業の詳細については、「CMU を使用した 2FA の管理」を参照してください。

```
loginHSM <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

<user type>

HSM にログインしているユーザーのタイプを指定します。詳細については、上記の「ユーザータイプ」を参照してください。

必須: はい

<user name>

HSM にログインしているユーザーのユーザー名を指定します。

必須: はい

<password |-hpswd >

HSM にログインしているユーザーのパスワードを指定します。パスワードを非表示にするには、-hpswd パラメータをパスワードの代わりに入力し、プロンプトに従います。

必須: はい

[-2fa </path/to/authdata>]

この 2FA 対応の CO ユーザーを認証するために、システムが第 2 要素を使用するように指定します。2FA でログインするために必要なデータを取得するには、-2fa パラメータの後にファイル名を指定して、ファイルシステム内の場所へのパスを含めます。2FA の操作の詳細については、「CMU を使用した 2FA の管理」を参照してください。

必須: いいえ

関連トピック

- cloudhsm_mgmt_util の起動方法
- クラスターのアクティブ化

registerQuorumPubキー

cloudhsm_mgmt_util registerQuorumPubKey のコマンドは、ハードウェアセキュリティモジュール（HSM）ユーザーを非対称の RSA-2048 キーペアに関連付けます。HSM ユーザーをキーに関連付けると、それらのユーザーはプライベートキーを使用してクォーラム要求を承認することができ、クラスターは登録された公開キーを使用して、署名がユーザーからのものであることを確認できます。クォーラム認証の詳細については、「クォーラム認証の管理（M of N アクセス制御）」を参照してください。

> **Tip**
>
> AWS CloudHSM ドキュメントでは、クォーラム認証は M of N (MofN) と呼ばれることがあります。これは、N 人の承認者の総数のうち最低 M 人の承認者を意味します。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto Officer (CO)

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-token> <public-key>
```

例

この例では registerQuorumPubKey を使用して、クォーラム認証の要求に対する承認者として Crypto Officer (CO) を登録する方法を示します。このコマンドを実行するには、非対称 RSA-2048 キーペア、署名付きトークン、および署名なしトークンが必要です。これらの要件の詳細については、「the section called “引数”」を参照してください。

Example : HSM ユーザーをクォーラム認証に登録する

この例では、クォーラム認証の承認者として quorum_officer という CO を登録します。

```
aws-cloudhsm> registerQuorumPubKey CO <quorum_officer> </path/to/quorum_officer.token>
 </path/to/quorum_officer.token.sig> </path/to/quorum_officer.pub>

************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
**************************************************************

Do you want to continue(y/n)?y
registerQuorumPubKey success on server 0(10.0.0.1)
```

最後のコマンドは listUsers コマンドを使用して、quorum_officer が MofN ユーザーとして登録されていることを確認します。

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
        1               PCO             admin                                   NO
          0               NO
        2               AU              app_user                                NO
          0               NO
        3               CO              quorum_officer                         YES
          0               NO
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-token> <public-key>
```

<user-type>

ユーザーのタイプを指定します。このパラメータは必須です。

HSM のユーザータイプの詳細については、「HSM ユーザーを理解する」を参照してください。

有効値:

- CO: Crypto officers はユーザーを管理できますが、キーを管理することはできません。

必須: はい

<user-name>

ユーザーのわかりやすい名前を指定します。最大長は 31 文字です。許可されている唯一の特殊文字はアンダースコア (_) です。

ユーザーの作成後にユーザー名を変更することはできません。cloudhsm_mgmt_util コマンドでは、ユーザータイプとパスワードは大文字と小文字が区別されますが、ユーザー名は区別されません。

必須: はい

<registration-token>

署名なし登録トークンを含むファイルへのパスを指定します。最大ファイルサイズが 245 バイトの任意のランダムデータを持つことができます。署名なしの登録トークンの作成についての詳細は、「登録トークンの作成と署名」を参照してください。

必須: はい

<signed-registration-token>

登録トークンの SHA256_PKCS メカニズム署名付きハッシュを含むファイルへのパスを指定します。詳細については、「登録トークンの作成と署名」を参照してください。

必須: はい

<public-key>

非対称 RSA-2048 キーペアの公開キーを含むファイルへのパスを指定します。プライベートキーを使用して、登録トークンに署名します。詳細については、「RSA キーペアの作成」を参照してください。

必須: はい

> **Note**
>
> クラスターは、クォーラム認証と 2 要素認証 (2FA) に同じキーを使用します。つまり、registerQuorumPubKey を使用して 2FA が有効になっているユーザーのクォーラムキーをローテーションすることはできません。キーをローテーションするには、changePswd のようにします。クォーラム認証と 2FA の使用の詳細については、「クォーラム認証と 2FA」を参照してください。

関連トピック

- RSA キーペアの作成
- 登録トークンの作成と署名
- HSM で公開キーを登録する
- クォーラム認証 (M of N アクセス制御) の管理
- クォーラム認証と 2FA
- listUsers

server

通常、cloudhsm_mgmt_util でコマンドを発行すると、コマンドは指定されたクラスター (グローバルモード) 内のすべての HSM に影響を及ぼします。ただし、単一の HSM にコマンドを発行する必要がある場合があります。たとえば、自動同期に失敗した場合は、クラスター全体の一貫性を維持するために、HSM 上のキーとユーザーを同期する必要がある場合があります。cloudhsm_mgmt_util 内の server コマンドを使用すると サーバーモードになり、特定の HSM インスタンスと直接対話することができます。

初期化が成功すると、`aws-cloudhsm>` コマンドプロンプトは、`server>` コマンドプロンプトに置き換わります。

サーバーモードを終了するには、exit コマンドを使用します。正常に終了すると、cloudhsm_mgmt_util のコマンドプロンプトに戻ります。

cloudhsm_mgmt_util コマンドを実行する前に、cloudhsm_mgmt_util を起動する必要があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。

前提条件

サーバーモードを入力するには、まず送信先 HSM のサーバー数を知る必要があります。サーバー番号は、cloudhsm_mgmt_util が起動時に生成するトレース出力に記載されています。サーバー数は、設定ファイルに表示されている HSM と同じ順序で表示されます。この例では、server 0 が、目的の HSM に対応するサーバーであることを前提としています。

構文

サーバーモードを起動するには:

```
server <server-number>
```

サーバーモードを終了するには:

```
server> exit
```

例

このコマンドでは、サーバー番号 0 の HSM のサーバーモードを入力します。

```
aws-cloudhsm> server 0

Server is in 'E2' mode...
```

サーバーモードを終了するには、exit コマンドを使用します。

```
server0> exit
```

引数

```
server <server-number>
```

<server-number>

送信先 HSM のサーバー数を指定します。

必須: はい

exit コマンドの引数がありません。

関連トピック

- syncKey
- createUser
- deleteUser

setAttribute

cloudhsm_mgmt_util の setAttribute コマンドは、HSM のキーのラベル、暗号化、復号化、ラップ、アンラップの属性の値を変更します。また、key_mgmt_util の setAttribute コマンドを使用して、セッションキーを永続キーに変換することができます。自分が所有するキーの属性のみ変更できます。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- Crypto User (CU)

Syntax

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
setAttribute <key handle> <attribute id>
```

例

次の例では、対称キーの復号機能を無効にする方法を示します。次のようなコマンドを使用すると、他のキーのラップやラップ解除はできるが、データの暗号化や復号はできないようにラップキーを設定できます。

最初のステップでは、ラップキーを作成します。このコマンドは key_mgmt_util genSymKeyで を使用して 256 ビット AES 対称キーを生成します。出力は、新しいキーのキーハンドルが 14 であることを示しています。

```
$  genSymKey -t 31 -s 32 -l aes256

Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

次に、復号属性の現在の値を確認します。復号属性の属性 ID を取得するには、listAttributes を使用します。出力は、OBJ_ATTR_DECRYPT 属性を表す定数が 261 であることを示しています。キー属性の解釈については、「キー属性リファレンス」を参照してください。

```
aws-cloudhsm> listAttributes

Following are the possible attribute values for getAttribute:

      OBJ_ATTR_CLASS                = 0
      OBJ_ATTR_TOKEN                = 1
      OBJ_ATTR_PRIVATE              = 2
      OBJ_ATTR_LABEL                = 3
      OBJ_ATTR_TRUSTED              = 134
```

```
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ID                     = 258
      OBJ_ATTR_SENSITIVE              = 259
      OBJ_ATTR_ENCRYPT                = 260
      OBJ_ATTR_DECRYPT                = 261
      OBJ_ATTR_WRAP                   = 262
      OBJ_ATTR_UNWRAP                 = 263
      OBJ_ATTR_SIGN                   = 264
      OBJ_ATTR_VERIFY                 = 266
      OBJ_ATTR_DERIVE                 = 268
      OBJ_ATTR_LOCAL                  = 355
      OBJ_ATTR_MODULUS                = 288
      OBJ_ATTR_MODULUS_BITS           = 289
      OBJ_ATTR_PUBLIC_EXPONENT        = 290
      OBJ_ATTR_VALUE_LEN              = 353
      OBJ_ATTR_EXTRACTABLE            = 354
      OBJ_ATTR_NEVER_EXTRACTABLE      = 356
      OBJ_ATTR_ALWAYS_SENSITIVE       = 357
      OBJ_ATTR_DESTROYABLE            = 370
      OBJ_ATTR_KCV                    = 371
      OBJ_ATTR_WRAP_WITH_TRUSTED      = 528
      OBJ_ATTR_WRAP_TEMPLATE          = 1073742353
      OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
      OBJ_ATTR_ALL                    = 512
```

キー 14 の 復号属性の現在の値を取得するために、次のコマンドは cloudhsm_mgmt_util の getAttribute を使用します。

出力は、復号属性の値がクラスター内の両方の HSM において true (1) であることを示しています。

```
aws-cloudhsm> getAttribute 14 261

Attribute Value on server 0(10.0.0.1):
OBJ_ATTR_DECRYPT
0x00000001

Attribute Value on server 1(10.0.0.2):
OBJ_ATTR_DECRYPT
0x00000001
```

次のコマンドでは、setAttribute を使用してキー 14 の復号属性の値 (属性 261) を 0 に変更します。これにより、キーの復号機能が無効になります。

出力は、クラスター内の両方の HSM でコマンドが成功したことを示しています。

```
aws-cloudhsm> setAttribute 14 261 0
************************CAUTION**********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****************************************************************

Do you want to continue(y/n)? y
setAttribute success on server 0(10.0.0.1)
setAttribute success on server 1(10.0.0.2)
```

最後のコマンドでも、getAttribute コマンドを使用します。この場合も、キー 14 の復号属性 (属性 261) が取得されます。

今回の出力は、復号属性の値がクラスター内の両方の HSM において false (0) であることを示しています。

```
aws-cloudhsm>getAttribute 14 261
Attribute Value on server 0(10.0.3.6):
OBJ_ATTR_DECRYPT
0x00000000

Attribute Value on server 1(10.0.1.7):
OBJ_ATTR_DECRYPT
0x00000000
```

引数

```
setAttribute <key handle> <attribute id>
```

<key-handle>

所有するキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、key_mgmt_util の findKey を使用します。キーのユーザーを検索するには、 を使用しますgetKeyInfo。

必須: はい

<attribute id>

変更する属性を表す定数を指定します。各コマンドに指定できる属性は 1 つのみです。属性とその整数値を取得するには、listAttributes を使用します。キー属性の解釈については、「キー属性リファレンス」を参照してください。

有効値:

- 3 – OBJ_ATTR_LABEL。
- 134 – OBJ_ATTR_TRUSTED。
- 260 – OBJ_ATTR_ENCRYPT。
- 261 – OBJ_ATTR_DECRYPT。
- 262 – OBJ_ATTR_WRAP。
- 263 – OBJ_ATTR_UNWRAP。
- 264 – OBJ_ATTR_SIGN。
- 266 – OBJ_ATTR_VERIFY。
- 268 – OBJ_ATTR_DERIVE。
- 370 – OBJ_ATTR_DESTROYABLE。
- 528 – OBJ_ATTR_WRAP_WITH_TRUSTED。
- 1073742353 – OBJ_ATTR_WRAP_TEMPLATE。
- 1073742354 – OBJ_ATTR_UNWRAP_TEMPLATE。

必須: はい

関連トピック

- key_mgmt_util で setAttribute
- getAttribute
- listAttributes
- キー属性リファレンス

quit

cloudhsm_mgmt_util の quit コマンドは cloudhsm_mgmt_util を終了します。タイプに関係なく、すべてのユーザーがこのコマンドを使用できます。

cloudhsm_mgmt_util コマンドを実行する前に、cloudhsm_mgmt_util を起動する必要があります。

ユーザーのタイプ

このコマンドは、次のユーザーが実行できます。

- すべてのユーザー。このコマンドは、ログインしていなくても実行できます。

Syntax

```
quit
```

例

このコマンドは、cloudhsm_mgmt_util を終了します。正常に完了すると、通常のコマンドラインに戻ります。このコマンドには出力パラメータはありません。

```
aws-cloudhsm> quit

disconnecting from servers, please wait...
```

関連トピック

- cloudhsm_mgmt_util の起動方法

shareKey

cloudhsm_mgmt_util の shareKey コマンドは、自分が所有するキーを他の Crypto ユーザーと共有したり、共有解除したりします。キーの共有や共有解除ができるのは、キーの所有者のみです。キーの作成時にキーの共有もできます。

キーを共有するユーザーは、暗号化オペレーションでキーを使用することはできますが、そのキーの削除、エクスポート、共有や共有解除、およびキー属性の変更はできません。キーでクォーラム認証が有効になっている場合、そのクォーラムは、キーの共有や共有解除を行うオペレーションを承認する必要があります。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto User (CU)

Syntax

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

ユーザータイプ: Crypto user (CU)

```
shareKey <key handle> <user id> <(share/unshare key?) 1/0>
```

例

以下の例では、shareKey を使用して、所有しているキーを他の Crypto User と共有および共有解除する方法を示します。

Example : キーを共有する

次の例では、shareKey を使用して、現在のユーザーが所有している ECC プライベートキーを HSM の別の Crypto User と共有します。パブリックキーは HSM のすべてのユーザーが利用できるため、共有も共有解除もできません。

最初のコマンドではgetKeyInfo、 を使用して262177、HSM の ECC プライベートキーであるキー HSMs のユーザー情報を取得します。

出力は、キー 262177 の所有者がユーザー 3 であり、他に共有されていないことを示しています。

```
aws-cloudhsm>getKeyInfo 262177

Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

Key Info on server 1(10.0.3.6):
```

```
        Token/Flash Key,

        Owned by user 3
```

このコマンドでは、shareKey を使用して、キー 262177 を HSM の別の Crypto User であるユーザー 4 と共有します。最後の引数では、値 1 で共有オペレーションを指定します。

出力は、クラスター内の両方の HSM でオペレーションが成功したことを示しています。

```
aws-cloudhsm>shareKey 262177 4 1
**************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************

Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

オペレーションが成功したことを検証するため、この例では最初の getKeyInfo コマンドを再度実行します。

出力は、キー 262177 がユーザー 4 と共有されるようになったことを示しています。

```
aws-cloudhsm>getKeyInfo 262177

Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                4
Key Info on server 1(10.0.3.6):

        Token/Flash Key,

        Owned by user 3
```

```
        also, shared to following 1 user(s):

                4
```

Example : キーを共有解除する

次の例では、対称キーを共有解除します。つまり、キーの共有ユーザーのリストから Crypto User を削除します。

このコマンドでは、shareKey を使用して、キー 6 の共有ユーザーのリストからユーザー 4 を削除します。最後の引数では、値 0 で共有解除オペレーションを指定します。

出力は、両方の HSM でコマンドが成功したことを示しています。その結果、ユーザー 4 は暗号化オペレーションでキー 6 を使用できなくなります。

```
aws-cloudhsm>shareKey 6 4 0
**************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************

Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
shareKey <key handle> <user id> <(share/unshare key?) 1/0>
```

<key-handle>

所有するキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、key_mgmt_util の findKey を使用します。キーを所有していることを確認するには、 を使用しますgetKeyInfo。

必須: はい

<user id>

キーを共有または共有解除する Crypto User (CU) のユーザー ID を指定します。ユーザーのユーザー ID を検索するには、listUsers を使用します。

必須: はい

<share 1 or unshare 0>

指定したユーザーとキーを共有するには「1」と入力します。キーを共有解除するには、つまり、指定したユーザーをキーの共有ユーザーのリストから削除するには「0」と入力します。

必須: はい

関連トピック

- getKeyInfo

syncKey

cloudhsm_mgmt_util の syncKey コマンドを使用して、クラスター内、または、クローンクラスター間の HSM インスタンス間でキーを手動で同期することができます。通常、クラスター内の HSM インスタンスは自動的にキーを同期させるため、このコマンドを使用する必要はありません。ただし、クローン複製されたクラスター間でキーを同期する場合は、手動で実行する必要があります。クローンされたクラスターは通常、グローバルスケーリングとディザスタリカバリのプロセスを簡素化するために、異なる AWS リージョンに作成されます。

syncKey を使用して、任意のクラスター間でキーを同期させることはできません。クラスターの 1 つは、他のクラスターのバックアップから作成する必要があります。さらに、オペレーションが成功するように、両方のクラスターの CO および CU の認証情報が一致している必要があります。詳細については、「HSM ユーザー」を参照してください。

を使用するにはsyncKey、まずソースクラスターから 1 つの HSM を指定し、宛先クラスターから 1 つの HSM を指定するAWS CloudHSM 設定ファイルを作成する必要があります。これにより、cloudhsm_mgmt_util は両方の HSM インスタンスに接続することができます。cloudhsm_mgmt_util を起動するには、この設定を使用します。。次に、同期させるキーを所有する CO または CU の認証情報を使用して、ログインします。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto Officer (CO)
- Crypto User (CU)

> **Note**
>
> CO は、任意のキーの syncKey を使用できますが、CU は、所有するキーでのみこのコマンドを使用できます。詳細については、「the section called “HSM ユーザーを理解する”」を参照してください。

前提条件

開始する前に、送信先の HSM と同期する送信元 HSM のキーの `key handle` を把握しておく必要があります。`key handle` を検索するには、listUsers コマンドを使用して、名前付きユーザーのすべての識別子を表示します。次に、 findAllKeys コマンドを使用して、特定のユーザーに属するすべてのキーを検索します。

また、cloudhsm_mgmt_util が開始時に返すトレース出力に表示される、出典とデスティネーションの HSM に割り当てられた `server IDs` も知っておく必要があります。これらは、設定ファイルに表示されている HSM と同じ順序で表示されます。

「クローンされたクラスター間で CMU を使用する」の手順に従って、新しい設定ファイルで cloudhsm_mgmt_util を初期化します。続いて、server コマンドを発行して、送信元 HSM のサーバーモードを入力します。

構文

> **Note**
>
> syncKey を実行するには、まず HSM のサーバーモードを入力します。これには、同期されるキーが含まれます。

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

ユーザータイプ: Crypto user (CU)

```
syncKey <key handle> <destination hsm>
```

例

server コマンドを実行して、送信元 HSM にログインし、サーバーモードを入力します。この例では、`server 0` が送信元 HSM であると仮定しています。

```
aws-cloudhsm> server 0
```

ここで、syncKey コマンドを実行します。この例では、キー `261251` が、`server 1` に同期されると仮定しています。

```
aws-cloudhsm> syncKey 261251 1
syncKey success
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
syncKey <key handle> <destination hsm>
```

<key handle>

同期させるキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、HSM サーバーにログインfindAllKeysしているときに を使用します。

必須：はい

<destination hsm>

キーを同期しているサーバーの数を指定します。

必須: はい

関連トピック

- listUsers
- findAllKeys
- のクラスターを記述する AWS CLI

- サーバー

syncUser

cloudhsm_mgmt_util の syncUser コマンドを使用して、クラスター内の HSM CUs) または crypto Officer (COs) を手動で同期できます。ユーザーは自動的に同期 AWS CloudHSM されません。通常、クラスター内の HSM がすべてまとめて更新されるように、ユーザーはグローバルモードで管理します。HSM が誤って同期解除された場合 (たとえば、パスワードの変更など)、またはクローンされたクラスター間でユーザーの認証情報を更新する場合は、syncUser を使用する場合があります。クローンされたクラスターは通常、グローバルスケーリングとディザスタリカバリのプロセスを簡素化するために、異なる AWS リージョンに作成されます。

CMU コマンドを実行する前に CMU を起動し、HSM にログインする必要があります。ログインに使用するユーザータイプで、使用するコマンドを実行できることを確認してください。

HSM を追加または削除する場合は、CMU の設定ファイルを更新します。さもないと、クラスター内のすべての HSM で変更が有効にならない場合があります。

ユーザーのタイプ

このコマンドは、次のタイプのユーザーが実行できます。

- Crypto Officer (CO)

前提条件

開始する前に、送信先の HSM と同期する送信元 HSM のユーザーの `user ID` を把握しておく必要があります。`user ID` を確認するには、listUsers コマンドを使用して、クラスター内の HSM のすべてのユーザーを表示します。

また、cloudhsm_mgmt_util が開始時に返すトレース出力に表示される、出典とデスティネーションの HSM に割り当てられた `server ID` も知っておく必要があります。これらは、設定ファイルに表示されている HSM と同じ順序で表示されます。

クローンされたクラスター間で HSM を同期する場合は、「クローンされたクラスター間で CMU を使用する」の説明に従って、cloudhsm_mgmt_util を新しい設定ファイルで初期化します。

syncUser を実行する準備ができたら、server コマンドを発行して、送信元 HSM のサーバーモードを入力します。

構文

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
syncUser <user ID> <server ID>
```

例

server コマンドを実行して、送信元 HSM にログインし、サーバーモードを入力します。この例では、`server 0` が送信元 HSM であると仮定しています。

```
aws-cloudhsm> server 0
```

syncUser コマンドを実行します。この例では、ユーザー 6 が同期対象のユーザー、`server 1` が送信先 HSM であると仮定しています。

```
server 0> syncUser 6 1
ExtractMaskedObject: 0x0 !
InsertMaskedObject: 0x0 !
syncUser success
```

引数

このコマンドには名前付きパラメータがないため、引数は図表で指定された順序で入力する必要があります。

```
syncUser <user ID> <server ID>
```

<user ID>

同期するユーザーの ID を指定します。各コマンドに指定できるユーザーは 1 つのみです。ユーザーの ID を取得するには、listUsers を使用します。

必須：はい

<server ID>

ユーザーを同期している HSM のサーバーの数を指定します。

必須: はい

関連トピック

- listUsers
- のクラスターを記述する AWS CLI
- サーバー

キー管理ユーティリティ (KMU)

キー管理ユーティリティ (KMU) は、暗号ユーザー (CU) がハードウェアセキュリティモジュール (HSM) のキーを管理するのに役立つコマンドラインツールです。KMU には、キーの生成、削除、インポート、エクスポート、属性の取得と設定、キーの検索、暗号化操作の実行を行う複数のコマンドが含まれています。

KMU と CMU は Client SDK 3 スイートの一部です。

クイックスタートについては、「key_mgmt_util の起動方法」を参照してください。コマンドの詳細については、key_mgmt_util コマンドリファレンス を参照してください。キー属性の解釈については、キー属性リファレンス を参照してください。

Linux をしている場合に key_mgmt_util を使用するには、クライアントインスタンスに接続し、「AWS CloudHSM クライアントのインストールと設定 (Linux)」を参照してください。Windows を使用している場合は、「AWS CloudHSM クライアントのインストールと設定 (Windows)」を参照してください。

トピック

- key_mgmt_util の起動方法
- AWS CloudHSM クライアントのインストールと設定 (Linux)
- AWS CloudHSM クライアントのインストールと設定 (Windows)
- key_mgmt_util コマンドリファレンス

key_mgmt_util の起動方法

AWS CloudHSM には、AWS CloudHSM クライアントソフトウェア に 2 つのコマンドラインツールが含まれています。cloudhsm_mgmt_util ツールには、HSM ユーザーを管理するためのコマンドが含まれています。key_mgmt_util ツールには、キーを管理するためのコマンドが含まれています。key_mgmt_util コマンドラインツールの使用を開始するには、次のトピックを参照してください。

トピック

- key_mgmt_util のセットアップ
- key_mgmt_util の基本用法

コマンドでエラーメッセージまたは予期しない結果が発生した場合は、「トラブルシューティング AWS CloudHSM」のトピックを参照してください。key_mgmt_util コマンドの詳細については、key_mgmt_util コマンドリファレンス を参照してください。

key_mgmt_util のセットアップ

key_mgmt_util を使用する前に、次のセットアップを完了します。

AWS CloudHSM クライアントを起動する

key_mgmt_util を使用する前に、 AWS CloudHSM クライアントを起動する必要があります。クライアントは、クラスター内の HSMs との end-to-end 暗号化された通信を確立するデーモンです。key_mgmt_util ツールはクライアント接続を使用して、クラスターの HSM と通信します。クライアント接続がない場合、key_mgmt_util は動作しません。

AWS CloudHSM クライアントを起動するには

AWS CloudHSM クライアントを起動するには、次のコマンドを使用します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

key_mgmt_util の起動

AWS CloudHSM クライアントを起動したら、次のコマンドを使用して key_mgmt_util を起動します。

Amazon Linux

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Amazon Linux 2

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 16.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 18.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Windows

```
c:\Program Files\Amazon\CloudHSM> .\key_mgmt_util.exe
```

key_mgmt_util が実行されていると、プロンプトは `Command:` に変わります。

`Daemon socket connection error` メッセージが返されるなど、コマンドが失敗した場合は、設定ファイルを更新してみます。

key_mgmt_util の基本用法

key_mgmt_util ツールの基本用法については、次のトピックを参照してください。

トピック

- HSM へのログイン

- HSM からのログアウト
- key_mgmt_util の停止

HSM へのログイン

HSM にログインするには、loginHSM コマンドを使用します。次のコマンドは、example_user という名前の 暗号ユーザー (CU) としてログインします。出力は、クラスターの 3 つすべての HSM のログインが成功したことを示します。

```
Command:  loginHSM -u CU -s example_user -p <PASSWORD>
Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

loginHSM コマンドの構文を次に示します。

```
Command:  loginHSM -u <USER TYPE> -s <USERNAME> -p <PASSWORD>
```

HSM からのログアウト

HSM からログアウトするには、logoutHSM コマンドを使用します。

```
Command:  logoutHSM
Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

key_mgmt_util の停止

key_mgmt_util を停止するには、exit コマンドを使用します。

```
Command:  exit
```

AWS CloudHSM クライアントのインストールと設定 (Linux)

AWS CloudHSM クラスター内の HSM を操作するには、Linux 用の AWS CloudHSM クライアントソフトウェアが必要です。このクライアントを以前に作成した Linux EC2 クライアントインスタンスにインストールする必要があります。Windows を使用している場合は、クライアントをインストールすることもできます。詳細については、「AWS CloudHSM クライアントのインストールと設定 (Windows)」を参照してください。

タスク

- AWS CloudHSM クライアントツールとコマンドラインツールをインストールする
- クライアント設定の編集

AWS CloudHSM クライアントツールとコマンドラインツールをインストールする

クライアントインスタンスに接続し、次のコマンドを実行して、 AWS CloudHSM クライアントおよびコマンドラインツールをダウンロードしてインストールします。

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb
```

クライアント設定の編集

AWS CloudHSM クライアントを使用してクラスターに接続する前に、クライアント設定を編集する必要があります。

クライアント設定を編集するには

1. 発行証明書 (クラスターの証明書に署名するために使用したもの) を、クライアントインスタンスの次の場所にコピーします：/opt/cloudhsm/etc/customerCA.crt。この場所に証明書をコピーするには、クライアントインスタンスにルートユーザーアクセス権限が必要です。
2. 次の configure コマンドを使用して、 AWS CloudHSM クライアントツールとコマンドラインツールの設定ファイルを更新し、クラスター内の HSM の IP アドレスを指定します。HSM の IP アドレスを取得するには、AWS CloudHSM コンソール でクラスターを表示するか、 describe-clusters AWS CLI コマンドを実行します。コマンドの出力では、HSM の IP アドレスは EniIp フィールドの値です。複数の HSM がある場合は、いずれかの HSM の IP アドレスを選択してください。どれでも構いません。

   ```
   sudo /opt/cloudhsm/bin/configure -a <IP address>

   Updating server config in /opt/cloudhsm/etc/cloudhsm_client.cfg
   Updating server config in /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

3. クラスターのアクティブ化 に移動します。

AWS CloudHSM クライアントのインストールと設定 (Windows)

Windows で AWS CloudHSM クラスター内の HSM を使用するには、Windows 用の AWS CloudHSM クライアントソフトウェアが必要です。このクライアントを以前に作成した Windows Server インスタンスにインストールする必要があります。

最新 Windows クライアントとコマンドラインツールをインストール (または更新) します。

1. Windows Server インスタンスに接続します。
2. ダウンロードページ から最新の (AWSCloudHSMClient-latest.msi) をダウンロードします。
3. ダウンロード場所に移動し、管理者権限を持つインストーラ (AWSCloudHSMClient-latest.msi) を実行します。
4. インストーラの手順に従い、インストーラが終了したら 閉じる を選択します。
5. クラスターの証明書に署名するために使用した 自己署名発行証明書を `C:\ProgramData\Amazon\CloudHSM` フォルダにコピーします。
6. 以下のコマンドを実行して、設定ファイルを更新します。更新中は、再設定の間に必ずクライアントを停止してから再開します。

```
C:\Program Files\Amazon\CloudHSM\bin\ .\configure.exe -a <HSM IP address>
```

7. クラスターのアクティブ化 に移動します。

注意:

- クライアントを更新する場合、以前のインストールに存在する設定ファイルは上書き されません。
- Windows 用の AWS CloudHSM クライアントインストーラは、Cryptography API: Next Generation (CNG) とキーストレージプロバイダー (KSP) を自動的に登録します。クライアントをアンインストールするには、インストーラーを再度実行し、アンインストール手順に従います。
- Linux を使用している場合は、Linux クライアントをインストールすることもできます。詳細については、「AWS CloudHSM クライアントのインストールと設定 (Linux)」を参照してください。

key_mgmt_util コマンドリファレンス

key_mgmt_util コマンドラインツールでは、クラスターの HSM のキーを管理できます。たとえば、キーとその属性の作成、削除、検索などができます。このトピックでは、このツールに含まれている各コマンドについて詳しく説明します。

クイックスタートについては、「key_mgmt_util の起動方法」を参照してください。キー属性の解釈については、キー属性リファレンス を参照してください。クラスターの HSM とユーザーを管理するコマンドを含む cloudhsm_mgmt_util コマンドラインツールについては、CloudHSM 管理ユーティリティ (CMU) を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、暗号ユーザー (CU) として HSM に ログインする 必要があります。

すべての key_mgmt_util コマンドを一覧表示するには、次のように入力します。

```
Command: help
```

特定の key_mgmt_util コマンドのヘルプを表示するには、次のように入力します。

```
Command: <command-name> -h
```

key_mgmt_util セッションを終了するには、次のように入力します。

```
Command: exit
```

次のトピックでは、key_mgmt_util のコマンドについて説明します。

> **Note**
>
> key_mgmt_util と cloudhsm_mgmt_util のコマンドには、同じ名前のものがあります。ただし、コマンドは通常、構文が異なり、出力が異なり、機能がわずかに異なります。

コマンド	説明
aesWrapUnwrap	ファイルのキーの内容を暗号化および復号します。
deleteKey	HSM からキーを削除します。
Error2String	key_mgmt_util 16 進エラーコードに対応するエラーを取得します。
exit	key_mgmt_util を終了します。
exportPrivateKey	プライベートキーのコピーを HSM からディスク上のファイルにエクスポートします。

コマンド	説明
exportPubKey	パブリックキーのコピーを HSM からファイルにエクスポートします。
exSymKey	対称キーのプレーンテキストコピーを HSM からファイルにエクスポートします。
extractMaskedObject	HSM のキーをマスクされたオブジェクトとして抽出します。
findKey	キーの属性値に基づいてキーを検索します。
findSingleKey	クラスターのすべての HSM にキーが存在することを検証します。
genDSAKeyPair	デジタル署名アルゴリズム (DSA) キーペアを HSM に生成します。
genECCKeyPair	楕円曲線暗号 (ECC) キーペアを HSM に生成します。
genRSAKeyPair	RSA 非対称キーペアを HSM に生成します。
genSymKey	対称キーを HSM に生成します。
getAttribute	AWS CloudHSM キーの属性値を取得し、ファイルに書き込みます。
getCaviumPrivキー	プライベートキーのフェイク PEM 形式バージョンを作成し、ファイルにエクスポートします。
getCert	HSM のパーティション証明書を取得し、それをファイルに保存します。

コマンド	説明
getKeyInfo	キーを使用できるユーザーの HSM ユーザー ID を取得します。 キーがクォーラム制御されている場合は、クォーラムのユーザー数を取得します。
help (ヘルプ)	key_mgmt_util で使用可能なコマンドに関するヘルプ情報を表示します。
importPrivateKey	プライベートキーを HSM にインポートします。
importPubKey	パブリックキーを HSM にインポートします。
imSymKey	対称キーのプレーンテキストコピーをファイルから HSM 内にインポートします。
insertMaskedObject	ディスク上のファイルにあるマスクされたオブジェクトを、そのオブジェクトの元のクラスターの関連クラスターに格納されている HSM に挿入します。関連クラスターとは、元のクラスターのバックアップから生成されたクラスターを指します。
???	あるファイルに、実際のプライベートキーとフェイク PEM キーのどちらが含まれているかを特定します。
listAttributes	AWS CloudHSM キーの属性とそれを表す定数を一覧表示します。
listUsers	HSM のユーザー、そのユーザータイプと ID、およびその他の属性を取得します。
loginHSM および logoutHSM	クラスターの HSM でログインおよびログアウトを行います。

コマンド	説明
setAttribute	セッションキーを永続キーに変換します。
sign	選択されたプライベートキーを使ってファイルのための署名を生成します。
unWrapKey	ラップ (暗号化) されたキーをファイルから HSM 内にインポートします。
verify	特定のファイルへの署名に特定のキーが使用されたかどうかを検証します。
wrapKey	キーの暗号化されたコピーを HSM からファイルにエクスポートします。

aesWrapUnwrap

aesWrapUnwrap コマンドは、ディスク上のファイルのコンテンツを暗号化または復号します。このコマンドは、暗号化キーをラップおよびラップ解除するように設計されていますが、4 KB (4096 バイト) 未満のデータを含むすべてのファイルに使用できます。

aesWrapUnwrap は AES キーのラップを使用します。HSM で、ラップまたはラップ解除キーとして、AES キーを使用します。その後、ディスク上の別のファイルに結果が書き込まれます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
aesWrapUnwrap -h

aesWrapUnwrap -m <wrap-unwrap mode>
              -f <file-to-wrap-unwrap>
              -w <wrapping-key-handle>
              [-i <wrapping-IV>]
              [-out <output-file>]
```

例

以下の例では、aesWrapUnwrap を使用してファイルの暗号化キーを暗号化および復号する方法を示します。

Example : 暗号化キーをラップする

次のコマンドでは、aesWrapUnwrap を使用してプレーンテキストで HSM からエクスポートされた Triple DES 対称キーを 3DES.key ファイルにラップします。同様のコマンドを使用して、ファイルに保存されたキーをラップできます。

コマンドは、ラップモードを示す 1 の値で -m パラメータを使用します。-w パラメータを使用して HSM の AES キー (キーハンドル 6) をラップキーとして指定します。ラップしたキーは 3DES.key.wrapped ファイルに書き込まれます。

出力は、コマンドが正常に実行され、推奨されているデフォルトの IV をオペレーションが使用したことを示しています。

```
 Command:  aesWrapUnwrap -f 3DES.key -w 6 -m 1 -out 3DES.key.wrapped

        Warning: IV (-i) is missing.
                 0xA6A6A6A6A6A6A6A6 is considered as default IV
result data:
49 49 E2 D0 11 C1 97 22
17 43 BD E3 4E F4 12 75
8D C1 34 CF 26 10 3A 8D
6D 0A 7B D5 D3 E8 4D C2
79 09 08 61 94 68 51 B7

result written to file 3DES.key.wrapped

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
```

Example : 暗号化キーをラップ解除する

次の例では、aesWrapUnwrap を使用してラップ (暗号化) したキーをラップ解除 (復号) する方法を示します。HSM にキーをインポートする前に次のようなオペレーションを実行できます。例えば、imSymKey コマンドを使用して暗号化されたキーをインポートしようとすると、暗号化されたキーにそのタイプのプレーンテキストキーに必要な形式がないため、エラーが返されます。

コマンドは 3DES.key.wrapped ファイルのキーをラップ解除し、プレーンテキストを 3DES.key.unwrapped ファイルに書き込みます。コマンドは、ラップ解除モードを示す -m の値で

0 パラメータを使用します。-w パラメータを使用して HSM の AES キー (キーハンドル 6) をラップキーとして指定します。ラップしたキーは 3DES.key.unwrapped ファイルに書き込まれます。

```
 Command:  aesWrapUnwrap -m 0 -f 3DES.key.wrapped -w 6 -out 3DES.key.unwrapped

        Warning: IV (-i) is missing.
                 0xA6A6A6A6A6A6A6A6 is considered as default IV
result data:
14 90 D7 AD D6 E4 F5 FA
A1 95 6F 24 89 79 F3 EE
37 21 E6 54 1F 3B 8D 62

result written to file 3DES.key.unwrapped

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-m

モードを指定します。ファイルの内容をラップ (暗号化) するには「1」と入力します。ファイルの内容をラップ解除 (復号) するには「0」と入力します。

必須: はい

-f

ラップするファイルを指定します。4 KB (4096 バイト) 未満のデータを含むファイルを入力します。このオペレーションは、暗号化キーをラップおよびラップ解除するように設計されています。

必須: はい

-w

ラップキーを指定します。HSM で AES キーのキーハンドルを入力します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

ラッピングキーを作成するには、 genSymKeyを使用して AES キー (タイプ 31) を生成します。

必須: はい

-i

アルゴリズムの代替の初期値 (IV) を指定します。代替を必要とする特殊な条件がなければ、デフォルト値を使用します。

デフォルト: 0xA6A6A6A6A6A6A6A6。デフォルト値は AES キーのラップのアルゴリズム仕様で定義されています。

必須: いいえ

-out

ラップまたはラップ解除されたキーを含む出力ファイルに代わりの名前を指定します。デフォルトは、ローカルディレクトリの wrapped_key (ラップオペレーション) および unwrapped_key (ラップ解除オペレーション) です。

既存のファイルがある場合、aesWrapUnwrap は警告なしに上書きされます。コマンドが失敗すると、aesWrapUnwrap で内容のない出力ファイルが作成されます。

デフォルト: ラップ: wrapped_key。ラップ解除: unwrapped_key。

必須: いいえ

関連トピック

- exSymKey
- imSymKey
- unWrapKey
- wrapKey

deleteKey

key_mgmt_util の deleteKey コマンドは、HSMからキーを削除します。一度に削除できるキーは 1 つだけです。キーペアの一方のキーを削除しても、ペアの他方のキーには影響がありません。

キーを削除できるのは、キー所有者のみです。キーを共有するユーザーは、キーを暗号化オペレーションで使用することはできますが、削除することはできません。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
deleteKey -h

deleteKey -k
```

例

以下の例では、deleteKey を使用してキーを HSM から削除する方法を示します。

Example : キーを削除する

次のコマンドでは、キーハンドルが 6 のキーを削除します。コマンドが成功すると、deleteKey はクラスターの各 HSM から成功メッセージを返します。

```
Command: deleteKey -k 6

        Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : キーを削除する (失敗)

指定したキーハンドルに対応するキーがないためにコマンドが失敗すると、deleteKey はオブジェクトハンドルが無効であるというエラーメッセージを返します。

```
Command: deleteKey -k 252126

        Cfm3FindKey returned: 0xa8 : HSM Error: Invalid object handle is passed to this
 operation

        Cluster Error Status
        Node id 1 and err state 0x000000a8 : HSM Error: Invalid object handle is passed
 to this operation
        Node id 2 and err state 0x000000a8 : HSM Error: Invalid object handle is passed
 to this operation
```

現在のユーザーがキーの所有者ではないためにコマンドが失敗すると、コマンドはアクセス拒否エラーを返します。

```
Command:  deleteKey -k 262152

Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied.
```

パラメータ

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-k

削除するキーのキーハンドルを指定します。HSM のキーのキーハンドルを確認するには、findKey を使用します。

必須: はい

関連トピック

- findKey

Error2String

key_mgmt_util の Error2String helper コマンドは、key_mgmt_util の 16 進エラーコードに対応するエラーを返します。このコマンドは、コマンドとスクリプトのトラブルシューティングに使用できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
Error2String -h

Error2String -r <response-code>
```

例

これらの例は、key_mgmt_util エラーコードのエラー文字列を取得するために Error2String を使用する方法を示しています。

Example : エラーの説明を取得する

次のコマンドでは、0xdb エラーコードに関するエラーの説明を取得します。説明では、ユーザーがユーザータイプを間違えたため、key_mgmt_util へのログインに失敗したとなります。key_mgmt_util にログインできるのは暗号ユーザー (CU) のみです。

```
        Command:  Error2String -r 0xdb

        Error Code db maps to HSM Error: Invalid User Type.
```

Example : エラーコードを見つける

この例は、key_mgmt_util エラーのエラーコードの場所を示しています。エラーコード 0xc6 は、文字列 Cfm3*command-name* returned: の後に表示されます。

この例では、 は現在のユーザー (ユーザー 4) が暗号化オペレーションで キーを使用できるgetKeyInfoことを示します。その場合でも、ユーザーが deleteKey を使用してキーを削除しようとすると、コマンドはエラーコード 0xc6 を返します。

```
        Command:  deleteKey -k 262162

        Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied

        Cluster Error Status

        Command:  getKeyInfo -k 262162

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3

       also, shared to following 1 user(s):

                4
```

0xc6 エラーが返された場合は、次のような Error2String コマンドを使用してエラーを検索できます。この例で deleteKey コマンドが失敗しているのは、現在のユーザーがキーを共有していても、キーの所有者が別のユーザーであるために、アクセス拒否エラーとなるためです。キーを削除できるのはキーの所有者のみです。

```
        Command:  Error2String -r 0xa8

        Error Code c6 maps to HSM Error: Key Access is denied
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-r

16 進数のエラーコードを指定します。16 進数インジケータ 0x は必須です。

必須: はい

exit

key_mgmt_util の exit コマンドは、key_mgmt_util を終了します。正常に終了すると、標準のコマンドラインに戻ります。

どの key_mgmt_util コマンドを実行する場合でも、事前に key_mgmt_util を起動する必要があります。

構文

```
exit
```

パラメータ

このコマンドにはパラメータがありません。

関連トピック

- key_mgmt_util の起動

exportPrivateKey

key_mgmt_util の exportPrivateKey コマンドは、非対称プライベートキーを HSM からファイルにエクスポートします。HSM では、クリアテキストのキーを直接エクスポートすることはできません。このコマンドは、指定した AES ラップキーを使用してプライベートキーをラップし、ラップされたバイトを復号化して、クリアテキストのプライベートキーをファイルにコピーします。

exportPrivateKey コマンドはキーを HSM から削除したり、キー属性を変更したり、今後の暗号化操作でのキーの使用を禁止したりすることはありません。同じキーを複数回エクスポートできます。

OBJ_ATTR_EXTRACTABLE 属性値が 1 のプライベートキーのみエクスポートすることができます。OBJ_ATTR_WRAP と OBJ_ATTR_DECRYPT 属性値 1 を持つ AES ラップキーを指定する必要があります。キーの属性を確認するには、getAttribute コマンドを使用します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
exportPrivateKey -h

exportPrivateKey -k <private-key-handle
                 -w <wrapping-key-handle>
                 -out <key-file>
                 [-m <wrapping-mechanism>]
                 [-wk <wrapping-key-file>]
```

例

この例では、exportPrivateKey を使って HSM からプライベートキーをエクスポートする方法を示します。

Example : プライベートキーをエクスポートする

このコマンドは、ハンドルが 16 のラップキーを使い、ハンドルが 15 のプライベートキーを exportKey.pem という PEM ファイルにエクスポートします。exportPrivateKey は、コマンドが成功すると成功メッセージを返します。

```
Command: exportPrivateKey -k 15 -w 16 -out exportKey.pem
```

```
Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS

PEM formatted private key is written to exportKey.pem
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-k

エクスポートするプライベートキーのキーハンドルを指定します。

必須: はい

-w

ラップキーのキーハンドルを指定します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

キーをラップキーとして使用できるかどうかを確認するには、getAttribute を使用して OBJ_ATTR_WRAP 属性 (262) の値を取得します。ラップキーを作成するには、genSymKey を使用して AES キー (タイプ 31) を作成します。

-wk パラメータを使用して外部のラップ解除キーを指定した場合、エクスポート時の (ラップ解除ではなく) ラップに -w ラップキーが使われます。

必須: はい

-out

エクスポートしたプライベートキーの書き込み先とするファイルの名前を指定します。

必須: はい

-m

エクスポートするプライベートキーのラップ方法を指定します。唯一の有効な値は 4 です。これは NIST_AES_WRAP mechanism. を指します。

デフォルト: 4 (NIST_AES_WRAP)

必須: いいえ

-wk

エクスポートするキーをラップ解除するためのキーを指定します。プレーンテキストの AES キーが含まれているファイルのパスと名前を入力します。

このパラメータを含めた場合、exportPrivateKey は、エクスポートするキーをラップする際に -w ファイルのキーを使用し、ラップ解除する際に -wk パラメータで指定されたキーを使用します。

デフォルト: -w パラメータで指定されたラップキーを使用して、ラップとラップ解除の両方を行う。

必須: いいえ

関連トピック

- importPrivateKey
- wrapKey
- unWrapKey
- genSymKey

exportPubKey

key_mgmt_util の exportPubKey コマンドは、HSM の公開キーをファイルにエクスポートします。これを使用すると、HSM で生成したパブリックキーをエクスポートすることができます。また、このコマンドを使うと、importPubKey コマンドでインポートされたパブリックキーなど、HSM にインポートされたパブリックキーをエクスポートすることもできます。

exportPubKey オペレーションは、キーマテリアルを指定のファイルにコピーします。しかし、キーを HSM から削除したり、キー属性を変更したり、今後の暗号化操作でのキーの使用を禁止したりすることはありません。同じキーを複数回エクスポートできます。

エクスポート可能なパブリックキーは、OBJ_ATTR_EXTRACTABLE 値が 1 であるものに限られます。キーの属性を確認するには、getAttribute コマンドを使用します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする必要があります。

構文

```
exportPubKey -h

exportPubKey -k <public-key-handle
             -out <key-file>
```

例

この例では、exportPubKey を使って HSM からパブリックキーをエクスポートする方法を示します。

Example : パブリックキーをエクスポートする

このコマンドは、ハンドルが 10 のパブリックキーを public.pem というファイルにエクスポートします。exportPubKey は、コマンドが成功すると成功メッセージを返します。

```
Command: exportPubKey -k 10 -out public.pem

PEM formatted public key is written to public.pem

Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-k

エクスポートするパブリックキーのキーハンドルを指定します。

必須: はい

-out

エクスポートしたパブリックキーの書き込み先とするファイルの名前を指定します。

必須: はい

関連トピック

- importPubKey
- キーの生成

exSymKey

key_mgmt_util ツールの exSymKey コマンドは、対称キーのプレーンテキストコピーを HSM からエクスポートし、ディスク上のファイルに保存します。キーの暗号化 (ラップ) されたコピーをエクスポートするには、wrapKey キーを使用します。exSymKey エクスポートするキーのようにプレーンテキストキーをインポートするには、 を使用しますimSymKey。

エクスポートプロセス中に、exSymKey は、指定した AES キー (ラッピングキー) を使用して、エクスポートするキーを ラップ (暗号化) してから アンラップ (復号化)します。ただし、エクスポートオペレーションの結果は、ディスク上のプレーンテキスト (ラップ解除された) キーとなります。

キーの所有者 (キーを作成した CU ユーザー) のみがキーをエクスポートできます。キーを共有するユーザーは、キーを暗号化オペレーションで使用することはできますが、エクスポートすることはできません。

exSymKey オペレーションは、キーマテリアルをユーザーが指定したファイルにコピーしますが、キーを HSM から削除したり、そのキー属性を変更したり、暗号化オペレーションでのキーの使用を禁止したりはしません。同じキーを複数回エクスポートできます。

exSymKey は対称キーのみをエクスポートします。パブリックキーをエクスポートするには、exportPubKey を使用します。プライベートキーをエクスポートするには、exportPrivateKey を使用します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
exSymKey -h

exSymKey -k <key-to-export>
         -w <wrapping-key>
```

```
        -out <key-file>
        [-m 4]
        [-wk <unwrapping-key-file> ]
```

例

以下の例では、exSymKey を使用してユーザーが所有する対称キーを HSM からエクスポートする方法を示しています。

Example : 3 DES 対称キーをエクスポートする

次のコマンドでは、Triple DES (3DES) 対称キー (キーハンドル 7) をエクスポートします。HSM の既存の AES キー (キーハンドル 6) をラップキーとして使用します。次に、3DES キーのプレーンテキストを 3DES.key ファイルに書き込みます。

出力は、キー 7 (3DES キー) が正常にラップ/ラップ解除されて 3DES.key ファイルに書き込まれたことを示しています。

> **Warning**
>
> 出力では「ラップされた対称キー」が出力ファイルに書き込まれたことになっていますが、出力ファイルに含まれているのはプレーンテキスト (ラップ解除された) キーです。

```
        Command: exSymKey -k 7 -w 6 -out 3DES.key

      Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

       Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "3DES.key"
```

Example : セッション専用のラップキーでエクスポートする

次の例では、セッションでのみ有効なキーをラップキーとして使用する方法を示します。エクスポートするキーはラップされた後で、すぐにラップ解除されて、プレーンテキストとして配信されるため、ラップキーを保持する必要はありません。

以下のコマンドでは、キーハンドル 8 の AES キーを HSM からエクスポートします。このために専用の AES セッションキーを作成して使用します。

最初のコマンドではgenSymKey、 を使用して 256 ビット AES キーを作成します。-sess パラメータを使用して、現在のセッションでのみ有効なキーを作成します。

出力は、HSM でキー 262168 が作成されたことを示しています。

```
        Command:  genSymKey -t 31 -s 32 -l AES-wrapping-key -sess

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262168

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

次の例では、キー 8 (エクスポートするキー) が抽出可能な対称キーであることを検証します。また、ラップキー (キー 262168) がセッションでのみ有効な AES キーであることも検証します。findKey コマンドを使用することもできますが、この例では両方のキーの属性をファイルにエクスポートし、grep を使用してファイルの関連する属性値を見つけます。

以下のコマンドでは、getAttribute で -a 値として 512 (すべて) を使用し、キー 8 とキー 262168 のすべての属性を取得します。キー属性の詳細については、「the section called “キー属性リファレンス”」を参照してください。

```
getAttribute -o 8 -a 512 -out attributes/attr_8
getAttribute -o 262168 -a 512 -out attributes/attr_262168
```

以下のコマンドでは、grep を使用してエクスポートするキー (キー 8) の属性と、セッション専用のラップキー (キー 262168) を検証します。

```
    // Verify that the key to be exported is a symmetric key.
    $  grep -A 1 "OBJ_ATTR_CLASS" attributes/attr_8
    OBJ_ATTR_CLASS
    0x04

    // Verify that the key to be exported is extractable.
    $  grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_8
    OBJ_ATTR_EXTRACTABLE
    0x00000001

    // Verify that the wrapping key is an AES key
    $  grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_262168
```

```
    OBJ_ATTR_KEY_TYPE
    0x1f

    // Verify that the wrapping key is a session key
    $  grep -A 1 "OBJ_ATTR_TOKEN" attributes/attr_262168
    OBJ_ATTR_TOKEN
    0x00

    // Verify that the wrapping key can be used for wrapping
     $  grep -A 1 "OBJ_ATTR_WRAP" attributes/attr_262168
    OBJ_ATTR_WRAP
    0x00000001
```

最後に、exSymKey コマンドを使用してキー 8 をエクスポートします。ラップキーとしてセッションキー (キー 262168) を使用します。

セッションが終了すると、キー 262168 は消滅します。

```
        Command:  exSymKey -k 8 -w 262168 -out aes256_H8.key

        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "aes256_H8.key"
```

Example : 外部のラップ解除キーを使用する

次の例では、外部のラップ解除キーを使用して HSM からキーをエクスポートする方法を示します。

HSM からキーをエクスポートする場合、HSM の AES キーをラップキーとして指定します。デフォルトでは、そのラップキーを使用して、エクスポートするキーがラップおよびラップ解除されます。ただし、-wk パラメータを使用すると、exSymKey でディスク上のファイルにある外部キーを使用してラップ解除できます。この場合は、-w パラメータで指定したキーでターゲットキーをラップし、-wk パラメータで指定したファイルのキーでラップ解除します。

ラップキーは AES (対称) キーである必要があるため、HSM のラップキーとディスク上のラップ解除キーは、キーマテリアルが同じであることが必要です。そのためには、エクスポートオペレーションに先立って、HSM に対するラップキーのインポートまたはエクスポートを行う必要があります。

次の例では、HSM の外部でキーを作成して HSM 内にインポートします。エクスポートする対称キーはキーの内部コピーでラップし、ファイルのキーのコピーでラップ解除します。

最初のコマンドでは、OpenSSL を使用して 256 ビット AES キーを生成します。生成したキーは、aes256-forImport.key ファイルに保存されます。OpenSSL コマンドから返される出力はありませんが、いくつかのコマンドを使用して成功したかどうかを確認できます。次の例では、wc (ワードカウント) ツールを使用してファイルに 32 バイトのデータが保存されていることを確認しています。

```
$  openssl rand -out keys/aes256-forImport.key 32

$ wc keys/aes256-forImport.key
 0  2 32 keys/aes256-forImport.key
```

このコマンドは imSymKey コマンドを使用して、aes256-forImport.keyファイルから HSM に AES キーをインポートします。コマンドが完了すると、キーはキーハンドル 262167 で HSM の aes256-forImport.key ファイルに格納されます。

```
Command:  imSymKey -f keys/aes256-forImport.key -t 31 -l aes256-imported -w 6

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 262167

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、エクスポートオペレーションでキーを使用します。このコマンドでは、exSymKey を使用してキー 21 (192 ビット AES キー) をエクスポートします。キーをラップするために、HSM 内にコピーとしてインポートしたキー 262167 を使用します。キーをラップ解除するには、aes256-forImport.key の同じキーマテリアルを使用します。コマンドが完了すると、キー 21 は aes192_h21.key ファイルにエクスポートされます。

```
        Command:  exSymKey -k 21 -w 262167 -out aes192_H21.key -wk aes256-forImport.key
```

```
        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

Wrapped Symmetric Key written to file "aes192_H21.key"
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-k

エクスポートするキーのキーハンドルを指定します。このパラメータは必須です。所有する対称キーのキーハンドルを入力します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

キーがエクスポート可能であることを検証するには、getAttribute コマンドを使用して、OBJ_ATTR_EXTRACTABLE 属性の値を取得します。この属性は定数 354 で表されます。また、ユーザーが所有するキーのみをエクスポートすることもできます。キーの所有者を検索するには、 getKeyInfo コマンドを使用します。

必須: はい

-w

ラップキーのキーハンドルを指定します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

ラップキーは、エクスポートするキーの暗号化 (ラップ) と復号 (ラップ解除) に使用する HSM のキーです。ラップキーとして使用できるのは AES キーのみです。

任意の AES キー (任意のサイズ) をラップキーとして使用できます。ラップキーは、ターゲットキーをラップし、直後にラップ解除するため、セッション専用の AES キーをラップキーとして使用できます。キーをラップキーとして使用できるかどうかを確認するには、getAttribute を使用して、OBJ_ATTR_WRAP 属性の値を取得します。この属性は定数 262 で表されます。ラッピングキーを作成するには、 genSymKeyを使用して AES キー (タイプ 31) を作成します。

-wk パラメータを使用して外部のラップ解除キーを指定すると、エクスポート時に -w ラップキーがラップに使用されます。ただし、ラップ解除には使用されません。

> **Note**
>
> キー 4 は、サポートされていない内部キーを表します。AES キーをラップキーとして作成および管理することをお勧めします。

必須: はい

-out

出力ファイルのパスと名前を指定します。コマンドが成功すると、このファイルに、エクスポートされたキーがプレーンテキストとして配置されます。既存のファイルがある場合は、警告なしに上書きされます。

必須: はい

-m

ラップ方法を指定します。唯一の有効な値は 4 です。これは NIST_AES_WRAP メカニズムを表します。

必須: いいえ

デフォルト: 4

-wk

指定したファイルの AES キーを使用して、エクスポートするキーをラップ解除します。プレーンテキストの AES キーが含まれているファイルのパスと名前を入力します。

このパラメータを含める場合、exSymKey は、-w パラメータで指定した HSM のキーを使用してエクスポートするキーをラップし、-wk ファイルのキーを使用してラップ解除します。-w パラメータと -wk パラメータの値は同じプレーンテキストのキーに解決される必要があります。

必須: いいえ

デフォルト: HSM のラップキーを使用してラップ解除します。

関連トピック

- genSymKey
- imSymKey

- wrapKey

extractMaskedObject

key_mgmt_util の extractMaskedObject コマンドは、HSM からキーを抽出し、隠されたオブジェクトとしてファイルに保存します。マスクされたオブジェクトとは、クローンされたオブジェクトで、insertMaskedObject コマンドを使用して再び元のクラスターに挿入して初めて使用可能になります。マスクされたオブジェクトは、生成元であるクラスター、またはそのクラスターのクローンにしか挿入できません。これには、リージョン間でのバックアップのコピーによって生成されたクラスターのクローンバージョンや、そのバックアップを使って新しいクラスターを作成することで生成されたクラスターのクローンバージョンが含まれます。

マスクされたオブジェクトは、抽出不可能なキー (OBJ_ATTR_EXTRACTABLE 値が 0 であるキー) を含め、キーを効率的にオフロードおよび同期する手段です。これにより、 AWS CloudHSM 設定ファイル を更新しなくても、異なるリージョンの関連クラスター間でキーを安全に同期できます。

> **Important**
>
> マスクされたオブジェクトは、挿入時に復号され、元のキーのキーハンドルとは異なるキーハンドルを与えられます。マスクされたオブジェクトには、属性、所有権、共有情報、クォーラム設定など、元のキーに関連付けられているすべてのメタデータが含まれます。アプリケーションのクラスター間でキーを同期する必要がある場合は、代わりに cloudhsm_mgmt_util で syncKey を使用してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、 HSM に ログインする 必要があります。extractMaskedObject コマンドは、キーを所有する CU または任意の CO が使用できます。

Syntax

```
extractMaskedObject -h

extractMaskedObject -o <object-handle>
                    -out <object-file>
```

例

この例は、extractMaskedObject を使い、HSM のキーをマスクされたオブジェクトとして抽出する方法を示します。

Example : マスクされたオブジェクトを抽出する

このコマンドは、HSM のハンドルが 524295 であるキーからマスクされたオブジェクトを抽出し、maskedObj というファイルとして保存します。extractMaskedObject は、コマンドが成功すると成功メッセージを返します。

```
Command: extractMaskedObject -o 524295 -out maskedObj

Object was masked and written to file "maskedObj"

        Cfm3ExtractMaskedObject returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-o

マスクされたオブジェクトとして抽出するキーのハンドルを指定します。

必須: はい

-out

マスクされたオブジェクトの保存先とするファイルの名前を指定します。

必須: はい

関連トピック

- insertMaskedObject
- syncKey
- リージョン間のバックアップのコピー
- 以前のバックアップからの AWS CloudHSM クラスターの作成

findKey

key_mgmt_util の findKey コマンドを使用して、キー属性の値でキーを検索します。設定したすべての基準にキーが一致すると、findKey はキーハンドルを返します。パラメータがない場合、findKey は HSM で使用できるすべてのキーのキーハンドルを返します。特定のキーの属性値を検索するには、getAttribute を使用します。

すべての key_mgmt_util コマンドと同様に、findKey はユーザー固有です。暗号化オペレーションで現在のユーザーが使用できるキーのみが返されます。これには、現在のユーザーが所有しているキーおよび現在のユーザーと共有されているキーが含まれます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
findKey -h

findKey [-c <key class>]
        [-t <key type>]
        [-l <key label>]
        [-id <key ID>]
        [-sess (0 | 1)]
        [-u <user-ids>]
        [-m <modulus>]
        [-kcv <key_check_value>]
```

例

この例では、findKey を使用して HSM でキーを検索および特定する方法を示します。

Example : すべてのキーを検索する

このコマンドは、HSM の現在のユーザーのすべてのキーを検索します。出力には、そのユーザーが所有および共有しているキーと、HSM のすべてのパブリックキーが含まれます。

特定のキーハンドルを持つキーの属性を取得するには、getAttribute を使用します。現在のユーザーが特定のキーを所有または共有しているかどうかを確認するには、cloudhsm_mgmt_util findAllKeysで getKeyInfoまたは を使用します。

```
Command: findKey
```

```
Total number of keys present 13

 number of keys matched from start index 0::12
6, 7, 524296, 9, 262154, 262155, 262156, 262157, 262158, 262159, 262160, 262161, 262162

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : タイプ、ユーザー、およびセッションでキーを検索する

次のコマンドでは、現在のユーザーとユーザー 3 が使用できる永続 AES キーを検索します (ユーザー 3 は、現在のユーザーが表示できない他のキーを使用できる場合があります)。

```
Command: findKey -t 31 -sess 0 -u 3
```

Example : クラスおよびラベルでキーを検索する

次のコマンドでは、2018-sept ラベルで現在のユーザーのすべてのパブリックキーを検索します。

```
Command: findKey -c 2 -l 2018-sept
```

Example : モジュラスで RSA キーを検索する

次のコマンドでは、m4.txt ファイルのモジュラスを使用して作成された、現在のユーザーの RSA キー (タイプ 0) を検索します。

```
Command: findKey -t 0 -m m4.txt
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-t

指定されたタイプのキーを検索します。キークラスを表す定数を入力します。たとえば、3DES キーを検索するには、-t 21 と入力します。

有効値:

- 0: RSA
- 1: DSA
- 3: EC
- 16: GENERIC_SECRET
- 18: RC4
- 21: Triple DES (3DES)
- 31: AES

必須: いいえ

-c

指定されたクラスのキーを検索します。キークラスを表す定数を入力します。たとえば、パブリックキーを検索するには「`-c 2`」と入力します。

各キータイプに有効な値:

- 2: パブリック。このクラスには、公開 - プライベートのキーペアの公開キーが含まれています。
- 3: プライベート。このクラスには、公開 - プライベートのキーペアの公開キーが含まれています。
- 4: シークレット。このクラスには、対称キーすべてが含まれています。

必須: いいえ

-l

指定されたラベルのキーを検索します。正確なラベルを入力します。`--l` 値にはワイルドカード文字も正規表現も使用できません。

必須: いいえ

-id

指定された ID のキーを検索します。正確な ID 文字列を入力します。`-id` 値にはワイルドカード文字も正規表現も使用できません。

必須: いいえ

-sess

セッションステータスでキーを検索します。現在のセッションでのみ有効なキーを検索するには「1」と入力します。永続キーを検索するには「0」と入力します。

必須: いいえ

-u

指定されたユーザーと現在のユーザーが共有しているキーを検索します。HSM ユーザー ID のカンマ区切りリスト (-u 3 や -u 4,7 など) を入力します。HSM のユーザーの ID を検索するには、listUsers を使用します。

1 つのユーザー ID を指定すると、findKey はそのユーザーのキーを返します。複数のユーザー ID を指定すると、findKey は指定したユーザーすべてが使用できるキーを返します。

findKey は現在のユーザーが使用できるキーのみを返すため、-u の結果は常に、現在のユーザーのキーと同じかそのサブセットです。任意のユーザーが所有または共有しているすべてのキーを取得するには、crypto Officer (COsが cloudhsm_mgmt_util findAllKeysで を使用できます。

必須: いいえ

-m

指定したファイルの RSA モジュラスを使用して作成されたキーを検索します。モジュラスを保存するファイルのパスを入力します。

-m は、一致する RSA モジュラスを含むバイナリファイルを指定します (オプション)。

必須: いいえ

-kcv

指定されたキーのチェック値のキーを検索します。

キーチェック値 (KCV) は、HSM がキーをインポートまたは生成するときに生成されるキーの3バイトのハッシュまたはチェックサムです。キーを エクスポートした後など、HSM の外部で KCV を計算することもできます。次に、KCV 値を比較して、キーのアイデンティティと整合性を確認できます。キーの KCV を取得するには、getAttribute を使用します。

AWS CloudHSM は、次の標準メソッドを使用してキーチェック値を生成します。

- 対称キー: ゼロブロックをキーで暗号化した結果の最初の 3 バイト。
- 非対称キーペア: 公開キーの SHA-1 ハッシュの最初の 3 バイト。
- HMAC キー : 現時点では、HMAC キーの KCV はサポートされていません。

必須: いいえ

出力

findKey の出力には、一致するキーの合計数とそのキーハンドルが一覧表示されます。

```
        Command:  findKey
Total number of keys present 10

 number of keys matched from start index 0::9
6, 7, 8, 9, 10, 11, 262156, 262157, 262158, 262159

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

関連トピック

- findSingleKey
- getKeyInfo
- getAttribute
- findAllKeys cloudhsm_mgmt_util の
- キー属性リファレンス

findSingleKey

key_mgmt_util ツールの findSingleKey コマンドは、クラスターのすべての HSM にキーが存在することを確認します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
findSingleKey -h

findSingleKey -k <key-handle>
```

例

Example

次のコマンドでは、クラスターの 3 つすべての HSM にキー 252136 が存在することを検証します。

```
Command: findSingleKey -k 252136
Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-k

HSM で 1 つのキーのキーハンドルを指定します。このパラメータは必須です。

キーハンドルを見つけるには、findKey コマンドを使用します。

必須: はい

関連トピック

- findKey
- getKeyInfo
- getAttribute

genDSAKeyPair

key_mgmt_util ツールの genDSAKeyPair コマンドは、HSM に デジタル署名アルゴリズム (DSA) キーペアを生成します。ユーザーは、モジュラスの長さを指定する必要があります。モジュラスの値はコマンドで生成されます。ユーザーは、ID を割り当て、他の HSM ユーザーとキーを共有し、

抽出不可のキーとセッション終了時に失効するキーを作成することもできます。コマンドが成功すると、キーハンドルが返されます。HSM は、このキーハンドルをパブリックキーとプライベートキーに割り当てます。このキーハンドルでキーを識別することで、他のコマンドでキーを使用できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、暗号ユーザー (CU) として HSM に ログインする 必要があります。

> **Tip**
>
> タイプ、長さ、ラベル、ID など、作成したキーの属性を検索するには、getAttribute を使用します。特定のユーザーのキーを検索するには、 を使用しますgetKeyInfo。属性値に基づいてキーを検索するには、findKey を使用します。

構文

```
genDSAKeyPair -h

genDSAKeyPair -m <modulus length>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
              [-nex]
              [-sess]
              [-timeout <number of seconds> ]
              [-u <user-ids>]
              [-attest]
```

例

以下の例では、genDSAKeyPair を使用して DSA キーペアを作成する方法を示します。

Example : DSA キーペアを作成する

次のコマンドでは、DSA をラベルとする DSA キーペアを作成します。出力は、パブリックキーのキーハンドルが 19、プライベートキーのキーハンドルが 21 であることを示しています。

```
Command: genDSAKeyPair -m 2048 -l DSA

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS
```

```
        Cfm3GenerateKeyPair:    public key handle: 19    private key handle: 21

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : セッション専用の DSA キーペアを作成する

次のコマンドでは、現在のセッションでのみ有効な DSA キーペアを作成します。コマンドは、必須の (一意でない) ラベルに加えて DSA_temp_pair の一意な ID を割り当てます。次のようなキーペアを作成して、セッション専用のトークンの署名および検証ができます。出力は、パブリックキーのキーハンドルが 12、プライベートキーのキーハンドルが 14 であることを示しています。

```
Command: genDSAKeyPair -m 2048 -l DSA-temp -id DSA_temp_pair -sess

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 12    private key handle: 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

キーペアがセッションでのみ有効であることを確認するには、-sessfindKey の パラメータで、値 1 (true) を使用します。

```
  Command: findKey -sess 1

  Total number of keys present 2

 number of keys matched from start index 0::1
12, 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : 抽出不可の共有 DSA キーペアを作成する

次のコマンドでは、DSA キーペアを作成します。プライベートキーは、他の 3 ユーザーと共有され、HSM からエクスポートすることはできません。パブリックキーは、すべてのユーザーが使用可能であり、常に抽出可能です。

```
Command:  genDSAKeyPair -m 2048 -l DSA -id DSA_shared_pair -nex -u 3,5,6

Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 11    private key handle: 19

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : クォーラム制御のキーペアを作成する

次のコマンドでは、DSA-mV2 をラベルとする DSA キーペアを作成します。このコマンドでは、-u パラメータを使用してプライベートキーをユーザー 4 および 6 と共有します。-m_value パラメータを使用して、プライベートキーを使用する暗号化オペレーションごとに 2 つ以上の承認のクォーラムを要求します。また、-attest パラメータを使用して、キーペアを生成するファームウェアの整合性を検証します。

出力は、コマンドでキーハンドル 12 のパブリックキーとキーハンドル 17 のプライベートキーが生成され、クラスターファームウェアの認証チェックが合格であることを示しています。

```
Command:  genDSAKeyPair -m 2048 -l DSA-mV2 -m_value 2 -u 4,6 -attest

Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 12    private key handle: 17

Attestation Check : [PASS]

Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

このコマンドは、プライベートキー (キーハンドル 17) getKeyInfoで を使用します。出力は、キーの所有者が現在のユーザー (ユーザー 3) であり、キーがユーザー 4 および 6 (それ以外はなし) と共有されていることを示しています。また、クォーラム認証が有効になっていて、クォーラムサイズが 2 であることも示しています。

```
        Command:  getKeyInfo -k 17

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3

        also, shared to following 2 user(s):

                4
                6
         2 Users need to approve to use/manage this key
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-m

モジュラスの長さをビット単位で指定します。唯一の有効な値は 2048 です。

必須: はい

-l

キーペアのユーザー定義ラベルを指定します。文字列を入力します。同じラベルがペアの両方のキーに適用されます。label の最大長は 127 文字です。

キーを識別するのに役立つ任意のフレーズを使用できます。ラベルは一意である必要がないため、このラベルを使用してキーをグループ化および分類できます。

必須: はい

-id

キーペアのユーザー定義識別子 (ID) を指定します。クラスター内で一意の文字列を入力します。デフォルトは空の文字列です。指定した ID は、ペアの両方のキーに適用されます。

デフォルト: ID 値なし。

必須: いいえ

-min_srv

`-timeout` パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-m_value

ペアの秘密キーを使用する暗号化オペレーションを承認する必要があるユーザーの数を指定します。`0` から `8` までの値を入力します。

このパラメータにより、プライベートキーのクォーラム認証要件が確立されます。デフォルト値、`0` で、キーのクォーラム認証機能を無効にします。クォーラム認証が有効になっている場合、指定された数のユーザーがトークンに署名して、プライベートキーを使用する暗号化オペレーション、およびプライベートキーを共有または共有解除するオペレーションを承認する必要があります。

キー`m_value`の を検索するには、 を使用しますgetKeyInfo。

このパラメーターは、コマンドの `-u` パラメーターが `m_value` 要件を満たすのに十分なユーザーとキーペアを共有している場合にのみ有効です。

デフォルト: 0

必須: いいえ

-nex

プライベートキーを抽出できなくなります。生成されたプライベートキーを HSM からエクスポートする ことはできません。公開キーは常に抽出可能です。

デフォルト: キーペアの公開キーとプライベートキーの両方が抽出可能です。

必須: いいえ

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須: いいえ

-timeout

キーが `min_srv` パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、`min_srv` パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須: いいえ

-u

ペアのプライベートキーを指定されたユーザーと共有します。このパラメータは、他の HSM 暗号ユーザー (CU) に暗号化オペレーションでプライベートキーを使用する許可を与えます。公開キーは、共有なしですべてのユーザーが使用可能です。

(`-u 5,6`などの) HSM ユーザー ID のカンマ区切りリストを入力します。現在のユーザーの HSM ユーザー ID を含めないでください。HSM で CU の HSM ユーザー ID を検索するには、listUsers を使用します。既存のキーを共有および共有解除するには、cloudhsm_mgmt_util で shareKey を使用します。

デフォルト: 現在のユーザーのみがプライベートキーを使用できます。

必須: いいえ

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須: いいえ

関連トピック

- genRSAKeyPair
- genSymKey
- genECCKeyPair

genECCKeyPair

key_mgmt_utilツール の `genECCKeyPair` コマンドは、HSM に 楕円曲線暗号(ECC)キーペアを生成します。`genECCKeyPair` コマンドを実行するときは、楕円曲線識別子とキーペアのラベルを指定する必要があります。また、他の CU ユーザーとプライベートキーを共有したり、抽出可能なキー、クォーラム制御キー、およびセッション終了時に失効するキーを作成したりできます。コマンドが成功すると、HSM がパブリックおよびプライベートの ECC キーに割り当てるキーハンドルが返されます。このキーハンドルでキーを識別することで、他のコマンドでキーを使用できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、暗号ユーザー (CU) として HSM に ログインする 必要があります。

> **Tip**
>
> タイプ、長さ、ラベル、ID など、作成したキーの属性を検索するには、getAttribute を使用します。特定のユーザーのキーを検索するには、 を使用しますgetKeyInfo。属性値に基づいてキーを検索するには、findKey を使用します。

構文

```
genECCKeyPair -h
```

```
genECCKeyPair -i <EC curve id>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
              [-nex]
              [-sess]
              [-timeout <number of seconds> ]
              [-u <user-ids>]
              [-attest]
```

例

以下の例では、genECCKeyPair を使用して ECC キーペアを HSM に作成する方法を示します。

Example : ECC キーペアを作成して検査する

次のコマンドでは、NID_secp384r1 楕円曲線と ecc14 ラベルを使用して ECC キーペアを作成します。出力は、プライベートキーのキーハンドルが 262177、パブリックキーのキーハンドルが 262179 であることを示しています。ラベルは、パブリックキーとプライベートキーの両方に適用されます。

```
Command: genECCKeyPair -i 14 -l ecc14

        Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 262179    private key handle: 262177

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

キーを生成した後、その属性を調べることができます。次のコマンドでは、getAttribute を使用して新しい ECC プライベートキーのすべての属性 (定数 512 で表される) を attr_262177 ファイルに書き込みます。

```
Command: getAttribute -o 262177 -a 512 -out attr_262177
got all attributes of size 529 attr cnt 19
Attributes dumped into attr_262177
```

```
        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

次に、cat コマンドを使用して、attr_262177 属性ファイルの内容を表示します。出力は、キーが楕円曲線プライベートキーであり、このキーは署名には使用できるが、暗号化、復号、ラップ、ラップ解除には使用できないことを示しています。キーは永続的で、エクスポート可能です。

```
$  cat attr_262177

OBJ_ATTR_CLASS
0x03
OBJ_ATTR_KEY_TYPE
0x03
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x00
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x01
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
ecc2
OBJ_ATTR_ID

OBJ_ATTR_VALUE_LEN
0x0000008a
OBJ_ATTR_KCV
0xbbb32a
OBJ_ATTR_MODULUS
```

```
044a0f9d01d10f7437d9fa20995f0cc742552e5ba16d3d7e9a65a33e20ad3e569e68eb62477a9960a87911e6121d112
OBJ_ATTR_MODULUS_BITS
0x0000019f
```

Example 無効な EEC 曲線の使用

次のコマンドでは、NID_X9_62_prime192v1 曲線を使用して ECC キーペアの作成を試行します。この楕円曲線は FIPS モードの HSM に対して無効であるため、コマンドは失敗します。クラスターのサーバーが使用不可であることがメッセージで報告されますが、通常、これはクラスターのサーバーに問題があることを示すものではありません。

```
Command:  genECCKeyPair -i 1 -l ecc1

       Cfm3GenerateKeyPair returned: 0xb3 : HSM Error: This operation violates the
 current configured/FIPS policies

       Cluster Error Status
       Node id 0 and err state 0x30000085 : HSM CLUSTER ERROR: Server in cluster is
 unavailable
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-i

楕円曲線の識別子を指定します。識別子を入力します。

有効値:

- 2: NID_X9_62_prime256v1
- 14: NID_secp384r1
- 16: NID_secp256k1

必須: はい

-l

キーペアのユーザー定義ラベルを指定します。文字列を入力します。同じラベルがペアの両方のキーに適用されます。label の最大長は 127 文字です。

キーを識別するのに役立つ任意のフレーズを使用できます。ラベルは一意である必要がないため、このラベルを使用してキーをグループ化および分類できます。

必須: はい

-id

キーペアのユーザー定義識別子 (ID) を指定します。クラスター内で一意の文字列を入力します。デフォルトは空の文字列です。指定した ID は、ペアの両方のキーに適用されます。

デフォルト: ID 値なし。

必須: いいえ

-min_srv

`-timeout` パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-m_value

ペアの秘密キーを使用する暗号化オペレーションを承認する必要があるユーザーの数を指定します。`0` から `8` までの値を入力します。

このパラメータにより、プライベートキーのクォーラム認証要件が確立されます。デフォルト値、`0` で、キーのクォーラム認証機能を無効にします。クォーラム認証が有効になっている場合、指定された数のユーザーがトークンに署名して、プライベートキーを使用する暗号化オペレーション、およびプライベートキーを共有または共有解除するオペレーションを承認する必要があります。

キー`m_value`の を検索するには、 を使用しますgetKeyInfo。

このパラメーターは、コマンドの `-u` パラメーターが `m_value` 要件を満たすのに十分なユーザーとキーペアを共有している場合にのみ有効です。

デフォルト: 0

必須: いいえ

-nex

プライベートキーを抽出できなくなります。生成されたプライベートキーを HSM からエクスポートする ことはできません。公開キーは常に抽出可能です。

デフォルト: キーペアの公開キーとプライベートキーの両方が抽出可能です。

必須: いいえ

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須: いいえ

-timeout

キーが `min_srv` パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、`min_srv` パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須: いいえ

-u

ペアのプライベートキーを指定されたユーザーと共有します。このパラメータは、他の HSM 暗号ユーザー (CU) に暗号化オペレーションでプライベートキーを使用する許可を与えます。公開キーは、共有なしですべてのユーザーが使用可能です。

(-u 5,6などの) HSM ユーザー ID のカンマ区切りリストを入力します。現在のユーザーの HSM ユーザー ID を含めないでください。HSM で CU の HSM ユーザー ID を検索するには、listUsers を使用します。既存のキーを共有および共有解除するには、cloudhsm_mgmt_util で shareKey を使用します。

デフォルト: 現在のユーザーのみがプライベートキーを使用できます。

必須: いいえ

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須: いいえ

関連トピック

- genSymKey
- genRSAKeyPair
- genDSAKeyPair

genRSAKeyPair

key_mgmt_util ツールの genRSAKeyPair コマンドは、RSA 非対称キーペアを生成します。ユーザーは、キーのタイプ、モジュラスの長さ、および公開指数を指定します。コマンドは、指定した長さのモジュラスを生成し、キーペアを作成します。ユーザーは、ID を割り当て、他の HSM ユーザーとプライベートキーを共有し、抽出不可のキーとセッション終了時に失効するキーを作成できます。コマンドが成功すると、HSM がキーに割り当てるキーハンドルが返されます。このキーハンドルでキーを識別して他のコマンドで使用できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、暗号ユーザー (CU) として HSM に ログインする 必要があります。

> **Tip**
>
> タイプ、長さ、ラベル、ID など、作成したキーの属性を検索するには、getAttribute を使用します。特定のユーザーのキーを検索するには、 を使用しますgetKeyInfo。属性値に基づいてキーを検索するには、findKey を使用します。

構文

```
genRSAKeyPair -h

genRSAKeyPair -m <modulus length>
              -e <public exponent>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
              [-nex]
              [-sess]
              [-timeout <number of seconds> ]
              [-u <user-ids>]
              [-attest]
```

例

以下の例では、genRSAKeyPair を使用して非対称キーペアを HSM に作成する方法を示します。

Example : RSA キーペアを作成して検査する

このコマンドでは、モジュラス 2048 ビットで指数が 65537 の RSA キーペアを作成します。出力は、パブリックキーのハンドルが 2100177、プライベートキーのハンドルが 2100426 であることを示しています。

```
Command: genRSAKeyPair -m 2048 -e 65537 -l rsa_test

Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 2100177    private key handle:
 2100426

        Cluster Status:
        Node id 0 status: 0x00000000 : HSM Return: SUCCESS
```

```
        Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、getAttribute を使用して、先ほど作成したパブリックキーの属性を取得します。出力の書き込み先は attr_2100177 ファイルです。この属性ファイルの内容を取得する cat コマンドが続けて実行されます。キー属性の解釈については、キー属性リファレンス を参照してください。

結果の 16 進値は、RSA タイプ (OBJ_ATTR_CLASS 0x02) のパブリックキー (OBJ_ATTR_KEY_TYPE 0x00) であることを示しています。この公開キーを使用して暗号化 (OBJ_ATTR_ENCRYPT 0x01) はできますが、復号 (OBJ_ATTR_DECRYPT 0x00) を行うことはできません。結果には、キーの長さ (512、0x200)、モジュラス、モジュラスの長さ (2048、0x800)、およびパブリック指数 (65537、0x10001) も含まれています。

```
Command:  getAttribute -o 2100177 -a 512 -out attr_2100177

Attribute size: 801, count: 26
Written to: attr_2100177 file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS

$  cat attr_2100177
OBJ_ATTR_CLASS
0x02
OBJ_ATTR_KEY_TYPE
0x00
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x01
OBJ_ATTR_LOCAL
```

```
0x01
OBJ_ATTR_SENSITIVE
0x00
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
rsa_test
OBJ_ATTR_ID

OBJ_ATTR_VALUE_LEN
0x00000200
OBJ_ATTR_KCV
0xc51c18
OBJ_ATTR_MODULUS
0xbb9301cc362c1d9724eb93da8adab0364296bde7124a241087d9436b9be57e4f7780040df03c2c
1c0fe6e3b61aa83c205280119452868f66541bbbffacbbe787b8284fc81deaeef2b8ec0ba25a077d
6983c77a1de7b17cbe8e15b203868704c6452c2810344a7f2736012424cf0703cf15a37183a1d2d0
97240829f8f90b063dd3a41171402b162578d581980976653935431da0c1260bfe756d85dca63857
d9f27a541676cb9c7def0ef6a2a89c9b9304bcac16fdf8183c0a555421f9ad5dfeb534cf26b65873
970cdf1a07484f1c128b53e10209cc6f7ac308669112968c81a5de408e7f644fe58b1a9ae1286fec
b3e4203294a96fae06f8f0db7982cb5d7f
OBJ_ATTR_MODULUS_BITS
0x00000800
OBJ_ATTR_PUBLIC_EXPONENT
0x010001
OBJ_ATTR_TRUSTED
0x00
OBJ_ATTR_WRAP_WITH_TRUSTED
0x00
OBJ_ATTR_DESTROYABLE
0x01
OBJ_ATTR_DERIVE
0x00
OBJ_ATTR_ALWAYS_SENSITIVE
0x00
OBJ_ATTR_NEVER_EXTRACTABLE
0x00
```

Example : 共有 RSA キーペアを生成する

次のコマンドでは、RSA キーペアを生成し、HSM の別の CU であるユーザー 4 とプライベートキーを共有します。コマンドでは、m_value パラメータを使用して少なくとも 2 つの承認を要求した上で、ペアのプライベートキーを暗号化オペレーションで使用できるようにします。m_value パラ

メータを使用する場合、コマンドで -u も使用する必要があります。これにより、m_value がユーザーの合計数 (-u の数値 + 所有者) を超えないようにします。

```
Command:  genRSAKeyPair -m 2048 -e 65537 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2

       Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

       Cfm3GenerateKeyPair:    public key handle: 27    private key handle: 28

       Cluster Error Status
       Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
       Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-m

モジュラスの長さをビット単位で指定します。最小値は 2048 です。

必須: はい

-e

パブリック指数を指定します。値は、65537 以上の奇数にする必要があります

必須: はい

-l

キーペアのユーザー定義ラベルを指定します。文字列を入力します。同じラベルがペアの両方のキーに適用されます。label の最大長は 127 文字です。

キーを識別するのに役立つ任意のフレーズを使用できます。ラベルは一意である必要がないため、このラベルを使用してキーをグループ化および分類できます。

必須: はい

-id

キーペアのユーザー定義識別子 (ID) を指定します。クラスター内で一意の文字列を入力します。デフォルトは空の文字列です。指定した ID は、ペアの両方のキーに適用されます。

デフォルト: ID 値なし。

必須: いいえ

-min_srv

`-timeout` パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-m_value

ペアの秘密キーを使用する暗号化オペレーションを承認する必要があるユーザーの数を指定します。`0` から 8 までの値を入力します。

このパラメータにより、プライベートキーのクォーラム認証要件が確立されます。デフォルト値、`0` で、キーのクォーラム認証機能を無効にします。クォーラム認証が有効になっている場合、指定された数のユーザーがトークンに署名して、プライベートキーを使用する暗号化オペレーション、およびプライベートキーを共有または共有解除するオペレーションを承認する必要があります。

キー`m_value`の を検索するには、 を使用しますgetKeyInfo。

このパラメーターは、コマンドの `-u` パラメーターが `m_value` 要件を満たすのに十分なユーザーとキーペアを共有している場合にのみ有効です。

デフォルト: 0

必須: いいえ

-nex

プライベートキーを抽出できなくなります。生成されたプライベートキーを HSM からエクスポートする ことはできません。公開キーは常に抽出可能です。

デフォルト: キーペアの公開キーとプライベートキーの両方が抽出可能です。

必須: いいえ

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須: いいえ

-timeout

キーが `min_srv` パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、`min_srv` パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須: いいえ

-u

ペアのプライベートキーを指定されたユーザーと共有します。このパラメータは、他の HSM 暗号ユーザー (CU) に暗号化オペレーションでプライベートキーを使用する許可を与えます。公開キーは、共有なしですべてのユーザーが使用可能です。

(`-u 5,6`などの) HSM ユーザー ID のカンマ区切りリストを入力します。現在のユーザーの HSM ユーザー ID を含めないでください。HSM で CU の HSM ユーザー ID を検索するには、listUsers

を使用します。既存のキーを共有および共有解除するには、cloudhsm_mgmt_util で shareKey を使用します。

デフォルト: 現在のユーザーのみがプライベートキーを使用できます。

必須: いいえ

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須: いいえ

関連トピック

- genSymKey
- genDSAKeyPair
- genECCKeyPair

genSymKey

key_mgmt_util ツールの genSymKey コマンドは、HSM に対称キーを生成します。キーのタイプとサイズを指定し、ID とラベルを割り当て、他の HSM ユーザーとキーを共有することができます。また、抽出不可のキーや、セッションが終了すると同時に失効するキーを作成することもできます。コマンドが成功すると、HSM がキーに割り当てるキーハンドルが返されます。このキーハンドルでキーを識別して他のコマンドで使用できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
genSymKey -h

genSymKey -t <key-type>
          -s <key-size>
          -l <label>
```

```
        [-id <key-ID>]
        [-min_srv <minimum-number-of-servers>]
        [-m_value <0..8>]
        [-nex]
        [-sess]
        [-timeout <number-of-seconds> ]
        [-u <user-ids>]
        [-attest]
```

例

以下の例では、genSymKey を使用して HSM に対称キーを作成する方法を示しています。

> **Tip**
>
> これらの例で作成したキーをHMACオペレーションに使用するには、OBJ_ATTR_SIGN キーを生成した後に OBJ_ATTR_VERIFY と TRUE を設定する必要があります。これらの値を設定するには、CloudHSM 管理ユーティリティ (CMU) で setAttribute を使用します。詳細については、setAttribute を参照してください。

Example : AES キーの生成

このコマンドは、aes256 というラベルを持つ 256 ビット AES キーを作成します。出力は、新しいキーのキーハンドルが 6 であることを示します。

```
Command: genSymKey -t 31 -s 32 -l aes256

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 6

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : セッションキーの作成

次のコマンドは、現在のセッションでのみ有効な、抽出不可の 192 ビット AES キーを作成します。次のようなキーを作成して、エクスポートするキーをラップ (および直後にラップ解除) することができます。

```
Command: genSymKey -t 31 -s 24 -l tmpAES -id wrap01 -nex -sess
```

Example : 迅速に戻る

次のコマンドでは、IT_test_key をラベルとする 512 バイトの汎用キーを作成します。このコマンドは、キーがクラスターのすべての HSM に同期されるまで待機しません。代わりに、いずれかの HSM でキーが作成された時点 (-min_srv 1) または 1 秒 (-timeout 1) のいずれか短い方で戻ります。タイムアウトが経過する前に、指定した最小数の HSM にキーが同期されない場合、キーは生成されません。次の例の for ループのように、多数のキーを作成するスクリプトでこのようなコマンドを使用できます。

```
Command: genSymKey -t 16 -s 512 -l IT_test_key -min_srv 1 -timeout 1

$  for i in {1..30};
     do /opt/cloudhsm/bin/key_mgmt_util singlecmd loginHSM -u CU -s example_user -p
 example_pwd genSymKey -l aes -t 31 -s 32 -min_srv 1 -timeout 1;
 done;
```

Example : クォーラム認証汎用キーの作成

次のコマンドでは、generic-mV2 をラベルとする 2048 ビットの汎用シークレットキーを作成します。このコマンドでは、-u パラメータを使用して別の CU、ユーザー 6、とキーを共有します。-m_value パラメータを使用して、キーを使用するすべての暗号オペレーションで 2 つ以上のクォーラムの承認を要求します。また、このコマンドでは、-attest パラメータを使用して、キーが生成されたファームウェアの整合性を検証します。

出力は、コマンドがキーハンドル 9 でキーを生成し、クラスターファームウェアの認証チェックが成功したことを示しています。

```
                Command:  genSymKey -t 16 -s 2048 -l generic-mV2 -m_value 2 -u 6 -
attest

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 9

        Attestation Check : [PASS]

        Cluster Error Status
```

```
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : キーの作成と検証

このコマンドは、ラベルが 3DES_shared で ID が IT-02 の Triple DES キーを作成します。現在のユーザーと、ユーザー 4 およびユーザー 5 がキーを使用できます。クラスター内で ID が一意でない場合、または現在のユーザーがユーザー 4 またはユーザー 5 の場合、コマンドは失敗します。

出力は、新規キーのキーハンドルが 7 であることを示しています。

```
Command: genSymKey -t 21 -s 24 -l 3DES_shared -id IT-02 -u 4,5

       Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 7

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

新しい 3DES キーが現在のユーザーによって所有され、ユーザー 4 とユーザー 5 と共有されていることを確認するには、getKeyInfo を使用します。このコマンドは、新しいキーに割り当てられたハンドル (Key Handle: 7) を使用します。

出力は、キーの所有者がユーザー 3 で、キーをユーザー 4 とユーザー 5 が共有していることを示しています。

```
Command:  getKeyInfo -k 7

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3

        also, shared to following 2 user(s):

                4, 5
```

キーの他のプロパティを確認するには、getAttribute を使用します。最初のコマンドでは、getAttribute を使用して、キーハンドル 7 (-o 7) のすべての属性 (-a 512) を取得します。

それを attr_7 ファイルに書き込みます。2 番目のコマンドは、cat を使用して、attr_7 ファイルの内容を取得します。

このコマンドは、キー 7 が 192 ビット (OBJ_ATTR_VALUE_LEN 0x00000018 または 24 バイト) 3DES (OBJ_ATTR_KEY_TYPE 0x15) 対称キー (OBJ_ATTR_CLASS 0x04) で、ラベルが 3DES_shared (OBJ_ATTR_LABEL 3DES_shared)、ID が IT_02 (OBJ_ATTR_ID IT-02) であることを示しています。このキーは永続的 (OBJ_ATTR_TOKEN 0x01) および抽出可能 (OBJ_ATTR_EXTRACTABLE 0x01) で、暗号化、復号、およびラッピングに使用できます。

> **Tip**
>
> タイプ、長さ、ラベル、ID など、作成したキーの属性を検索するには、getAttribute を使用します。特定のユーザーのキーを検索するには、 を使用しますgetKeyInfo。属性値に基づいてキーを検索するには、findKey を使用します。

キー属性の解釈については、 キー属性リファレンス を参照してください。

```
Command:  getAttribute -o 7 -a 512 -out attr_7

got all attributes of size 444 attr cnt 17
Attributes dumped into attr_7 file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS


$  cat attr_7

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
```

```
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
3DES_shared
OBJ_ATTR_ID
IT-02
OBJ_ATTR_VALUE_LEN
0x00000018
OBJ_ATTR_KCV
0x59a46e
```

> **Tip**
>
> これらの例で作成したキーをHMACオペレーションに使用するには、OBJ_ATTR_SIGN キーを生成した後に OBJ_ATTR_VERIFY と TRUE を設定する必要があります。これらの値を設定するには、CMUで setAttribute を使用します。詳細については、setAttribute を参照してください。

パラメータ

-h

コマンドに関するヘルプを表示します。

必須：はい

-t

対称キーのタイプを指定します。キーのタイプを表す定数を入力します。たとえば、AES キーを作成するには「`-t 31`」と入力します。

有効値:

- 16: GENERIC_SECRET。汎用シークレットキー は、AES キーの要件など、特定のスタンダードに準拠していないバイト配列です。
- 18: RC4。RC4 キーは FIPS モードの HSM では無効です
- 21: Triple DES (3DES)。NIST ガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。
- 31: AES

必須：はい

-s

キーのサイズをバイト単位で指定します。たとえば、192 ビットのキーを作成するには「24」と入力します。

各キータイプに有効な値:

- AES: 16 (128 ビット)、24 (192 ビット)、32 (256 ビット)
- 3DES: 24 (192 ビット)
- 汎用シークレット: <3584 (28672 ビット)

必須：はい

-l

キーのユーザー定義ラベルを指定します。文字列を入力します。

キーを識別するのに役立つ任意のフレーズを使用できます。ラベルは一意である必要がないため、このラベルを使用してキーをグループ化および分類できます。

必須：はい

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須：いいえ

-id

キーのユーザー定義識別子を指定します。クラスター内で一意の文字列を入力します。デフォルトは空の文字列です。

デフォルト: ID 値なし。

必須：いいえ

-min_srv

`-timeout` パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須：いいえ

-m_value

キーを使用する暗号化オペレーションを承認する必要があるユーザーの数を指定します。`0` から `8` までの値を入力します。

このパラメータは、キーのクォーラム認証要件を確立します。デフォルト値、`0` で、キーのクォーラム認証機能を無効にします。クォーラム認証が有効になっている場合、指定された数のユーザーは、キーを使用する暗号化オペレーション、およびキーを共有または共有解除するオペレーションを承認するためにトークンに署名する必要があります。

キー`m_value`の を検索するには、 を使用しますgetKeyInfo。

このパラメータが有効なのは、コマンドの `-u` パラメータが `m_value` の要件を満たすために十分な数のユーザーとキーを共有するときのみです。

デフォルト: 0

必須：いいえ

-nex

キーを抽出できなくなります。生成されたキーは HSM からエクスポートできません。

デフォルト: キーは抽出可能です。

必須：いいえ

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須：いいえ

-timeout

キーが `min_srv` パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、`min_srv` パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須：いいえ

-u

指定されたユーザーとキーを共有します。このパラメータは、別の HSM Crypto User (CU) に、暗号化オペレーションでこのキーを使用するアクセス許可を付与します。

(`-u 5,6`などの) HSM ユーザー ID のカンマ区切りリストを入力します。現在のユーザーの HSM ユーザー ID を含めないでください。HSM で CU の HSM ユーザー ID を検索するには、listUsers を使用します。既存のキーを共有および共有解除するには、cloudhsm_mgmt_util で shareKey を使用します。

デフォルト：現在のユーザーのみがキーを使用できます。

必須：いいえ

関連トピック

- exSymKey
- genRSAKeyPair
- genDSAKeyPair
- genECCKeyPair
- setAttribute

getAttribute

key_mgmt_util の getAttribute コマンドは、 AWS CloudHSM キーの属性値の 1 つまたはすべてをファイルに書き込みます。AES キーのモジュラスなど、キータイプに指定した属性が存在しない場合は、getAttribute はエラーを返します。

キー属性はキーのプロパティです。キー属性には、キータイプ、クラス、ラベル、ID などの特性と、キーで実行できるアクション (暗号化、復号、ラップ、署名、検証など) を表す値が含まれています。

getAttribute は、所有しているキーと共有しているキーに対してのみ使用できます。このコマンドまたは cloudhsm_mgmt_util で getAttribute コマンドを実行できます。このコマンドは、クラスターのすべての HSM からキーの1つの属性値を取得し、それを stdout またはファイルに書き込みます。

属性とそれを表す定数のリストを取得するには、listAttributes コマンドを使用します。既存のキーの属性値を変更するには、key_mgmt_util の setAttribute および cloudhsm_mgmt_util の setAttribute を使用します。キー属性の解釈については、キー属性リファレンス を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
getAttribute -h

getAttribute -o <key handle>
```

```
            -a <attribute constant>
            -out <file>
```

例

以下の例では、getAttribute を使用して HSM でキーの属性を取得する方法を示します。

Example : キータイプを取得する

次の例では、AES、3DES、汎用キーなどのキータイプ、RSA または楕円曲線キーペアを取得します。

最初のコマンドでは listAttributes を実行し、キーの属性およびそれを表す定数を取得します。出力は、キータイプの定数が 256 であることを示しています。キー属性の解釈については、キー属性リファレンス を参照してください。

```
Command: listAttributes

Description
===========
The following are all of the possible attribute values for getAttributes.

      OBJ_ATTR_CLASS                = 0
      OBJ_ATTR_TOKEN                = 1
      OBJ_ATTR_PRIVATE              = 2
      OBJ_ATTR_LABEL                = 3
      OBJ_ATTR_KEY_TYPE             = 256
      OBJ_ATTR_ID                   = 258
      OBJ_ATTR_SENSITIVE            = 259
      OBJ_ATTR_ENCRYPT              = 260
      OBJ_ATTR_DECRYPT              = 261
      OBJ_ATTR_WRAP                 = 262
      OBJ_ATTR_UNWRAP               = 263
      OBJ_ATTR_SIGN                 = 264
      OBJ_ATTR_VERIFY               = 266
      OBJ_ATTR_LOCAL                = 355
      OBJ_ATTR_MODULUS              = 288
      OBJ_ATTR_MODULUS_BITS         = 289
      OBJ_ATTR_PUBLIC_EXPONENT      = 290
      OBJ_ATTR_VALUE_LEN            = 353
      OBJ_ATTR_EXTRACTABLE          = 354
      OBJ_ATTR_KCV                  = 371
```

2 番目のコマンドでは getAttribute を実行します。これは、キーハンドル 524296 のキータイプ (属性 256) をリクエストし、attribute.txt ファイルに書き込みます。

```
Command: getAttribute -o 524296 -a 256 -out attribute.txt
Attributes dumped into attribute.txt file
```

最後のコマンドでは、キーファイルの内容を取得します。出力は、キータイプが 0x15 または 21 の Triple DES (3 DES) キーであることを示しています。クラスとタイプの値の定義については、「キー属性リファレンス」を参照してください。

```
$  cat attribute.txt
OBJ_ATTR_KEY_TYPE
0x00000015
```

Example : キーのすべての属性を取得する

次のコマンドでは、キーハンドル 6 でキーのすべての属性を取得し、attr_6 ファイルに書き込みます。すべての属性を表す属性値として 512 を使用します。

```
Command: getAttribute -o 6 -a 512 -out attr_6

got all attributes of size 444 attr cnt 17
Attributes dumped into attribute.txt file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS>
```

次のコマンドでは、すべての属性の値とサンプルの属性ファイルのコンテンツを示します。値の中で、キーは 256 ビット AES キーで ID は test_01、ラベルは aes256 であることを示しています。キーは抽出可能で永続的であり、セッション専用キーではありません。キー属性の解釈については、キー属性リファレンス を参照してください。

```
$  cat attribute.txt

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
```

```
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x01
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
aes256
OBJ_ATTR_ID
test_01
OBJ_ATTR_VALUE_LEN
0x00000020
OBJ_ATTR_KCV
0x1a4b31
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-オ

ターゲットキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、findKey を使用します。

また、指定するキーは所有しているか、共有している必要があります。キーのユーザーを検索するには、 を使用しますgetKeyInfo。

必須: はい

-a

属性を識別します。属性を表す定数を入力するか、すべての属性を表す 512 を入力します。たとえば、キーの種類を取得するには「256」と入力します。これは OBJ_ATTR_KEY_TYPE 属性を表す定数です。

属性とその定数のリスト化するために、listAttributes を使用します。キー属性の解釈については、キー属性リファレンス を参照してください。

必須: はい

-out

指定したファイルに出力を書き込みます。ファイルパスを入力します。出力を stdout に書き込むことはできません。

指定したファイルが既に存在する場合、getAttribute は警告なしにそのファイルを上書きします。

必須: はい

関連トピック

- cloudhsm_mgmt_util の中の setAttribute です。
- listAttributes
- setAttribute
- findKey
- キー属性リファレンス

getCaviumPrivキー

key_mgmt_util の getCaviumPrivKey コマンドは、フェイク PEM 形式で HSM からプライベートキーをエクスポートします。フェイク PEM ファイルは、実際のプライベートキーマテリアルを含むわけではなく、HSM のプライベートキーを参照し、ウェブサーバーから AWS CloudHSMへの SSL/TLS オフロードを確立するために使用できます。詳細については、「Linux 上の SSL/TLS オフロード」を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、暗号ユーザー (CU) として HSM に ログインする 必要があります。

構文

```
getCaviumPrivKey -h

getCaviumPrivKey -k <private-key-handle
                 -out <fake-PEM-file>
```

例

この例では、getCaviumPrivKey を使ってプライベートキーをフェイク PEM 形式でエクスポートする方法を示します。

Example : フェイク PEM ファイルをエクスポートする

このコマンドは、ハンドルが 15 のプライベートキーのフェイク PEM バージョンを作成してエクスポートし、cavKey.pem という名前のファイルを保存します。exportPrivateKey は、コマンドが成功すると成功メッセージを返します。

```
Command: getCaviumPrivKey -k 15 -out cavKey.pem

Private Key Handle is written to cavKey.pem in fake PEM format

        getCaviumPrivKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-k

フェイク PEM 形式でエクスポートするプライベートキーのキーハンドルを指定します。

必須: はい

-out

フェイク PEM キーの書き込み先とするファイルの名前を指定します。

必須: はい

関連トピック

- importPrivateKey
- Linux 上の SSL/TLS オフロード

getCert

key_mgmt_util の getCert コマンドは、HSM のパーティション証明書を取得し、それらをファイルに保存します。コマンドを実行する際、取得する証明書のタイプを指定します。そのためには、以下の「パラメータ」セクションで説明されているように、いずれかの整数を使用します。これらの各証明書のロールについては、「HSM のアイデンティティの確認」を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
getCert -h

getCert -f <file-name>
        -t <certificate-type>
```

例

この例では、getCert を使用して、クラスターのお客様のルート証明書を取得し、ファイルに保存する方法を示します。

Example : お客様のルート証明書を取得する

このコマンドは、お客様のルート証明書 (整数 4 で表現) をエクスポートし、userRoot.crt というファイルに保存します。getCert は、コマンドが成功すると成功メッセージを返します。

```
Command: getCert -f userRoot.crt -s 4

Cfm3GetCert() returned 0 :HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-f

取得された証明書の保存先とするファイルの名前を指定します。

必須: はい

-s

取得するパーティション証明書のタイプを指定する整数。整数とその証明書タイプは次のとおりです。

- 1 - 製造元のルート証明書。
- 2 - 製造元のハードウェア証明書。
- 4 - お客様のルート証明書。
- 8 - (お客様のルート証明書で署名されている)クラスターの証明書。
- 16 - (製造元のルート証明書に連鎖されている)クラスターの証明書。

必須: はい

関連トピック

- HSM のアイデンティティの確認
- (cloudhsm_mgmt_util の) getKeyInfo

getKeyInfo

key_mgmt_util の getKeyInfo コマンドは、キーを使用できるユーザーの HSM ユーザー ID を返します。これには、所有者およびキーを共有する暗号ユーザー (CU) も含まれます。キーに対するクォーラム認証が有効になっている場合、getKeyInfo はキーを使用する暗号化オペレーションを承認する必要があるユーザーの数も返します。getKeyInfo は、所有および共有しているキーに対してのみ実行できます。

パブリックキーに対して getKeyInfo を実行すると、HSM のすべてのユーザーがパブリックキーを使用できる場合でも、getKeyInfo はキー所有者のみを返します。HSM のユーザーの HSM ユーザー ID を確認するには、listUsers を使用します。特定のユーザーのキーを確認するには、findKey -u を使用します。

ユーザーは、自分で作成したキーを所有します。自分で作成したキーは、他のユーザーと共有できます。既存のキーを共有または共有解除するには、cloudhsm_mgmt_util の shareKey を使用します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする必要があります。

Syntax

```
getKeyInfo -h

getKeyInfo -k <key-handle>
```

例

以下の例では、getKeyInfo を使用してキーのユーザーに関する情報を取得する方法を示します。

Example : 対称キーのユーザーを取得する

次のコマンドでは、キーハンドルが 9 の AES (対称) キーを使用できるユーザーを取得します。出力は、キーの所有者がユーザー 3 であり、キーをユーザー 4 と共有していることを示しています。

```
Command:  getKeyInfo -k 9

       Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3

       also, shared to following 1 user(s):

                4
```

Example : 非対称キーペアのユーザーを取得する

以下のコマンドでは、getKeyInfo を使用して RSA (非対称) キーペアのキーを使用できるユーザーを取得します。パブリックキーのキーハンドルは 21 です。プライベートキーのキーハンドルは 20 です。

プライベートキー (getKeyInfo) に対して 20 を実行すると、キー所有者 (3) およびキーを共有している Crypto User (CU) 4 と 5 が返されます。

```
Command:  getKeyInfo -k 20

       Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3

       also, shared to following 2 user(s):

                4
                5
```

getKeyInfo をパブリックキー (21) に対して実行すると、キー所有者 (3) のみが返されます。

```
Command:  getKeyInfo -k 21

       Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3
```

ユーザー 4 がパブリックキー (および HSM のすべてのパブリックキー) を使用できることを確認するには、-ufindKey の パラメータを使用します。

出力は、ユーザー 4 がキーペアのパブリックキー (21) とプライベートキー (20) の両方を使用できることを示しています。ユーザー 4 は、他のすべてのパブリックキーと、自分で作成したプライベートキーまたは共有しているプライベートキーを使用することもできます。

```
Command:  findKey -u 4
Total number of keys present 8

 number of keys matched from start index 0::7
11, 12, 262159, 262161, 262162, 19, 20, 21

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : キーのクォーラム認証値 (m_value) を取得する

この例では、キーの m_value、つまりキーを使用する暗号化オペレーションを承認する必要があるクォーラムのユーザー数を取得する方法を示します。

キーに対してクォーラム認証を有効にすると、ユーザーのクォーラムは、そのキーを使用する暗号化オペレーションを承認する必要があります。クォーラム認証を有効にしてクォーラムサイズを設定するには、キーの作成時に -m_value パラメータを使用します。

このコマンドは genRSAKeyPair を使用して、ユーザー 4 と共有される RSA キーペアを作成します。また、m_value パラメータを使用してペアのプライベートキーでクォーラム認証を有効にし、クォーラムサイズを 2 ユーザーに設定します。ユーザー数は必要な承認を提供できるだけの大きさが必要です。

出力は、このコマンドでパブリックキー 27 とプライベートキー 28 が作成されたことを示しています。

```
Command:  genRSAKeyPair -m 2048 -e 195193 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2

       Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

       Cfm3GenerateKeyPair:    public key handle: 27    private key handle: 28

       Cluster Error Status
       Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
       Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、getKeyInfo を使用して、プライベートキーのユーザーに関する情報を取得します。出力は、キーの所有者がユーザー 3 であり、キーがユーザー 4 と共有されていることを示しています。また、2 ユーザーのクォーラムが、このキーを使用するすべての暗号化オペレーションを承認する必要があることも示しています。

```
Command:  getKeyInfo -k 28

       Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3

       also, shared to following 1 user(s):

                4
```

```
        2 Users need to approve to use/manage this key
```

パラメータ

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-k

HSM で 1 つのキーのキーハンドルを指定します。所有または共有するキーのキーハンドルを入力します。このパラメータは必須です。

キーハンドルを見つけるには、findKey コマンドを使用します。

必須: はい

関連トピック

- getKeyInfo cloudhsm_mgmt_util の
- listUsers
- findKey
- findAllKeys cloudhsm_mgmt_util の

ヘルプ

key_mgmt_util の help コマンドは、使用可能なすべての key_mgmt_util コマンドに関する情報を表示します。

help を実行する前に、key_mgmt_util を起動する必要があります。

Syntax

```
help
```

例

次の例は、help コマンドの出力を示します。

Example

```
Command:  help

Help Commands Available:

Syntax: <command> -h


   Command               Description
   =======               ===========

   exit                   Exits this application
   help                   Displays this information

        Configuration and Admin Commands
   getHSMInfo             Gets the HSM Information
   getPartitionInfo       Gets the Partition Information
   listUsers              Lists all users of a partition
   loginStatus            Gets the Login Information
   loginHSM               Login to the HSM
   logoutHSM              Logout from the HSM

        M of N commands
   getToken               Initiate an MxN service and get Token
   delToken               delete Token(s)
   approveToken           Approves an MxN service
   listTokens             List all Tokens in the current partition

        Key Generation Commands

        Asymmetric Keys:
   genRSAKeyPair          Generates an RSA Key Pair
   genDSAKeyPair          Generates a DSA Key Pair
   genECCKeyPair          Generates an ECC Key Pair

        Symmetric Keys:
   genPBEKey              Generates a PBE DES3 key
   genSymKey              Generates a Symmetric keys

        Key Import/Export Commands
   createPublicKey        Creates an RSA public key
   importPubKey           Imports RSA/DSA/EC Public key
   exportPubKey           Exports RSA/DSA/EC Public key
```

```
    importPrivateKey       Imports RSA/DSA/EC private key
    exportPrivateKey       Exports RSA/DSA/EC private key
    imSymKey               Imports a Symmetric key
    exSymKey               Exports a Symmetric key
    wrapKey                Wraps a key from from HSM using the specified handle
    unWrapKey              UnWraps a key into HSM using the specified handle

         Key Management Commands
    deleteKey              Delete Key
    setAttribute           Sets an attribute of an object
    getKeyInfo             Get Key Info about shared users/sessions
    findKey                Find Key
    findSingleKey          Find single Key
    getAttribute           Reads an attribute from an object

         Certificate Setup Commands
    getCert                Gets Partition Certificates stored on HSM

         Key Transfer Commands
    insertMaskedObject     Inserts a masked object
    extractMaskedObject    Extracts a masked object

         Management Crypto Commands
    sign                   Generates a signature
    verify                 Verifies a signature
    aesWrapUnwrap          Does NIST AES Wrap/Unwrap

         Helper Commands
    Error2String           Converts Error codes to Strings
                           save key handle in fake PEM format
    getCaviumPrivKey       Saves an RSA private key handle
                           in fake PEM format
    IsValidKeyHandlefile   Checks if private key file has
                           an HSM key handle or a real key
    listAttributes         List all attributes for getAttributes
    listECCCurveIds        List HSM supported ECC CurveIds
```

パラメータ

このコマンドにはパラメータがありません。

関連トピック

- loginHSM および logoutHSM

importPrivateKey

key_mgmt_util の importPrivateKey コマンドは、非対称プライベートキーをファイルから HSM にインポートします。HSM では、クリアテキストのキーを直接インポートすることはできません。このコマンドは、指定した AES ラップキーを使用してプライベートキーを暗号化し、HSM 内のキーをラップ解除します。 AWS CloudHSM キーを証明書に関連付ける場合は、このトピック を参照してください。

> **Note**
>
> パスワードで保護された PEM キーは、対称キーまたはプライベートキーを使用してインポートすることはできません。

OBJ_ATTR_UNWRAP と OBJ_ATTR_ENCRYPT 属性値 1 を含む AES ラップキーを指定する必要があります。キーの属性を確認するには、getAttribute コマンドを使用します。

> **Note**
>
> このコマンドでは、インポートしたキーをエクスポート不可として指定するオプションは使用できません。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
importPrivateKey -h

importPrivateKey -l <label>
                 -f <key-file>
                 -w <wrapping-key-handle>
                 [-sess]
                 [-id <key-id>]
                 [-m_value <0...8>]
                 [min_srv <minimum-number-of-servers>]
                 [-timeout <number-of-seconds>]
                 [-u <user-ids>]
                 [-wk <wrapping-key-file>]
```

```
                  [-attest]
```

例

この例では、importPrivateKey を使ってプライベートキーを HSM にインポートする方法を示します。

Example : プライベートキーをインポートする

このコマンドは、rsa2048.key というラベルとハンドルが 524299 のラップキーを使って、rsa2048-imported という名前のファイルからプライベートキーをインポートします。importPrivateKey コマンドは、成功すると、インポートされたキーのキーハンドルと成功メッセージを返します。

```
Command: importPrivateKey -f rsa2048.key -l rsa2048-imported -w 524299

BER encoded key length is 1216

Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Private Key Unwrapped.  Key Handle: 524301

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-l

ユーザー定義のプライベートキーラベルを指定します。

必須: はい

-f

インポートするキーのファイル名を指定します。

必須: はい

-w

ラップキーのキーハンドルを指定します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

キーをラップキーとして使用できるかどうかを確認するには、getAttribute を使用して OBJ_ATTR_WRAP 属性 (262) の値を取得します。ラップキーを作成するには、genSymKey を使用して AES キー (タイプ 31) を作成します。

-wk パラメータを使用して外部のラップ解除キーを指定した場合、インポート時の (ラップ解除ではなく) ラップには -w ラップキーが使われます。

必須: はい

-sess

インポートされたキーをセッションキーに指定します。

デフォルト: インポートされたキーは、クラスター内で永続 (トークン) キーとして保持されます。

必須: いいえ

-id

インポートするキーの ID を指定します。

デフォルト: ID 値なし。

必須: いいえ

-m_value

インポートされたキーを使用した暗号化オペレーションを、何人のユーザーが承認しなければならないかを指定します。**0** ~ **8** の値を入力します。

このパラメータが有効なのは、コマンドの -u パラメータが m_value の要件を満たすために十分な数のユーザーとキーを共有するときのみです。

デフォルト: 0

必須: いいえ

-min_srv

-timeout パラメータの値が期限切れになる前に、インポートされたキーが最小いくつの HSM で同期されるかを指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、min_srv の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-timout

min-serv パラメータが含まれている場合に、すべての HSM でキーが同期されるまで待機する秒数を指定します。数値が指定されていない場合、ポーリングが永遠に続きます。

デフォルト: 無制限

必須: いいえ

-u

インポートされたプライベートキーを共有するユーザーのリストを指定します。このパラメータは、他の HSM Crypto User (CU) に対し、インポートされたキーを暗号化オペレーションに使用するアクセス許可を付与します。

HSM ユーザー ID のカンマ区切りリスト (例: --u 5,6) を入力します。現在のユーザーの HSM ユーザー ID を含めないでください。HSM で CU の HSM ユーザー ID を検索するには、listUsers を使用します。

デフォルト: 現在のユーザーのみがインポートされたキーを使用できます。

必須: いいえ

-wk

インポートするキーのラップ解除に使用するキーを指定します。プレーンテキストの AES キーが含まれているファイルのパスと名前を入力します。

このパラメータを含めた場合、importPrivateKey は、インポートするキーのラップに -wk ファイルのキーを使用します。また、ラップ解除には -w パラメータで指定されたキーを使用します。

デフォルト: -w パラメータで指定されたラップキーを使用して、ラップとラップ解除の両方を行う。

必須: いいえ

-attest

ファームウェアレスポンスの証明チェックを実行し、クラスターを実行するファームウェアが侵害されていないことを確認します。

必須: いいえ

関連トピック

- wrapKey
- unWrapKey
- genSymKey
- exportPrivateKey

importPubKey

key_mgmt_util の importPubKey コマンドは、PEM 形式の公開キーを HSM にインポートします。このコマンドを使用すると、HSM の外部で生成されたパブリックキーをインポートできます。コマンドを使用して、 コマンドによってエクスポートされたキーなど、HSM からエクスポートされたキーをインポートすることもできますexportPubKey。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
importPubKey -h
```

```
importPubKey -l <label>
             -f <key-file>
             [-sess]
             [-id <key-id>]
             [min_srv <minimum-number-of-servers>]
             [-timeout <number-of-seconds>]
```

例

この例では、importPubKey を使ってパブリックキーを HSM にインポートする方法を示します。

Example : パブリックキーをインポートする

このコマンドは、importedPublicKey というラベルを使って public.pem という名前のファイルからパブリックキーをインポートします。importPubKey コマンドは、成功すると、インポートされたキーのキーハンドルと成功メッセージを返します。

```
Command: importPubKey -l importedPublicKey -f public.pem

Cfm3CreatePublicKey returned: 0x00 : HSM Return: SUCCESS

Public Key Handle: 262230

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-l

ユーザー定義のパブリックキーラベルを指定します。

必須: はい

-f

インポートするキーのファイル名を指定します。

必須: はい

-sess

インポートされたキーをセッションキーに指定します。

デフォルト: インポートされたキーは、クラスター内で永続 (トークン) キーとして保持されます。

必須: いいえ

-id

インポートするキーの ID を指定します。

デフォルト: ID 値なし。

必須: いいえ

-min_srv

`-timeout` パラメータの値が期限切れになる前に、インポートされたキーが最小いくつの HSM に同期されるかを指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-timeout

`min-serv` パラメータが含まれている場合に、すべての HSM でキーが同期されるまで待機する秒数を指定します。数値が指定されていない場合、ポーリングが永遠に続きます。

デフォルト: 無制限

必須: いいえ

関連トピック

- exportPubKey
- キーの生成

imSymKey

key_mgmt_util ツールの imSymKey コマンドは、対称キーのプレーンテキストコピーをファイルから HSM にインポートします。これを使用して、HSM 以外の任意の方法で生成したキーと、コマンドがファイルに書き込むキーなどexSymKey、HSM からエクスポートされたキーをインポートできます。

インポートプロセス中に、imSymKey は選択した AES キー (ラッピングキー) を使用して、インポートするキーをラップ (暗号化) してからアンラップ (復号化) します。ただし、imSymKey を使用できるのは、プレーンテキストのキーが含まれているファイルに対してのみです。暗号化されたキーをエクスポートおよびインポートするには、 wrapKey コマンドと unWrapKey コマンドを使用します。

また、imSymKey コマンドは対称キーのみをインポートします。パブリックキーをインポートするには、importPubKey を使用します。プライベートキーをインポートするには、 importPrivateKeyまたは wrapKeyを使用します。

> **Note**
>
> パスワードで保護された PEM キーは、対称キーまたはプライベートキーを使用してインポートすることはできません。

インポートしたキーは、HSM で生成したキーとほぼ同じように動作します。ただし、OBJ_ATTR_LOCAL 属性の値は 0 であり、ローカルに生成されたものでないことを示しています。次のコマンドを使用して、インポートした対称キーを共有します。shareKeycloudhsm_mgmt_util で コマンドを使用して、インポート後にキーを共有します。

```
imSymKey -l aesShared -t 31 -f kms.key -w 3296 -u 5
```

キーのインポート後に、必ずキーファイルをマークまたは削除してください。このコマンドでは、同じキーマテリアルを複数回インポートすることが禁止されません。その結果、キーハンドルが異なる複数のキーが同じキーマテリアルを持つ場合があり、キーマテリアルの使用の追跡が困難になります。また、暗号化の制限に制約されます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
imSymKey -h

imSymKey -f <key-file>
         -w <wrapping-key-handle>
         -t <key-type>
         -l <label>
         [-id <key-ID>]
         [-sess]
         [-wk <wrapping-key-file> ]
         [-attest]
         [-min_srv <minimum-number-of-servers>]
         [-timeout <number-of-seconds> ]
         [-u <user-ids>]
```

例

以下の例では、imSymKey を使用して対称キーを HSM 内にインポートする方法を示します。

Example : AES 対称キーをインポートする

次の例では、imSymKey を使用して AES 対称キーを HSM 内にインポートします。

最初のコマンドでは、OpenSSL を使用してランダムな 256 ビット AES 対称キーを生成します。生成したキーは、aes256.key ファイルに保存されます。

```
$  openssl rand -out aes256-forImport.key 32
```

2 番目のコマンドでは、imSymKey を使用し、AES キーを aes256.key ファイルから HSM 内にインポートします。HSM の AES キー (キー 20) をラップキーとして使用し、imported をラベルとし

て指定します。ID とは異なり、ラベルはクラスター内で一意である必要はありません。-t (タイプ) パラメータの値は、AES を表す 31 です。

出力は、ファイルのキーがラップ/ラップ解除され、次に HSM 内にインポートされて、キーハンドル 262180 が割り当てられたことを示しています。

```
Command:  imSymKey -f aes256.key -w 20 -t 31 -l imported

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 262180

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、getAttribute を使用して、新しくインポートしたキーの OBJ_ATTR_LOCAL 属性 (属性 355) を取得し、それを attr_262180 ファイルに書き込みます。

```
Command:  getAttribute -o 262180 -a 355 -out attributes/attr_262180
Attributes dumped into attributes/attr_262180_imported file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

属性ファイルを調べると、OBJ_ATTR_LOCAL 属性の値は 0 であり、キーマテリアルが HSM で生成されたものでないことがわかります。

```
$  cat attributes/attr_262180_local
OBJ_ATTR_LOCAL
0x00000000
```

Example : クラスター間で対称キーを移動する

この例では、 exSymKeyと を使用してクラスター間でプレーンテキストの AES キーimSymKeyを移動する方法を示します。次のようなプロセスを使用して HSM の両クラスターで有効な AES ラッ

ピングを作成できます。共有ラッピングキーを配置したら、 wrapKey と を使用して暗号化されたキーunWrapKeyをクラスター間で移動できます。

このオペレーションを実行する CU ユーザーには、両クラスターで HSM にログインするアクセス許可が必要です。

最初のコマンドではexSymKey、 を使用して、32 ビット AES キーであるキー 14 をクラスター 1 から aes.key ファイルにエクスポートします。ラップキーとして、クラスター 1 の HSM でキー 6 (AES キー) を使用します。

```
Command: exSymKey -k 14 -w 6 -out aes.key

        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "aes.key"
```

次に、ユーザーはクラスター 2 の key_mgmt_util にログインし、imSymKey コマンドを実行して aes.key ファイルのキーをクラスター 2 の HSM 内にインポートします。このコマンドでは、ラップキーとして、クラスター 2 の HSM で キー 252152 (AES キー) を使用します。

exSymKey と がimSymKey使用するラッピングキーはターゲットキーをラップしてすぐにラップ解除するため、異なるクラスターのラッピングキーが同じである必要はありません。

出力は、キーがクラスター 2 に正常にインポートされてキーハンドル 21 が割り当てられたことを示しています。

```
Command:  imSymKey -f aes.key -w 262152 -t 31 -l xcluster

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 21

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

```
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

クラスター 1 のキー 14 とクラスター 2 のキー 21 で、キーマテリアルが同じであることを確認するには、各キーのキーチェック値 (KCV) を取得します。KCV 値が同じであれば、キーマテリアルは同じです。

次のコマンドでは、クラスター 1 の getAttribute を使用してキー 14 の KCV 属性 (属性 371) の値を attr_14_kcv ファイルに書き込みます。次に、cat コマンドを使用して、attr_14_kcv ファイルの内容を取得します。

```
Command:  getAttribute -o 14 -a 371 -out attr_14_kcv
Attributes dumped into attr_14_kcv file

$  cat attr_14_kcv
OBJ_ATTR_KCV
0xc33cbd
```

次の同様のコマンドでは、クラスター 2 の getAttribute を使用してキー 21 の KCV 属性 (属性 371) の値を attr_21_kcv ファイルに書き込みます。次に、cat コマンドを使用して、attr_21_kcv ファイルの内容を取得します。

```
Command:  getAttribute -o 21 -a 371 -out attr_21_kcv
Attributes dumped into attr_21_kcv file

$  cat attr_21_kcv
OBJ_ATTR_KCV
0xc33cbd
```

出力は、2 つのキーの KCV 値が同じであり、キーマテリアルが同じであることを示しています。

両クラスターの HSM でキーマテリアルが同じであるため、プレーンテキストキーを公開することなく、クラスター間で暗号化されたキーを共有できます。たとえば、wrapKey コマンドでラップキー 14 を使用してクラスター 1 から暗号化されたキーをエクスポートし、次に unWrapKey でラップキー 21 を使用してクラスター 2 に暗号化されたキーをインポートできます。

Example : セッションキーをインポートする

次のコマンドでは、-sess の imSymKey パラメータを使用し、現在のセッションでのみ有効な 192 ビット Triple DES キーをインポートします。

このコマンドでは、インポートするキーが含まれているファイルを -f パラメータで指定します。また、キーのタイプを -t パラメータで指定し、ラップキーを -w パラメータで指定します。キーを分類するラベルを -l パラメータで指定し、キーのフレンドリーな一意の識別子を -id パラメータで作成します。さらに、キーをインポートするファームウェアを -attest パラメータで検証します。

出力は、キーが正常にラップ/ラップ解除され、HSM 内にインポートされて、キーハンドル 37 が割り当てられたことを示しています。また、認証チェックに合格し、ファームウェアが改ざんされていないことを示しています。

```
Command:  imSymKey -f 3des192.key -w 6 -t 21 -l temp -id test01 -sess -attest

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 37

        Attestation Check : [PASS]

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

次に、getAttribute コマンドまたは findKey コマンドを使用し、新しくインポートされたキーの属性を検証できます。次のコマンドでは、findKey を使用して、キー 37 のタイプ、ラベル、および ID がコマンドで指定されたとおりであること、さらにセッションキーであることを検証します。出力の 5 行目が示すように、findKey は、すべての属性に一致するキーがキー 37 のみであることを示しています。

```
Command:  findKey -t 21 -l temp -id test01 -sess 1
Total number of keys present 1

 number of keys matched from start index 0::0
37

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

```
        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須: いいえ

-f

インポートするキーが含まれているファイルを指定します。

ファイルには、指定された長さの AES キーまたは Triple DES キーのプレーンテキストコピーが含まれている必要があります。RC4 キーと DES キーは FIPS モードの HSM では無効です。

- AES: 16、24、または 32 バイト
- Triple DES (3 DES): 24 バイト

必須: はい

-h

コマンドに関するヘルプを表示します。

必須: はい

-id

キーのユーザー定義識別子を指定します。クラスター内で一意の文字列を入力します。デフォルトは空の文字列です。

デフォルト: ID 値なし。

必須: いいえ

-l

キーのユーザー定義ラベルを指定します。文字列を入力します。

キーを識別するのに役立つ任意のフレーズを使用できます。ラベルは一意である必要がないため、このラベルを使用してキーをグループ化および分類できます。

必須: はい

-min_srv

`-timeout` パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、`min_srv` の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須: いいえ

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須: いいえ

-timeout

キーが `min_srv` パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、`min_srv` パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須: いいえ

-t

対称キーのタイプを指定します。キーのタイプを表す定数を入力します。たとえば、AES キーを作成するには「`-t 31`」と入力します。

有効値:

- 21: Triple DES (3DES)。
- 31: AES

必須: はい

-u

指定したユーザーとインポートするキーを共有します。このパラメータは、別の HSM Crypto User (CU) に、暗号化オペレーションでこのキーを使用するアクセス許可を付与します。

1 つの ID または HSM ユーザー ID のカンマ区切りリスト (`5,6` など) を入力します。現在のユーザーの HSM ユーザー ID を含めないでください。ID を確認するには、cloudhsm_mgmt_util コマンドラインツールの listUsers コマンドまたは key_mgmt_util コマンドラインツールの listUsers コマンドを使用できます。

必須: いいえ

-w

ラップキーのキーハンドルを指定します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

ラップキーは、インポートプロセスでキーの暗号化 (ラップ) と復号 (ラップ解除) に使用する HSM のキーです。ラップキーとして使用できるのは AES キーのみです。

任意の AES キー (任意のサイズ) をラップキーとして使用できます。ラップキーは、ターゲットキーをラップし、直後にラップ解除するため、セッション専用の AES キーをラップキーとして使用できます。キーをラップキーとして使用できるかどうかを確認するには、getAttribute を使用して `OBJ_ATTR_WRAP` 属性 (262) の値を取得します。ラッピングキーを作成するには、genSymKeyを使用して AES キー (タイプ 31) を作成します。

`-wk` パラメータを使用して外部のラップキーを指定した場合、インポートしたキーは `-w` ラップキーでラップ解除することはできますが、ラップすることはできません。

> **Note**
>
> キー 4 は、サポートされていない内部キーです。AES キーをラップキーとして作成および管理することをお勧めします。

必須: はい

-wk

指定されたファイルの AES キーを使用して、インポートするキーをラップします。プレーンテキストの AES キーが含まれているファイルのパスと名前を入力します。

このパラメータを含めると、imSymKey は、-wk ファイルのキーを使用してインポートされたキーをラップし、-w パラメータで指定された HSM のキーを使用してラップ解除します。-w パラメータと -wk パラメータの値は同じプレーンテキストのキーに解決される必要があります。

デフォルト: HSM のラップキーを使用してラップ解除します。

必須: いいえ

関連トピック

- genSymKey
- exSymKey
- wrapKey
- unWrapKey
- exportPrivateKey
- exportPubKey

insertMaskedObject

key_mgmt_util の insertMaskedObject コマンドは、隠されたオブジェクトをファイルから指定された HSM に挿入します。マスクされたオブジェクトとは、クローンされたオブジェクトで、extractMaskedObject コマンドを使用して HSM から抽出されたものです。マスクされたオブジェクトは、再び元のクラスターに挿入して初めて使用可能になります。マスクされたオブジェクトは、生成元であるクラスター、またはそのクラスターのクローンにしか挿入できません。これには、リージョン間でのバックアップのコピーによって生成されたクラスターのクローンバージョン

や、そのバックアップを使って新しいクラスターを作成することで生成された元のクラスターのクローンバージョンが含まれます。

マスクされたオブジェクトは、抽出不可能なキー (OBJ_ATTR_EXTRACTABLE 値が 0 であるキー) を含め、キーを効率的にオフロードおよび同期する手段です。これにより、 AWS CloudHSM 設定ファイル を更新しなくても、異なるリージョンの関連クラスター間でキーを安全に同期できます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
insertMaskedObject -h

insertMaskedObject -f <filename>
                   [-min_srv <minimum-number-of-servers>]
                   [-timeout <number-of-seconds>]
```

例

この例では、insertMaskedObject を使ってマスクされたオブジェクトを HSM に挿入する方法を示します。

Example : マスクされたオブジェクトを挿入する

このコマンドは、maskedObj というファイルにあるマスクされたオブジェクトを HSM に挿入します。insertMaskedObject コマンドは、成功すると、マスクされたオブジェクトから複合されたキーのキーハンドルと成功メッセージを返します。

```
Command: insertMaskedObject -f maskedObj

Cfm3InsertMaskedObject returned: 0x00 : HSM Return: SUCCESS
        New Key Handle: 262433

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-f

マスクされたオブジェクトの挿入先とするファイル名を指定します。

必須: はい

-min_srv

`-timeout` パラメータの値が期限切れになる前に、挿入したマスクされたオブジェクトが最小いくつの HSM で同期されるかを指定します。オブジェクトが割り当てられた時間内に指定の数のサーバーに同期されなかった場合、そのオブジェクトは挿入されません。

デフォルト: 1

必須: いいえ

-timeout

`min-serv` パラメータが含まれている場合に、すべてのサーバーでキーが同期されるまで待機する秒数を指定します。数値が指定されていない場合、ポーリングが永遠に続きます。

デフォルト: 無制限

必須: いいえ

関連トピック

- extractMaskedObject
- syncKey
- リージョン間のバックアップのコピー
- 以前のバックアップからの AWS CloudHSM クラスターの作成

IsValidKeyHandlefile

key_mgmt_util の IsValidKeyHandlefile コマンドは、キーファイルに実際のプライベートキーとフェイク RSA PEM キーのどちらが含まれているかを確認するために使用されます。フェイク PEM ファ

イルは、実際のプライベートキーマテリアルを含まず、HSM のプライベートキーを参照します。このようなファイルは、ウェブサーバーから AWS CloudHSMへの SSL/TLS オフロードを確立するために使います。詳細については、「Linux 上の SSL/TLS オフロード」を参照してください。

> **Note**
>
> IsValidKeyHandlefile は RSA キーにのみ機能します。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
IsValidKeyHandlefile -h

IsValidKeyHandlefile -f <rsa-private-key-file>
```

例

以下の例では、IsValidKeyHandlefile を使って、あるキーファイルに含まれるのが実際のキーマテリアルなのか、フェイク PEM キーマテリアルなのかを特定する方法を示します。

Example : 実際のプライベートキーを検証する

次のコマンドは、privateKey.pem というファイルに実際のキーマテリアルが含まれていることを確認します。

```
Command: IsValidKeyHandlefile -f privateKey.pem

Input key file has real private key
```

Example : フェイク PEM キーを無効化する

次のコマンドは、caviumKey.pem というファイルにキーハンドル 15 から生成されたフェイク PEM キーが含まれていることを確認します。

```
Command: IsValidKeyHandlefile -f caviumKey.pem
```

```
Input file has invalid key handle: 15
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-h

コマンドのコマンドラインヘルプを表示します

必須: はい

-f

有効なキーマテリアルの存在を確認する RSA プライベートキーファイルを指定します。

必須: はい

関連トピック

- getCaviumPrivキー
- Linux 上の SSL/TLS オフロード

listAttributes

key_mgmt_util の listAttributes コマンドは、 AWS CloudHSM キーの属性とそれを表す定数を一覧表示します。これらの定数は、getAttribute コマンドおよび setAttribute コマンドの属性を特定するのに使用します。キー属性の解釈については、キー属性リファレンス を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

このコマンドにはパラメータはありません。

```
listAttributes
```

例

このコマンドは、key_mgmt_utilで取得および変更できるキー属性と、それらを表す定数を一覧表示します。キー属性の解釈については、キー属性リファレンス を参照してください。

key_mgmt_util の getAttribute コマンドですべての属性を表すには、512 を使用します。

```
Command: listAttributes

Following are the possible attribute values for getAttributes:

      OBJ_ATTR_CLASS                = 0
      OBJ_ATTR_TOKEN                = 1
      OBJ_ATTR_PRIVATE              = 2
      OBJ_ATTR_LABEL                = 3
      OBJ_ATTR_KEY_TYPE             = 256
      OBJ_ATTR_ENCRYPT              = 260
      OBJ_ATTR_DECRYPT              = 261
      OBJ_ATTR_WRAP                 = 262
      OBJ_ATTR_UNWRAP               = 263
      OBJ_ATTR_SIGN                 = 264
      OBJ_ATTR_VERIFY               = 266
      OBJ_ATTR_LOCAL                = 355
      OBJ_ATTR_MODULUS              = 288
      OBJ_ATTR_MODULUS_BITS         = 289
      OBJ_ATTR_PUBLIC_EXPONENT      = 290
      OBJ_ATTR_VALUE_LEN            = 353
      OBJ_ATTR_EXTRACTABLE          = 354
      OBJ_ATTR_KCV                  = 371
```

関連トピック

- cloudhsm_mgmt_util の setAttribute
- getAttribute
- setAttribute
- キー属性リファレンス

listUsers

key_mgmt_utilの listUsers コマンド は、HSM のユーザーを、ユーザータイプおよびその他の属性とともに取得します。

key_mgmt_util で、listUsers は、一貫性がない場合でも、クラスターのすべての HSM を表す出力を返します。各 HSM のユーザーに関する情報を取得するには、cloudhsm_mgmt_util の listUsers コマンドを使用します。

key_mgmt_util listUsersおよび のユーザーコマンドは、getKeyInfoCrypto User (CUsが実行するアクセス許可を持つ読み取り専用コマンドです。残りのユーザー管理コマンドは cloudhsm_mgmt_util の一部です。それらは、ユーザー管理アクセス権限を持つ Crypto Officer (CO) によって実行されます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
listUsers

listUsers -h
```

例

このコマンドは、クラスターの HSM のユーザーとその属性を一覧表示します。User ID 属性を使用して、findKey、 などの他のコマンドでユーザーを識別できますgetKeyInfo。 getAttribute

```
Command:  listUsers

        Number Of Users found 4

        Index       User ID     User Type       User Name           MofnPubKey
 LoginFailureCnt         2FA
        1               1      PCO             admin                    NO
    0               NO
        2               2      AU              app_user                 NO
    0               NO
        3               3      CU              alice                    YES
    0               NO
        4               4      CU              bob                      NO
    0               NO
        5               5      CU              trent                    YES
    0               NO

        Cfm3ListUsers returned: 0x00 : HSM Return: SUCCESS
```

この出力が示すユーザー属性は以下のとおりです。

- ユーザー ID: key_mgmt_util および cloudhsm_mgmt_util のコマンドでユーザーを識別します。

- User type: HSM でユーザーが実行できるオペレーションを決定します。
- User Name: ユーザー定義のわかりやすいユーザー名を表示します。
- MofnPubKey: ユーザーがクォーラム認証トークン に署名するためのキーペアを登録したかどうかを示します。
- LoginFailureCnt: ユーザーがログインに失敗した回数を示します。
- 2FA: ユーザーが多要素認証を有効にしていることを示します。

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

関連トピック

- cloudhsm_mgmt_util の listUsers
- findKey
- getAttribute
- getKeyInfo

loginHSM および logoutHSM

key_mgmt_util の loginHSM と logoutHSM コマンドを使用すると、クラスターの HSM にログインおよびログアウトできます。HSM にログインすると、key_mgmt_util を使用して、公開キーとプライベートキーの生成、同期、ラッピングなど、さまざまなキー管理オペレーションを実行できます。

どの key_mgmt_util コマンドを実行する場合でも、事前に key_mgmt_util を起動する必要があります。key_mgmt_util を使用してキーを管理するには、暗号化ユーザー (CU) として HSM にログインする必要があります。

> **Note**
>
> ログイン試行回数が 5 回を超えると、アカウントがロックアウトされます。2018 年 2 月より前にクラスターを作成した場合、ロックアウトされるまでのログイン試行回数は 20 回で

す。アカウントのロックを解除するには、暗号化オフィサー (CO) が cloudhsm_mgmt_util で changePswd コマンドを使用してパスワードをリセットする必要があります。
クラスター内に複数の HSM がある場合は、アカウントがロックアウトされるまでのログイン試行回数の上限が増える可能性があります。これは、CloudHSM クライアントがさまざまな HSM 間で負荷を分散するためです。したがって、ログイン試行は毎回同じ HSM で開始されない場合があります。この機能をテストしている場合は、アクティブな HSM が1つだけのクラスターでテストすることをお勧めします。

Syntax

```
loginHSM -h

loginHSM -u <user type>
         { -p | -hpswd } <password>
         -s <username>
```

例

この例では、loginHSM および logoutHSM コマンドを使ってクラスターの HSM でログインおよびログアウトする方法を示します。

Example : HSM にログインする

このコマンドは、CU というユーザー名と example_user というパスワードを使い、暗号ユーザー (aws) として HSM にログインします。出力には、クラスターのすべてのHSMにログインしたことが示されます。

```
Command:  loginHSM -u CU -s example_user -p aws

Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : 隠しパスワードでログインします。

このコマンドは上記の例と同じですが、今回はシステムがパスワードを隠すように指定することを除きます。

```
Command:  loginHSM -u CU -s example_user -hpswd
```

システムからパスワードの入力を求められます。パスワードを入力すると、システムはパスワードを非表示にし、コマンドが正常に実行されたことと HSM に接続したことが出力 で示されます。

```
Enter password:

Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

Command:
```

Example : HSM からログアウトする

このコマンドは HSM からログアウトします。出力は、クラスターのすべての HSM からログアウトしたことを示しています。

```
Command: logoutHSM

Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

-h

このコマンドに関するヘルプを表示します。

-u

ログインユーザーのタイプを指定します。key_mgmt_util を使用するには、CU としてログインする必要があります。

必須: はい

-s

ログインユーザー名を指定します。

必須: はい

{-p |-hpswd}

-p でログインパスワードを指定します。パスワードは、入力するとプレーンテキストで表示されます。パスワードを非表示にするには、-hpswd の代わりにオプションの -p パラメータを使用して、プロンプトに従います。

必須: はい

関連トピック

- exit

setAttribute

key_mgmt_util の setAttribute コマンドは、現在のセッションでのみ有効なキーを、削除するまで存在する永続キーに変換します。この変換を行うために、キーのトークン属性 (OBJ_ATTR_TOKEN) の値を false (0) から true (1) に変更します。自分が所有するキーの属性のみ変更できます。

cloudhsm_mgmt_util の setAttribute コマンドを使用して、ラベルの変更、属性のラップ、アンラップ、暗号化、および復号化を行うこともできます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
setAttribute -h

setAttribute -o <object handle>
             -a 1
```

例

次の例では、セッションキーを永続キーに変換する方法を示します。

最初のコマンドは、 の -sessパラメータgenSymKeyを使用して、現在のセッションでのみ有効な 192 ビット AES キーを作成します。出力は、新しいセッションキーのキーハンドルが 262154 であることを示しています。

```
Command: genSymKey -t 31 -s 24 -l tmpAES -sess

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262154

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、findKey を使用して現在のセッションのセッションキーを確認します。出力は、キー 262154 がセッションキーであることを示しています。

```
Command:  findKey -sess 1

Total number of keys present 1

 number of keys matched from start index 0::0
262154

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

次のコマンドでは、setAttribute を使用してキー 262154 をセッションキーから永続キーに変換します。そのために、キーのトークン属性 (OBJ_ATTR_TOKEN) の値を 0 (false)から 1 (true)に変更します。キー属性の解釈については、キー属性リファレンス を参照してください。

このコマンドでは、-o パラメータを使用してキーハンドル (262154) を指定し、-a パラメータを使用してトークン属性を表す定数 (1) を指定します。コマンドを実行すると、トークン属性の値を指定するよう求められます。唯一の有効な値は 1 (true) です。これは、永続キーの値です。

```
Command: setAttribute -o 262154 -a 1
         This attribute is defined as a boolean value.
          Enter the boolean attribute value (0 or 1):1
```

```
        Cfm3SetAttribute returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

次のコマンドでは、262154 が永続キーになったことを確認するために、findKey を使用してセッションキー (-sess 1) と永続キー (-sess 0) を検索します。今回は、コマンドでセッションキーが検出されず、永続キーのリストで 262154 が返されます。

```
Command: findKey -sess 1

Total number of keys present 0

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS



Command: findKey -sess 0

Total number of keys present 5

 number of keys matched from start index 0::4
6, 7, 524296, 9, 262154

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須: はい

-オ

ターゲットキーのキーハンドルを指定します。各コマンドに指定できるキーは 1 つのみです。キーのキーハンドルを取得するには、findKey を使用します。

必須: はい

-a

変更する属性を表す定数を指定します。唯一の有効な値は 1 です。これはトークン属性 OBJ_ATTR_TOKEN を表します。

属性とその整数値を取得するには、listAttributes を使用します。

必須: はい

関連トピック

- cloudhsm_mgmt_util の中で setAttribute です。
- getAttribute
- listAttributes
- キー属性リファレンス

sign

sign key_mgmt_util のコマンドは、選択したプライベートキーを使用してファイルの署名を生成します。

sign を使用するには、HSM 内にプライベートキーを持っておく必要があります。プライベートキーは、genSymKey コマンド、genRSAKeyPair コマンド、または genECCKeyPair コマンドで生成できます。また、importPrivateKey コマンドを使用してインポートすることもできます。詳細については、「キーの生成」を参照してください。

sign コマンドは、ユーザーが指定した (整数で表した) 署名機構を使って、メッセージファイルに署名します。使用できる署名機構のリストについては、「パラメータ」を参照してください。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
sign -h

sign -f <file name>
     -k <private key handle>
     -m <signature mechanism>
     -out <signed file name>
```

例

この例では、sign を使ってファイルに署名する方法を示します。

Example : ファイルに署名する

次のコマンドは、ハンドル 266309 を持つプライベートキーを使って messageFile という名前のファイルに署名します。このコマンドは、SHA256_RSA_PKCS (1) 署名機構を使い、署名したファイルを signedFile として保存します。

```
Command: sign -f messageFile -k 266309 -m 1 -out signedFile

Cfm3Sign returned: 0x00 : HSM Return: SUCCESS

signature is written to file signedFile

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-f

署名するファイルの名前。

必須: はい

-k

署名に使用するプライベートキーのハンドル。

必須: はい

-m

署名に使用する署名機構を表す整数。使用可能な署名機構は、次のような整数に対応します。

署名機構	対応する整数
SHA1_RSA_PKCS	0
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17
ECDSA_SHA384	18
ECDSA_SHA512	19

必須: はい

-out

署名したファイルの保存先とするファイルの名前。

必須: はい

関連トピック

- verify
- importPrivateKey
- genRSAKeyPair
- genECCKeyPair
- genSymKey
- キーの生成

unWrapKey

key_mgmt_util ツールの unWrapKey コマンドは、ラップされた (暗号化された) 対称キーまたはプライベートキーをファイルから HSM にインポートします。key_mgmt_util の wrapKey コマンドでラップされた暗号化されたキーをインポートするように設計されていますが、他のツールでラップされたキーをアンラップするためにも使用できます。ただし、このような場合は、PKCS#11 または JCE ソフトウェアライブラリを使用して、キーをラップ解除することをお勧めします。

インポートされたキーは、 によって生成されたキーのように機能します AWS CloudHSM。ただし、OBJ_ATTR_LOCAL 属性の値は 0 であり、ローカルに生成されたものでないことを示しています。

キーをインポートしたら、必ずキーファイルをマークまたは削除してください。このコマンドでは、同じキーマテリアルを複数回インポートすることが禁止されません。その結果、異なるキーハンドルと同じキーマテリアルを持つ複数のキーにより、キーマテリアルの使用を追跡し、暗号化の制限を超えないようにすることが困難になります。

key_mgmt_util コマンドを実行する前に、 key_mgmt_util を起動し、Crypto User (CU) として HSM に ログインする 必要があります。

Syntax

```
unWrapKey -h

unWrapKey -f <key-file-name>
          -w <wrapping-key-handle>
```

```
        [-sess]
        [-min_srv <minimum-number-of-HSMs>]
        [-timeout <number-of-seconds>]
        [-aad <additional authenticated data filename>]
        [-tag_size <tag size>]
        [-iv_file <IV file>]
        [-attest]
        [-m <wrapping-mechanism>]
        [-t <hash-type>]
        [-nex]
        [-u <user id list>]
        [-m_value <number of users needed for approval>]
        [-noheader]
        [-l <key-label>]
        [-id <key-id>]
        [-kt <key-type>]
        [-kc <key-class]
        [-i <unwrapping-IV>]
```

例

これらの例では、unWrapKey を使用して、ラップされたキーをファイルから HSM にインポートする方法を示します。最初の例では、wrapKey key_mgmt_util コマンドでラップされたキーをアンラップしているため、ヘッダーがあります。2 番目の例では、key_mgmt_util の外部でラップされたため、ヘッダーがないキーをアンラップします。

Example : キーのラップ解除 (ヘッダー付き)

このコマンドでは、3DES 対称キーのラップされたコピーを HSM にインポートします。キーはラベル 6 が付いた AES キーでラップ解除されます。このキーは、3 DES キーのラップに使用されたキーと暗号的に同一です。この出力は、ファイルのキーがラップ解除されてインポートされたことと、インポートされたキーのハンドルが 29 であることを示しています。

```
        Command:  unWrapKey -f 3DES.key -w 6 -m 4

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Key Unwrapped.  Key Handle: 29

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

```
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : キーのラップ解除 (ヘッダーなし)

このコマンドでは、3DES 対称キーのラップされたコピーを HSM にインポートします。キーはラベル 6 が付いた AES キーでラップ解除されます。このキーは、3 DES キーのラップに使用されたキーと暗号的に同一です。この 3DES キーは key_mgmt_util でラップされていないため、noheader パラメータは、キー・ラベル (unwrapped3DES)、キー・クラス (4)、キー・タイプ (21) など必要な付随パラメータとともに指定されます。この出力は、ファイルのキーがラップ解除されてインポートされたことと、インポートされたキーのハンドルが 8 であることを示しています。

```
Command: unWrapKey -f 3DES.key -w 6 -noheader -l unwrapped3DES -kc 4 -kt 21 -m 4

      Cfm3CreateUnwrapTemplate2 returned: 0x00 : HSM Return: SUCCESS
       Cfm2UnWrapWithTemplate3 returned: 0x00 : HSM Return: SUCCESS

       Key Unwrapped. Key Handle: 8

       Cluster Error Status
       Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
       Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須：はい

-f

ラップされたキーが含まれているファイルのパスと名前を指定します。

必須：はい

-w

ラップキーを指定します。HSM の AES キーまたは RSA キーのキーハンドルを入力します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

ラッピングキーを作成するには、 genSymKeyを使用して AES キー (タイプ 31) を生成し、genRSAKeyPair を使用して RSA キーペア (タイプ 0) を生成します。RSA キーペアを使用

している場合は、必ず一方のキーでキーをラップし、もう一方のキーでアンラップしてください。キーをラッピングキーとして使用できることを確認するには、getAttribute を使用して、定数 OBJ_ATTR_WRAP で表される 262 属性の値を取得します。

必須：はい

-sess

現在のセッションにのみ存在するキーを作成します。セッション終了後、キーをリカバリすることはできません。

このパラメータは、別のキーを暗号化してからすばやく復号化するラッピングキーなど、キーが短時間だけ必要な場合に使用します。セッション終了後に復号する必要がある可能性のあるデータを暗号化するためにセッションキーを使用しないでください。

セッションキーを永続(トークン)キーに変更するには、setAttribute を使用します。

デフォルト: キーは永続的です。

必須：いいえ

-min_srv

-timeout パラメーターの値が期限切れになる前に、キーが同期される HSM の最小数を指定します。キーが割り当てられた時間内に指定された数のサーバーに同期されない場合は、作成されません。

AWS CloudHSM は、すべてのキーをクラスター内のすべての HSM に自動的に同期します。プロセスを高速化するため、min_srv の値をクラスターの HSM の数より少なく設定し、低いタイムアウト値を設定します。ただし、一部のリクエストでキーが生成されない場合があることに注意してください。

デフォルト: 1

必須：いいえ

-timeout

キーが min_srv パラメータで指定された HSM の数に同期されるのをコマンドが待機する時間 (秒単位) を指定します。

このパラメータは、min_srv パラメータがコマンドでも使用されている場合にのみ有効です。

デフォルト: タイムアウトなし。このコマンドは無期限に待機し、キーが最小数のサーバーと同期されている場合にのみ戻ります。

必須：いいえ

-attest

クラスターを実行するファームウェアが改ざんされていないことを確認する整合性チェックを実行します。

デフォルト: 認証チェックなし。

必須：いいえ

-nex

キーを抽出できなくなります。生成されたキーは HSM からエクスポートできません。

デフォルト: キーは抽出可能です。

必須：いいえ

-m

ラップメカニズムを表す値。CloudHSM は、次のメカニズムをサポートしています。

メカニズム	値
AES_KEY_WRAP_PAD_PKCS5	4
NIST_AES_WRAP_NO_PAD	5
NIST_AES_WRAP_PAD	6
RSA_AES	7
RSA_OAEP (最大データサイズについてはこのセクションの後半のメモを参照)	8
AES_GCM	10
CLOUDHSM_AES_GCM	11
RSA_PKCS (最大データサイズについてはこのセクションの後半のメモを参照)。今後の変更	12

メカニズム	値
については、以下の注記「1」を参照してください。	

必須：はい

> **Note**
>
> RSA_OAEP ラップメカニズムを使用する場合、ラップできる最大キーサイズは、RSA キーのモジュラスと指定されたハッシュの長さによって次のように決定されます。最大キーサイズ = modulusLengthInバイト (2*hashLengthInバイト)-2。
> RSA_PKCS ラップメカニズムを使用する場合、ラップできる最大キーサイズは、RSA キーのモジュラスによって次のように決定されます。最大キーサイズ = (modulusLengthInバイト -11)。

-t

ハッシュアルゴリズム	値
SHA1	2
SHA256	3
SHA384	4
SHA512	5
SHA224 (RSA_AES および RSA_OAEP メカニズムに対して有効)	6

必須：いいえ

-noheader

key_mgmt_util の外部でラップされたキーをアンラップする場合は、このパラメータと他のすべての関連パラメータを指定する必要があります。

必須：いいえ

> **Note**
>
> このパラメータを指定する場合は、-noheader パラメータも指定する 必要があります。

- -l

  ラップ解除されたキーに追加するラベルを指定します。

  必須：はい

- -kc

  ラップ解除するキーのクラスを指定します。使用できる値は以下のとおりです。

  3 = パブリックキーとプライベートキーのペアのプライベートキー

  4: シークレット (対称) キー

  必須：はい

- -kt

  ラップ解除するキーのタイプを指定します。使用できる値は以下のとおりです。

  0 = RSA

  1 = DSA

  3 = ECC

  16 = GENERIC_SECRET

  21 = DES3

  31 = AES

  必須：はい

オプションで次の -noheader パラメータを指定することもできます。

- -id

ラップ解除されたキーに追加する ID。

必須：いいえ

- -i

使用するラップ解除対象の初期ベクトル (IV)。

必須：いいえ

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

関連トピック

- wrapKey
- exSymKey
- imSymKey

認証

key_mgmt_util の verify コマンドは、ファイルが特定のキーによって署名されているかどうかを確認します。その際、verify コマンドは、署名されたファイルをソースファイルと比較し、両ファイルが特定のパブリックキーと署名機構に基づいて暗号的に関連するかどうかを分析します。ファイルは signオペレーション AWS CloudHSM でサインインできます。

署名機構は、「パラメータ」セクションにリストされている整数によって表されます。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
verify -h

verify -f <message-file>
       -s <signature-file>
       -k <public-key-handle>
```

```
        -m <signature-mechanism>
```

例

これらの例は、verify を使って、特定のファイルの署名に特定のパブリックキーが使用されたかどうかを確認する方法を示します。

Example : ファイル署名の認証

このコマンドは、hardwarCert.crt というファイルがパブリックキー 262276 と署名機構 SHA256_RSA_PKCS を使って署名され、hardwareCertSigned という署名ファイルが作成されたかどうかの検証を試みます。指定のパラメータが真の署名関係を表すため、コマンドは、成功メッセージを返します。

```
Command: verify -f hardwareCert.crt -s hardwareCertSigned -k 262276 -m 1

Signature verification successful

Cfm3Verify returned: 0x00 : HSM Return: SUCCESS
```

Example : 偽の署名関係を証明する

このコマンドは、hardwareCert.crt というファイルがパブリックキー 262276 と署名機構 SHA256_RSA_PKCS を使って署名され、userCertSigned という署名ファイルが作成されたことを検証します。指定のパラメータが真の署名関係を構成していないため、コマンドは、エラーメッセージを返します。

```
Command: verify -f hardwarecert.crt -s usercertsigned -k 262276 -m 1
Cfm3Verify returned: 0x1b

CSP Error: ERR_BAD_PKCS_DATA
```

パラメータ

このコマンドでは、以下のパラメータを使用します。

-f

元のメッセージファイルの名前。

必須: はい

-s

署名したファイルの名前。

必須: はい

-k

ファイルの署名に使用されたと考えられるパブリックキーのハンドル。

必須: はい

-m

ファイルの署名に使用するよう提案された署名機構を表す整数。使用可能な署名機構は、次のような整数に対応します。

署名機構	対応する整数
SHA1_RSA_PKCS	0
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17

署名機構	対応する整数
ECDSA_SHA384	18
ECDSA_SHA512	19

必須: はい

関連トピック

- sign
- getCert
- キーの生成

wrapKey

key_mgmt_util の wrapKey コマンドは、対称キーまたはプライベートキーの暗号化されたコピーを HSM からファイルにエクスポートします。wrapKey を実行するときに、エクスポートするキー、エクスポートするキーを暗号化 (ラップ) するための HSM 上のキー、出力ファイルを指定します。

wrapKey コマンドは、暗号化されたキーを指定したファイルに書き込みますが、キーを HSM から削除したり、暗号化オペレーションでのキーの使用を禁止したりすることはありません。同じキーを複数回エクスポートできます。

キーは、キーの所有者 (キーを作成した Crypto User (CU)) のみエクスポートできます。キーを共有するユーザーは、キーを暗号化オペレーションで使用することはできますが、エクスポートすることはできません。

暗号化されたキーを HSM にインポートし直すには、 を使用しますunWrapKey。HSM からプレーンテキストキーをエクスポートするには、exportPrivateKey必要に応じて exSymKeyまたは を使用します。aesWrapUnwrap コマンドは、暗号化するキーを復号 (アンラップ) wrapKey できません。

key_mgmt_util コマンドを実行する前に、key_mgmt_util を起動し、Crypto User (CU) として HSM にログインする 必要があります。

Syntax

```
wrapKey -h
```

```
wrapKey -k <exported-key-handle>
        -w <wrapping-key-handle>
        -out <output-file>
        [-m <wrapping-mechanism>]
        [-aad <additional authenticated data filename>]
        [-t <hash-type>]
        [-noheader]
        [-i <wrapping IV>]
        [-iv_file <IV file>]
        [-tag_size <num_tag_bytes>>]
```

例

Example

次のコマンドでは、192 ビット Triple DES (3DES) 対称キー (キーハンドル 7) をエクスポートします。HSM で 256 ビット AES キー (キーハンドル 14) を使用してキー 7 をラップします。次に、暗号化された 3DES キーを 3DES-encrypted.key ファイルに書き込みます。

次の出力は、キー 7 (3DES キー) が正常にラップされて指定したファイルに書き込まれたことを示しています。暗号化されたキーの長さは 307 バイトです。

```
        Command:  wrapKey -k 7 -w 14 -out 3DES-encrypted.key -m 4

        Key Wrapped.

        Wrapped Key written to file "3DES-encrypted.key length 307

        Cfm2WrapKey returned: 0x00 : HSM Return: SUCCESS
```

パラメータ

-h

コマンドに関するヘルプを表示します。

必須：はい

-k

エクスポートするキーのキーハンドル。所有する対称キーまたはプライベートキーのキーハンドルを入力します。キーハンドルを見つけるには、findKey コマンドを使用します。

キーがエクスポート可能であることを検証するには、getAttribute コマンドを使用して、OBJ_ATTR_EXTRACTABLE 属性の値を取得します。この属性は定数 354 で表されます。キー属性の解釈については、キー属性リファレンス を参照してください。

ユーザーが所有するキーのみをエクスポートすることができます。キーの所有者を検索するには、 getKeyInfo コマンドを使用します。

必須：はい

-w

ラップキーを指定します。HSM の AES キーまたは RSA キーのキーハンドルを入力します。このパラメータは必須です。キーハンドルを見つけるには、findKey コマンドを使用します。

ラッピングキーを作成するには、 genSymKeyを使用して AES キー (タイプ 31) を生成し、genRSAKeyPair を使用して RSA キーペア (タイプ 0) を生成します。RSA キーペアを使用している場合は、必ず一方のキーでキーをラップし、もう一方のキーでアンラップしてください。キーをラッピングキーとして使用できることを確認するには、getAttribute を使用して、定数 OBJ_ATTR_WRAP で表される 262 属性の値を取得します。

必須：はい

-out

出力ファイルのパスと名前。コマンドが成功すると、このファイルに、エクスポートされたキーの暗号化されたコピーが格納されます。既存のファイルがある場合は、警告なしに上書きされます。

必須：はい

-m

ラップメカニズムを表す値。CloudHSM は、次のメカニズムをサポートしています。

メカニズム	値
AES_KEY_WRAP_PAD_PKCS5	4
NIST_AES_WRAP_NO_PAD	5

メカニズム	値
NIST_AES_WRAP_PAD	6
RSA_AES	7
RSA_OAEP (最大データサイズについてはこのセクションの後半のメモを参照)	8
AES_GCM	10
CLOUDHSM_AES_GCM	11
RSA_PKCS (最大データサイズについてはこのセクションの後半のメモを参照)。今後の変更については、以下の注記「1」を参照してください。	12

必須：はい

> **Note**
>
> RSA_OAEP ラップメカニズムを使用する場合、ラップできる最大キーサイズは、RSA キーのモジュラスと指定されたハッシュの長さによって次のように決定されます。最大キーサイズ = (modulusLengthInBytes-2*hashLengthInBytes-2)。
> RSA_PKCS ラップメカニズムを使用する場合、ラップできる最大キーサイズは、RSA キーのモジュラスによって次のように決定されます。最大キーサイズ = (modulusLengthInバイト -11)。

-t

ハッシュアルゴリズムを表す値。CloudHSM は、次のアルゴリズムをサポートしています。

ハッシュアルゴリズム	値
SHA1	2

ハッシュアルゴリズム	値
SHA256	3
SHA384	4
SHA512	5
SHA224 (RSA_AES および RSA_OAEP メカニズムに対して有効)	6

必須：いいえ

-aad

AAD を含むファイル名。

> **Note**
>
> AES_GCM および CLOUDHSM_AES_GCM メカニズムに対してのみ有効です。

必須：いいえ

-noheader

CloudHSM 固有の キー属性を指定するヘッダーを除外します。このパラメータは、key_mgmt_util 以外のツールでキーをアンラップする場合に のみ 使用してください。

必須：いいえ

-i

初期化ベクトル (IV) (16 進値)。

> **Note**
>
> CLOUDHSM_AES_KEY_WRAP と NIST_AES_WRAP メカニズムの -noheader パラメータで渡された場合にのみ有効です。

必須：いいえ

-iv_file

応答で取得した IV 値を書き込むファイル。

> **Note**
>
> AES_GCM メカニズムの -noheader パラメータで渡された場合にのみ有効です。

必須：いいえ

-tag_size

ラップされた blob とともに保存されるタグのサイズ。

> **Note**
>
> AES_GCM と CLOUDHSM_AES_GCM メカニズムの -noheader パラメータで渡された場合にのみ有効です。タグの最小サイズは 8 です。

必須：いいえ

[1] NIST ガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

関連トピック

- exSymKey
- imSymKey
- unWrapKey

キー属性リファレンス

key_mgmt_util コマンドは、定数を使用して HSM のキーの属性を表します。このトピックは、属性の識別、コマンドでその属性を表す定数の検索、およびその値の理解に役立ちます。

キーの作成時に、キー属性を設定します。キーが永続的であるかセッションにのみ存在するかを示すトークン属性を変更するには、key_mgmt_utilの setAttribute コマンドを使用します。ラベル

を変更、属性をラップ、アンラップ、暗号化、または復号化するには、cloudhsm_mgmt_util の `setAttribute`コマンドを使用します。

属性とその定数のリストを取得するには、listAttributes を使用します。キーの属性値を取得するには、getAttribute を使用します。

次の表に、キーの属性、その定数、および有効な値を示します。

属性	定数	[値]
OBJ_ATTR_ALL	512	すべての属性を表します。
OBJ_ATTR_ALWAYS_SENSITIVE	357	0: False。 1: True。
OBJ_ATTR_CLASS	0	2 : 公開 - プライベートキーのキーペアの公開キー。 3 : 公開 - プライベートキーのキーペアの公開キー。 4: シークレット (対称) キー。
OBJ_ATTR_DECRYPT	261	0: False。 1: True。キーはデータの復号に使用できます。
OBJ_ATTR_DERIVE	268	0: False。 1: True。この関数は、キーを派生させます。
OBJ_ATTR_DESTROYABLE	370	0: False。 1: True。
OBJ_ATTR_ENCRYPT	260	0: False。 1: True。キーはデータの暗号化に使用できます。

属性	定数	[値]
OBJ_ATTR_EXTRACTABLE	354	0: False。 1: True。キーは HSM から エクスポートできます。
OBJ_ATTR_ID	258	ユーザー定義の文字列。クラスター内で一意である必要があります。デフォルトは空の文字列です。
OBJ_ATTR_KCV	371	キーのキーチェック値。詳細については、「その他の詳細」を参照してください。
OBJ_ATTR_KEY_TYPE	256	0: RSA。 1: DSA。 3: EC。 16: 汎用秘密。 18: RC4。 21: Triple DES (3DES)。 31: AES。
OBJ_ATTR_LABEL	3	ユーザー定義の文字列。クラスター内で一意である必要はありません。
OBJ_ATTR_LOCAL	355	0. False。キーは HSM にインポートされました。 1: True。

属性	定数	[値]
OBJ_ATTR_MODULUS	288	RSA キーペアを生成するために使用されたモジュラス。EC キーの場合、この値は ANSI X9.62 ECPoint 値「Q」の DER エンコーディングを 16 進数形式で表します。 他のキータイプの場合、この属性は存在しません。
OBJ_ATTR_MODULUS_BITS	289	RSA キーペアを生成するために使用されたモジュラスの長さ。EC キーの場合、これはキーの生成に使用された楕円曲線の ID を表します。 他のキータイプの場合、この属性は存在しません。
OBJ_ATTR_NEVER_EXTRACTABLE	356	0: False。 1: True。このキーは HSM からエクスポートできません。
OBJ_ATTR_PUBLIC_EXPONENT	290	RSA キーペアを生成するために使用された公開指数。 他のキータイプの場合、この属性は存在しません。

属性	定数	[値]
OBJ_ATTR_PRIVATE	2	0: False。 1: True。この属性は、認証されていないユーザーがキーの属性を表示できるかどうかを示します。CloudHSM PKCS#11 プロバイダーでは、現在パブリックセッションはサポートされていないため、すべてのキー (パブリックキーとプライベートキーのペアのパブリックキーを含む) のこの属性は 1 に設定されています。
OBJ_ATTR_SENSITIVE	259	0: False。公開 - プライベートキーのキーペアの公開キー。 1: True。
OBJ_ATTR_SIGN	264	0: False。 1: True。キーは署名 (プライベートキー) に使用できます。
OBJ_ATTR_TOKEN	1	0: False。セッションキー。 1: True。永続キー。
OBJ_ATTR_TRUSTED	134	0: False。 1: True。
OBJ_ATTR_UNWRAP	263	0: False。 1: True。キーはキーの復号に使用できます。

属性	定数	[値]
OBJ_ATTR_UNWRAP_TEMPLATE	1073742354	値は、このラッピングキーを使用してラップ解除されたキーに適用される属性テンプレートを使用する必要があります。
OBJ_ATTR_VALUE_LEN	353	キーの長さ (バイト単位)
OBJ_ATTR_VERIFY	266	0: False。 1: True。キーは検証 (パブリックキー) に使用できます。
OBJ_ATTR_WRAP	262	0: False。 1: True。キーはキーの暗号化に使用できます。
OBJ_ATTR_WRAP_TEMPLATE	1073742353	値は、属性テンプレートを使用し、このラッピングキーでラップされたキーと一致させる必要があります。
OBJ_ATTR_WRAP_WITH_TRUSTED	528	0: False。 1: True。

その他の詳細

キーチェック値 (KCV)。

キーチェック値 (KCV) は、HSM がキーをインポートまたは生成するときに生成されるキーの 3 バイトのハッシュまたはチェックサムです。キーをエクスポートした後など、HSM の外部で KCV を計算することもできます。次に、KCV 値を比較して、キーのアイデンティティと整合性を確認できます。キーの KCV を取得するには、getAttribute を使用します。

AWS CloudHSM は、次の標準メソッドを使用してキーチェック値を生成します。

- 対称キー: ゼロブロックをキーで暗号化した結果の最初の 3 バイト。
- 非対称キーペア: 公開キーの SHA-1 ハッシュの最初の 3 バイト。
- HMAC キー: 現時点では HMAC キーの KCV はサポートされていません。

AWS CloudHSM クライアント SDKs

Client SDK を使用して、暗号化オペレーションをプラットフォームまたは言語ベースのアプリケーションからハードウェアセキュリティモジュール (HSM) にオフロードします。

AWS CloudHSM には 2 つのメジャーバージョンがあり、Client SDK 5 が最新です。Client SDK 3 (以前のシリーズ) よりも多くの、さまざまな利点があります。詳細については、「Benefits of Client SDK 5」を参照してください。サポートされるプラットフォームの詳細については、「Client SDK 5 がサポートするプラットフォーム」を参照してください。

Client SDK 3 の使用の詳細については、「以前のクライアント SDK (クライアント SDK 3)」を参照してください。

the section called “PKCS #11 ライブラリ”

PKCS #11 は、ハードウェアセキュリティモジュール (HSM) で暗号化オペレーションを実行するための標準です。 AWS CloudHSM は、PKCS #11 バージョン 2.40 に準拠した PKCS #11 ライブラリの実装を提供します。

the section called “OpenSSL Dynamic Engine”

AWS CloudHSM OpenSSL Dynamic Engine を使用すると、OpenSSL API を使用して暗号化オペレーションを CloudHSM OpenSSL クラスターにオフロードできます。

the section called “JCE プロバイダー”

AWS CloudHSM JCE プロバイダーは Java 暗号化アーキテクチャ (JCA) に準拠しています。そのプロバイダーは HSM 上での暗号化オペレーションを許可します。

the section called “KSP および CNG プロバイダー”

Windows 用の AWS CloudHSM クライアントには、CNG プロバイダーと KSP プロバイダーが含まれています。現在、CNG および KSP プロバイダーをサポートしているのは、Client SDK 3 だけです。

Client SDK 5 がサポートするプラットフォーム

基本サポートは、 AWS CloudHSM クライアント SDK のバージョンごとに異なります。SDK 内のコンポーネントのプラットフォームのサポートは通常、基本サポートと一致しますが、必ずしもそう

とは限りません。特定のコンポーネントのプラットフォームサポートを決定するには、まず目的のプラットフォームが SDK のベースセクションに表示されていることを確認してから、コンポーネントセクションで除外やその他の関連情報がないか確認します。

AWS CloudHSM は 64 ビットオペレーティングシステムのみをサポートします。

プラットフォームのサポートは時間の経過とともに変化します。以前のバージョンの CloudHSM クライアント SDK では、ここに記載されているすべてのオペレーティングシステムがサポートされていない場合があります。リリースノートを使用して、以前のバージョンの CloudHSM クライアント SDK に対するオペレーティングシステムサポートを確認します。詳細については、「Client SDK AWS CloudHSM のダウンロード」を参照してください。

以前の Client SDK でサポートされるプラットフォームについては、「Client SDK 3 がサポートするプラットフォーム」を参照してください。

クライアント SDK 5 にはクライアントデーモンは必要ありません。

Client SDK 5 の Linux サポート

サポートされているプラットフォーム	x86_64 アーキテクチャ	ARM アーキテクチャ
Amazon Linux 2	はい	あり
Amazon Linux 2023	はい	あり
CentOS 7 (7.8 以降)	あり	なし
Red Hat Enterprise Linux 7 (7.8 以降)	あり	なし
Red Hat Enterprise Linux 8 (8.3 以降)	あり	なし
Red Hat Enterprise Linux 9 (9.2 以降)	はい	あり
Ubuntu 20.04 LTS	はい	いいえ
Ubuntu 22.04 LTS	はい	あり

注: SDK 5.4.2 は、CentOS 8 プラットフォームのサポートを提供する最後のリリースでした。詳細については、「CentOS のウェブサイト」を参照してください。

Client SDK 5 の Windows サポート

- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Client SDK 5 のサーバーレスサポート

- AWS Lambda
- Docker および ECS

Client SDK 5 の HSM 互換性

hsm1.medium	hsm2m.medium
Client SDK バージョン 5.0.0 以降と互換性があります。	Client SDK バージョン 5.12.0 以降と互換性があります。

コンポーネントのサポート

CloudHSM CLI

CloudHSM CLI は、管理者がクラスター内のユーザーを管理するのに役立つコマンドラインツールです。詳細については、「CloudHSM コマンドラインインターフェイス (CLI)」を参照してください。

PKCS #11 ライブラリ

PKCS #11 ライブラリは、Linux および Windows クライアント SDK 5 の基本サポートに一致するクロスプラットフォームコンポーネントです。詳細については、「the section called “Client SDK 5 の Linux サポート”」および「the section called “Client SDK 5 の Windows サポート”」を参照してください。

OpenSSL Dynamic Engine

OpenSSL Dynamic Engine は、OpenSSL 1.0.2、1.1.1、または 3.x を必要とする Linux 専用コンポーネントです。

JCE プロバイダー

JCE プロバイダーは、サポートされているすべてのプラットフォームで OpenJDK 8、OpenJDK 11、OpenJDK 17、および OpenJDK 21 と互換性がある Java SDK です。

Client SDK 5 の利点

Client SDK 3 と比較して、Client SDK 5 は管理が容易で、設定しやすく、信頼性も高くなっています。Client SDK 5 には、Client SDK 3 には他にも主要な利点がいくつかあります。

サーバーレスアーキテクチャ向けの設計

Client SDK 5 にはクライアントデーモンが必要ないため、バックグラウンドサービスを管理する必要がなくなりました。これは、ユーザーにとって次に挙げる重要なポイントで役立ちます。

- アプリケーションの起動プロセスを簡素化します。CloudHSM を使い始めるために必要なことは、アプリケーションを実行する前に SDK を設定することだけです。
- プロセスを常に実行する必要がないため、Lambda や Elastic Container Service (ECS) などのサーバーレスコンポーネントとの統合が容易になります。

サードパーティとの統合が強化され、ポータビリティが容易に

Client SDK 5 は JCE 仕様に厳密に準拠しており、異なる JCE プロバイダー間のポータビリティが容易で、サードパーティとの統合も良好です。

ユーザーエクスペリエンスと設定可能性が向上

Client SDK 5 では、ログメッセージの読みやすさが向上し、例外やエラー処理メカニズムが明確になり、ユーザーによるセルフサービスのトリアージがはるかに簡単になりました。SDK 5 にはさまざまな設定も用意されており、設定ツールページに一覧表示されています。

より広範なプラットフォームサポート

Client SDK 5 は、最新のオペレーティングプラットフォームにより幅広いサポートを提供しています。これには、ARM テクノロジーのサポートのほか、JCE、PKCS #11、および OpenSSL のサポートの強化が含まれます。詳細については、「Supported platforms」を参照してください。

その他の機能やメカニズム

Client SDK 5 には、Client SDK 3 にはない機能やメカニズムが追加されており、Client SDK 5 には今後もメカニズムが追加されていく予定です。

Client SDK 3 から Client SDK 5 への移行

Client SDK 3 から Client SDK 5 への移行の詳細な手順については、個々の Client SDK の移行手順を参照してください。

- PKCS #11 ライブラリをクライアント SDK 3 からクライアント SDK 5 に移行する
- OpenSSL 動的エンジンをクライアント SDK 3 からクライアント SDK 5 に移行する
- JCE プロバイダーをクライアント SDK 3 からクライアント SDK 5 に移行する
- Client SDK 3 CMU および KMU から Client SDK 5 CloudHSM CLI に移行する

CloudHSM CLI でサポートされていない機能やユースケースについては、 に連絡してサポートを依頼してください。

> **Note**
>
> クライアント SDK 5 PKCS #11 ライブラリが Windows プラットフォームでサポートされるようになりました。CNG プロバイダーと KSP プロバイダーが置き換え先と見なすことができ、そうすべきほとんどのユースケースを処理できます。KSP は現在、クライアント SDK 3 でのみ使用できます。

PKCS #11 ライブラリ

PKCS #11 は、ハードウェアセキュリティモジュール (HSM) で暗号化オペレーションを実行するための標準です。 AWS CloudHSM は、PKCS #11 バージョン 2.40 に準拠した PKCS #11 ライブラリの実装を提供します。

ブートストラップの詳細については、「クラスターへの接続」を参照してください。トラブルシューティングについては、「」を参照してくださいPKCS#11 ライブラリの既知の問題。

Client SDK 3 の使用の詳細については、「以前のクライアント SDK (クライアント SDK 3)」を参照してください。

トピック

Client SDK 5 用の PKCS #11 ライブラリをインストールする

このトピックでは、クライアント SDK 5 バージョンシリーズ用の PKCS #11 ライブラリの最新バージョンをインストールする手順について説明します。Client SDK または PKCS #11 ライブラリの詳細については、[Client SDK の使用] と [PKCS #11 ライブラリ] を参照してください。

インストール

クライアント SDK 5 では、クライアントデーモンをインストールまたは実行する必要はありません。

クライアント SDK 5 で単一の HSM クラスターを実行するには、まず `disable_key_availability_check` を `True` に設定してクライアントキーの耐久性の設定を管理する必要があります。詳細については、キーの同期 と クライアント SDK 5 設定ツール を参照してください。

Client SDK 5 の PKCS #11 ライブラリの詳細については、[PKCS #11 ライブラリ] を参照してください。

> **Note**
>
> クライアント SDK 5 で単一の HSM クラスターを実行するには、まず `disable_key_availability_check` を `True` に設定してクライアントキーの耐久性の設定を管理する必要があります。詳細については、キーの同期 と クライアント SDK 5 設定ツール を参照してください。

PKCS #11 ライブラリをインストールおよび設定するには

1. 次のコマンドを使用して、PKCS #11 ライブラリのダウンロードとインストールを行います。

   Amazon Linux 2

   X86_64 アーキテクチャ上で Amazon Linux 2 用の PKCS #11 ライブラリをインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
   ```

   ARM64 アーキテクチャ上で Amazon Linux 2 用の PKCS #11 ライブラリをインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.aarch64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-pkcs11-latest.el7.aarch64.rpm
   ```

   Amazon Linux 2023

   Amazon Linux 2023 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-pkcs11-latest.amzn2023.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-pkcs11-latest.amzn2023.x86_64.rpm
   ```

   Amazon Linux 2023 用の PKCS #11 ライブラリを ARM64 アーキテクチャにインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-pkcs11-latest.amzn2023.aarch64.rpm
   ```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.amzn2023.aarch64.rpm
```

CentOS 7 (7.8+)

CentOS 7.8+ 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

RHEL 7 (7.8+)

RHEL 7 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

RHEL 8 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

RHEL 9 (9.2+)

RHEL 9 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-pkcs11-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el9.x86_64.rpm
```

RHEL 9 用の PKCS #11 ライブラリを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-pkcs11-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el9.aarch64.rpm
```

Ubuntu 20.04 LTS

Ubuntu 20.04 LTS 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/cloudhsm-pkcs11_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u20.04_amd64.deb
```

Ubuntu 22.04 LTS

Ubuntu 22.04 LTS 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-pkcs11_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u22.04_amd64.deb
```

Ubuntu 22.04 LTS 用の PKCS #11 ライブラリを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-pkcs11_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u22.04_arm64.deb
```

Windows Server 2016

Windows Server 2016 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

1. [Client SDK 5 用の PKCS #11 ライブラリ] をダウンロードします。
2. Windows 管理権限を持つ PKCS #11 ライブラリインストーラ (AWSCloudHSMPKCS11-latest.msi) を実行します。

Windows Server 2019

Windows Server 2019 用の PKCS #11 ライブラリを X86_64 アーキテクチャにインストールします。

1. [Client SDK 5 用の PKCS #11 ライブラリ] をダウンロードします。
2. Windows 管理権限を持つ PKCS #11 ライブラリインストーラ (AWSCloudHSMPKCS11-latest.msi) を実行します。

2. 構成ツールを使用して、証明書の発行場所を指定します。手順については、「発行証明書の場所を指定する」を参照してください。
3. クラスターに接続して使用するには、「クライアント SDK をブートストラップする」を参照してください。
4. PKCS #11 ライブラリのファイルは、次の場所にあります。

   - Linuxのバイナリ、設定スクリプト、およびログファイル:

     ```
     /opt/cloudhsm
     ```

     Windows のバイナリ:

     ```
     C:\ProgramFiles\Amazon\CloudHSM
     ```

     Windows の設定スクリプトとログファイル:

     ```
     C:\ProgramData\Amazon\CloudHSM
     ```

PKCS #11 ライブラリへの認証

PKCS #11 を使用すると、アプリケーションは HSMs で特定の [Crypto User (CU)] として実行されます。アプリケーションは、CU が所有して共有するキーのみを表示および管理できます。既存の CU を HSM で使用することも、アプリケーションに新しい CU を作成することもできます。CU の管理については、「CloudHSM CLI による HSM ユーザーの管理」および「CloudHSM 管理ユーティリティ (CMU) による HSM ユーザーの管理」を参照してください

PKCS #11 に CU を指定するには、PKCS #11 [C_Login 関数] のピンパラメーターを使用します。の場合 AWS CloudHSM、ピンパラメータの形式は次のとおりです。

```
<CU_user_name>:<password>
```

たとえば、次のコマンドでユーザー名 CryptoUser とパスワード CUPassword123! を使用して PKCS #11 ライブラリのピンを CU に設定します。

```
CryptoUser:CUPassword123!
```

PKCS #11 ライブラリでサポートされているキータイプ

PKCS #11 ライブラリは、次のキータイプをサポートしています。

キータイプ	説明
AES	128、192、256 ビットの AES キーを生成します。
Triple DES (3DES、DESede)	192 ビットのトリプル DES キーを生成します。今後の変更については、以下の注記「1」を参照してください。
EC	secp224r1 (P-224)、secp256r1 (P-256)、secp256k1 (ブロックチェーン)、secp384r1 (P-384)、secp521r1 (P-521) のカーブを使用してキーを生成します。
[GENERIC_SECRET]	1 ~ 800 バイトの汎用シークレットを生成します。

キータイプ	説明
RSA	256 ビットの増分で、2048 ~ 4096 ビットの RSA キーを生成します。

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

PKCS #11 ライブラリでサポートされているメカニズム

PKCS #11 ライブラリは PKCS #11 仕様のバージョン 2.40 に準拠しています。PKCS#11 を使用して暗号化機能を呼び出すには、指定されたメカニズムで関数を呼び出します。次の表は、 AWS CloudHSMでサポートされている関数とメカニズムの組み合わせをまとめたものです。

PKCS #11 ライブラリは、次のアルゴリズムをサポートしています。

- [暗号化と復号化] AES-CBC、AES-CTR、AES-ECB、AES-GCM、DES3-CBC、DES3-ECB、RSA-OAEP、RSA-PKCS
- [署名と確認] RSA、HMAC、ECDSA (ハッシュあり、なし)
- [ハッシュ/ダイジェスト] SHA1、SHA224、SHA256、SHA384、SHA512
- [キーラップ] AES キーラップ、[1] AES-GCM、RSA-AES、RSA-OAEP

トピック

- キーとキーペアの関数を生成する
- 署名および検証
- リカバリ機能への署名、リカバリ、検証
- ダイジェスト関数
- 暗号化と復号
- キー機能の導出
- ラップ関数とアンラップ関数
- 各メカニズムの最大データサイズ
- メカニズムの注釈

キーとキーペアの関数を生成する

PKCS #11 ライブラリの AWS CloudHSM ソフトウェアライブラリでは、キーとキーペアの生成関数に次のメカニズムを使用できます。

- CKM_RSA_PKCS_KEY_PAIR_GEN
- CKM_RSA_X9_31_KEY_PAIR_GEN – このメカニズムは機能的には CKM_RSA_PKCS_KEY_PAIR_GEN メカニズムと似ていますが、p と q の生成に関してより強力な保証を提供します。
- CKM_EC_KEY_PAIR_GEN
- CKM_GENERIC_SECRET_KEY_GEN
- CKM_AES_KEY_GEN
- CKM_DES3_KEY_GEN— 今後の変更は脚注 5 に記載されています。

署名および検証

PKCS #11 ライブラリ用の AWS CloudHSM ソフトウェアライブラリを使用すると、署名および検証関数に次のメカニズムを使用できます。Client SDK 5 では、データはソフトウェアでローカルにハッシュされます。つまり、SDK でハッシュできるデータのサイズに制限はありません。

Client SDK 5 では、RSA と ECDSA のハッシュはローカルで行われるため、データ制限はありません。HMAC にはデータ制限があります。詳細については、脚注 2 を参照してください。

RSA

- CKM_RSA_X_509
- CKM_RSA_PKCS – シングルパートのオペレーションのみ。
- CKM_RSA_PKCS_PSS – シングルパートのオペレーションのみ。
- CKM_SHA1_RSA_PKCS
- CKM_SHA224_RSA_PKCS
- CKM_SHA256_RSA_PKCS
- CKM_SHA384_RSA_PKCS
- CKM_SHA512_RSA_PKCS
- CKM_SHA512_RSA_PKCS
- CKM_SHA1_RSA_PKCS_PSS

- CKM_SHA224_RSA_PKCS_PSS
- CKM_SHA256_RSA_PKCS_PSS
- CKM_SHA384_RSA_PKCS_PSS
- CKM_SHA512_RSA_PKCS_PSS

ECDSA

- CKM_ECDSA – シングルパートのオペレーションのみ。
- CKM_ECDSA_SHA1
- CKM_ECDSA_SHA224
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384
- CKM_ECDSA_SHA512

HMAC

- CKM_SHA_1_HMAC[2]
- CKM_SHA224_HMAC[2]
- CKM_SHA256_HMAC[2]
- CKM_SHA384_HMAC[2]
- CKM_SHA512_HMAC[2]

CMAC

- CKM_AES_CMAC

リカバリ機能への署名、リカバリ、検証

Client SDK 5 は、署名と復号機能をサポートしていません。

ダイジェスト関数

PKCS #11 ライブラリ用の AWS CloudHSM ソフトウェアライブラリを使用すると、ダイジェスト関数に次のメカニズムを使用できます。Client SDK 5 では、データはソフトウェアでローカルにハッシュされます。つまり、SDK でハッシュできるデータのサイズに制限はありません。

- CKM_SHA_1
- CKM_SHA224
- CKM_SHA256
- CKM_SHA384
- CKM_SHA512

暗号化と復号

PKCS #11 ライブラリ用の AWS CloudHSM ソフトウェアライブラリを使用すると、暗号化および復号関数に次のメカニズムを使用できます。

- CKM_RSA_X_509
- CKM_RSA_PKCS – シングルパートのオペレーションのみ。今後の変更は脚注 5 に記載されています。
- CKM_RSA_PKCS_OAEP – シングルパートのオペレーションのみ。
- CKM_AES_ECB
- CKM_AES_CTR
- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_DES3_CBC— 今後の変更は脚注 5 に記載されています。
- CKM_DES3_ECB— 今後の変更は脚注 5 に記載されています。
- CKM_DES3_CBC_PAD— 今後の変更は脚注 5 に記載されています。
- CKM_AES_GCM [1, 2]
- CKM_CLOUDHSM_AES_GCM[3]

キー機能の導出

PKCS #11 ライブラリ用の AWS CloudHSM ソフトウェアライブラリを使用すると、派生関数に次のメカニズムを使用できます。

- CKM_SP800_108_COUNTER_KDF

ラップ関数とアンラップ関数

PKCS #11 ライブラリ用の AWS CloudHSM ソフトウェアライブラリを使用すると、ラップ関数とアンラップ関数に次のメカニズムを使用できます。

AES キーラップに関する追加情報については、[AES キーラップ] を参照してください。

- CKM_RSA_PKCS – シングルパートのオペレーションのみ。今後の変更は、脚注 5 に記載されています。
- CKM_RSA_PKCS_OAEP[4]
- CKM_AES_GCM[1, 3]
- CKM_CLOUDHSM_AES_GCM[3]
- CKM_RSA_AES_KEY_WRAP
- CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD[3]
- CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD[3]
- CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD[3]

各メカニズムの最大データサイズ

次の表は、各メカニズムに設定されている最大データサイズを示します:

最大データセットサイズ

メカニズム	最大データサイズ (バイト単位)
CKM_SHA_1_HMAC	16288
CKM_SHA224_HMAC	16256

メカニズム	最大データサイズ (バイト単位)
CKM_SHA256_HMAC	16288
CKM_SHA384_HMAC	16224
CKM_SHA512_HMAC	16224
CKM_AES_CBC	16272
CKM_AES_GCM	16224
CKM_CLOUDHSM_AES_GCM	16224
CKM_DES3_CBC	16280

メカニズムの注釈

- [1] AES-GCM の暗号化を実行している際、HSM はアプリケーションからの初期化ベクトル (IV) データを受け入れません。HSM が生成した IV を使用する必要があります。HSM で生成された 12 バイトの IV は、指定した CK_GCM_PARAMS パラメータ構造の pIV 要素が指すメモリ参照に書き込まれます。ユーザーが混乱しないよう、バージョン 1.1.1 以降の PKCS#11 SDK では、AES-GCM 暗号化が初期化されると、pIV はゼロ化されたバッファを指し示すようになっています。
- [2] 以下の仕組みでデータをオペレーションする場合、データバッファが最大データサイズを超えるとオペレーションがエラーとなります。これらのメカニズムでは、すべてのデータ処理が HSM 内で行われる必要があります。各メカニズムの最大データサイズセットについては、「各メカニズムの最大データサイズ」を参照してください。
- [3] ベンダー定義のメカニズム。CloudHSM ベンダー定義のメカニズムを使用するには、コンパイル時に PKCS #11 アプリケーションに /opt/cloudhsm/include/pkcs11t.h を含める必要があります。

  CKM_CLOUDHSM_AES_GCM: この独自のメカニズムは、標準 CKM_AES_GCM よりもプログラム的に安全な代替手段です。これは、HSM によって生成された IV を、暗号の初期化中に提供される CK_GCM_PARAMS 構造体に書き戻すのではなく、暗号文の先頭に付加します。このメカニズムは C_Encrypt、C_WrapKey、C_Decrypt、C_UnwrapKey 関数で使用できます。このメカニズムを使用する場合は、CK_GCM_PARAMS 構造体内の pIV 変数を NULL に設定する必要があります。このメカニズムを C_Decrypt および C_UnwrapKey と共に使用する場合、IV は、ラップ解除される暗号文の前に付加されることが想定されます。

CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD: PKCS #5 パディングを使用する AES キーラップ。

CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD: ゼロパディングを使用する AES キーラップ。

- [4] 次の CK_MECHANISM_TYPE および CK_RSA_PKCS_MGF_TYPE は、CK_RSA_PKCS_OAEP_PARAMS の CKM_RSA_PKCS_OAEP としてサポートされています:
  - CKG_MGF1_SHA1 を使用する CKM_SHA_1
  - CKG_MGF1_SHA224 を使用する CKM_SHA224
  - CKG_MGF1_SHA256 を使用する CKM_SHA256
  - CKM_MGF1_SHA384 を使用する CKM_SHA384
  - CKM_MGF1_SHA512 を使用する CKM_SHA512
- [5] NIST ガイダンスに従い、これは 2023 年以降、FIPS モードでのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

PKCS #11 ライブラリでサポートされている API オペレーション

PKCS #11 ライブラリは、次の PKCS #11 API オペレーションをサポートしています。

- C_CloseAllSessions
- C_CloseSession
- C_CreateObject
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DeriveKey
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate

- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate
- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignUpdate

- C_UnWrapKey
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyUpdate
- C_WrapKey

PKCS #11 ライブラリでサポートされているキー属性

キーオブジェクトには、パブリックキー、プライベートキー、またはシークレットキーを指定できます。キーオブジェクトで許可されているアクションは属性で指定されます。属性は、キーオブジェクトの作成時に定義されます。PKCS#11 ライブラリを使用する際、PKCS#11 標準で指定されたデフォルト値が割り当てられます。

AWS CloudHSM は、PKCS #11 仕様に記載されているすべての属性をサポートしているわけではありません。サポートするすべての属性の仕様に準拠しています。これらの属性は、それぞれのテーブルにリストされています。

オブジェクトを作成、変更、またはコピーする C_CreateObject、C_GenerateKey、C_GenerateKeyPair、C_UnwrapKey、C_DeriveKey などの暗号化関数は、属性テンプレートをパラメータの 1 つとして使用します。オブジェクトの作成中に属性テンプレートを渡す方法の詳細については、「PKCS #11 ライブラリを使用したキーの生成」のサンプルを参照してください。

PKCS#11 ライブラリの属性テーブルの解釈

PKCS#11 ライブラリのテーブルには、キータイプによって異なる属性のリストが含まれています。で特定の暗号化関数を使用する場合、特定の属性が特定のキータイプでサポートされているかどうかを示します AWS CloudHSM。

凡例:

- ✓ CloudHSM が特定のキータイプの属性をサポートしていることを示します。
- ✖ CloudHSM が特定のキータイプの属性をサポートしていないこと示します。
- R は、属性値が特定のキータイプに対して読み取り専用に設定されていることを示します。
- S は、属性が機密であるため、GetAttributeValue で読み取れないことを示します。

- [Default Value] 列のセルが空の場合は、属性に割り当てられている特定のデフォルト値がないことを示します。

GenerateKeyPair

属性	キータイプ				デフォルト値
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	
CKA_CLASS	✓	✓	✓	✓	
CKA_KEY_TYPE	✓	✓	✓	✓	
CKA_LABEL	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	True
CKA_TOKEN	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✓	✖	✓	False
CKA_DECRYPT	✓	✖	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	False

属性	キータイプ				デフォルト値
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	✓	True
CKA_SIGN	✓	✖	✓	✖	False
CKA_SIGN_RECOVER	✖	✖	✖	✖	
CKA_VERIFY	✖	✓	✖	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	✖	
CKA_WRAP	✖	✓	✖	✓	False
CKA_WRAP_TEMPLATE	✖	✓	✖	✓	
CKA_TRUSTED	✖	✓	✖	✓	False
CKA_WRAP_WITH_TRUSTED	✓	✖	✓	✖	False
CKA_UNWRAP	✓	✖	✓	✖	False
CKA_UNWRAP_TEMPLATE	✓	✖	✓	✖	

属性	キータイプ				デフォルト値
CKA_SENSITIVE	✓[1]	✖	✓[1]	✖	True
CKA_ALWAYS_SENSITIVE	R	✖	R	✖	
CKA_EXTRACTABLE	✓	✖	✓	✖	True
CKA_NEVER_EXTRACTABLE	R	✖	R	✖	
CKA_MODULUS	✖	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✓[2]	
CKA_PRIME_1	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	✖	

属性	キータイプ				デフォルト値
CKA_PRIVATE_EXPONENT	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	✓[2]	
CKA_EC_PARAMS	✖	✓[2]	✖	✖	
CKA_EC_POINT	✖	✖	✖	✖	
CKA_VALUE	✖	✖	✖	✖	
CKA_VALUE_LEN	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	

GenerateKey

属性	キータイプ			デフォルト値
	AES	DES3	汎用シークレット	
CKA_CLASS	✓	✓	✓	
CKA_KEY_TYPE	✓	✓	✓	

属性	キータイプ			デフォルト値
CKA_LABEL	✓	✓	✓	
CKA_ID	✓	✓	✓	
CKA_LOCAL	R	R	R	True
CKA_TOKEN	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✓	✓	✖	False
CKA_DECRYPT	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	True
CKA_SIGN	✓	✓	✓	True
CKA_SIGN_RECOVER	✖	✖	✖	
CKA_VERIFY	✓	✓	✓	True
CKA_VERIFY_RECOVER	✖	✖	✖	

属性	キータイプ			デフォルト値
CKA_WRAP	✓	✓	✖	False
CKA_WRAP_TEMPLATE	✓	✓	✖	
CKA_TRUSTED	✓	✓	✖	False
CKA_WRAP_WITH_TRUSTED	✓	✓	✓	False
CKA_UNWRAP	✓	✓	✖	False
CKA_UNWRAP_TEMPLATE	✓	✓	✖	
CKA_SENSITIVE	✓	✓	✓	True
CKA_ALWAYS_SENSITIVE	✖	✖	✖	
CKA_EXTRACTABLE	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	
CKA_MODULUS	✖	✖	✖	

属性	キータイプ			デフォルト値
CKA_MODULUS_BITS	✖	✖	✖	
CKA_PRIME_1	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	
CKA_EC_POINT	✖	✖	✖	
CKA_VALUE	✖	✖	✖	
CKA_VALUE_LEN	✓[2]	✖	✓[2]	

属性	キータイプ			デフォルト値
CKA_CHECK_VALUE	R	R	R	

CreateObject

属性	キータイプ							デフォルト値
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	AES	DES3	汎用シークレット	
CKA_CLASS	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✖	✖	✓	✓	✓	✖	False

属性	キータイプ							デフォルト値
CKA_DECRYPT	✖	✖	✓	✖	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✖	✓	✖	✓	✓	✓	False
CKA_SIGN_RECOVER	✖	✖	✖	✖	✖	✖	✖	False
CKA_VERIFY	✖	✓	✖	✓	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	✖	✖	✖	✖	
CKA_WRAP	✖	✖	✖	✓	✓	✓	✖	False
CKA_WRAP_TEMPLATE	✖	✓	✖	✓	✓	✓	✖	
CKA_TRUSTED	✖	✓	✖	✓	✓	✓	✖	False
CKA_WRAP_WITH_TRUSTED	✓	✖	✓	✖	✓	✓	✓	False

属性	キータイプ							デフォルト値
CKA_UNWRAP	✖	✖	✓	✖	✓	✓	✖	False
CKA_UNWRAP_TEMPLATE	✓	✖	✓	✖	✓	✓	✖	
CKA_SENSITIVE	✓	✖	✓	✖	✓	✓	✓	True
CKA_ALWAYS_SENSITIVE	R	✖	R	✖	R	R	R	
CKA_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	✖	R	✖	R	R	R	
CKA_MODULUS	✖	✖	✓[2]	✓[2]	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✖	✖	✖	✖	
CKA_PRIME_1	✖	✖	✓	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✓	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✓	✖	✖	✖	✖	

属性	キータイプ							デフォルト値
CKA_EXPONENT_1	✖	✖	✓	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✓	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✓[2]	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✓[2]	✓[2]	✖	✖	✖	
CKA_EC_PARAMS	✓[2]	✓[2]	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✓[2]	✖	✖	✖	✖	✖	
CKA_VALUE	✓[2]	✖	✖	✖	✓[2]	✓[2]	✓[2]	
CKA_VALUE_LEN	✖	✖	✖	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	R	R	R	

UnwrapKey

属性	キータイプ					デフォルト値
	EC プライベート	RSA プライベート	AES	DES3	汎用シークレット	
CKA_CLASS	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✖	✓	✓	✖	False
CKA_DECRYPT	✖	✓	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True

属性	キータイプ					デフォルト値
CKA_DESTROYABLE	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✓	✓	✓	✓	False
CKA_SIGN_RECOVER	✖	✖	✖	✖	✖	False
CKA_VERIFY	✖	✖	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	✖	✖	
CKA_WRAP	✖	✖	✓	✓	✖	False
CKA_UNWRAP	✖	✓	✓	✓	✖	False
CKA_SENSITIVE	✓	✓	✓	✓	✓	True
CKA_EXTRACTABLE	✓	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	R	R	
CKA_ALWAYS_SENSITIVE	R	R	R	R	R	
CKA_MODULUS	✖	✖	✖	✖	✖	

属性	キータイプ					デフォルト値
CKA_MODULUS_BITS	✖	✖	✖	✖	✖	
CKA_PRIME_1	✖	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✖	✖	✖	✖	
CKA_VALUE	✖	✖	✖	✖	✖	

属性	キータイプ					デフォルト値
CKA_VALUE_LEN		✖	✖	✖	✖	✖	
CKA_CHECK_VALUE		R	R	R	R	R	

DeriveKey

属性	キータイプ				デフォルト値
	AES	DES3	汎用シークレット	
CKA_CLASS	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	
CKA_ID	✓	✓	✓	
CKA_LOCAL	R	R	R	True
CKA_TOKEN	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✓	✓	✖	False
CKA_DECRYPT	✓	✓	✖	False

属性	キータイプ			デフォルト値
CKA_DERIVE	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓[1]	✓[1]	✓[1]	True
CKA_SIGN	✓	✓	✓	False
CKA_SIGN_RECOVER	✖	✖	✖	
CKA_VERIFY	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	
CKA_WRAP	✓	✓	✖	False
CKA_UNWRAP	✓	✓	✖	False
CKA_SENSITIVE	R	R	R	True
CKA_EXTRACTABLE	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	

属性	キータイプ			デフォルト値
CKA_ALWAYS_SENSITIVE	R	R	R	
CKA_MODULUS	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	
CKA_PRIME_1	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	

属性	キータイプ			デフォルト値
CKA_EC_POINT	✖	✖	✖	
CKA_VALUE	✖	✖	✖	
CKA_VALUE_LEN	✓[2]	✖	✓[2]	
CKA_CHECK_VALUE	R	R	R	

GetAttributeValue

属性	キータイプ						
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	AES	DES3	汎用シークレット
CKA_CLASS	✓	✓	✓	✓	✓	✓	✓
CKA_KEY_TYPE	✓	✓	✓	✓	✓	✓	✓
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓
CKA_ID	✓	✓	✓	✓	✓	✓	✓
CKA_LOCAL	✓	✓	✓	✓	✓	✓	✓
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓

属性	キータイプ						
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
CKA_ENCRYPT	✖	✖	✖	✓	✓	✓	✖
CKA_DECRYPT	✖	✖	✓	✖	✓	✓	✖
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓
CKA_MODIFIABLE	✓	✓	✓	✓	✓	✓	✓
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓
CKA_SIGN	✓	✖	✓	✖	✓	✓	✓
CKA_SIGN_RECOVER	✖	✖	✓	✖	✖	✖	✖
CKA_VERIFY	✖	✓	✖	✓	✓	✓	✓
CKA_VERIFY_RECOVER	✖	✖	✖	✓	✖	✖	✖
CKA_WRAP	✖	✖	✖	✓	✓	✓	✖
CKA_WRAP_TEMPLATE	✖	✓	✖	✓	✓	✓	✖
CKA_TRUSTED	✖	✓	✖	✓	✓	✓	✓

属性	キータイプ						
CKA_WRAP_WITH_TRUSTED	✓	✖	✓	✖	✓	✓	✓
CKA_UNWRAP	✖	✖	✓	✖	✓	✓	✖
CKA_UNWRAP_TEMPLATE	✓	✖	✓	✖	✓	✓	✖
CKA_SENSITIVE	✓	✖	✓	✖	✓	✓	✓
CKA_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓
CKA_NEVER_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓
CKA_ALWAYS_SENSITIVE	R	R	R	R	R	R	R
CKA_MODULUS	✖	✖	✓	✓	✖	✖	✖
CKA_MODULUS_BITS	✖	✖	✖	✓	✖	✖	✖
CKA_PRIME_1	✖	✖	S	✖	✖	✖	✖
CKA_PRIME_2	✖	✖	S	✖	✖	✖	✖

属性	キータイプ						
CKA_COEFFICIENT	✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_1	✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_2	✖	✖	S	✖	✖	✖	✖
CKA_PRIVATE_EXPONENT	✖	✖	S	✖	✖	✖	✖
CKA_PUBLIC_EXPONENT	✖	✖	✓	✓	✖	✖	✖
CKA_EC_PARAMS	✓	✓	✖	✖	✖	✖	✖
CKA_EC_POINT	✖	✓	✖	✖	✖	✖	✖
CKA_VALUE	S	✖	✖	✖	✓	✓	✓
CKA_VALUE_LEN	✖	✖	✖	✖	✓	✖	✓
CKA_CHECK_VALUE	✓	✓	✓	✓	✓	✓	✖

属性注釈

- [1] この属性はファームウェアによって部分的にサポートされており、デフォルト値にのみ明示的に設定する必要があります。

- [2] 必須属性。

属性の変更

オブジェクトの属性には、オブジェクトが作成された後に変更できるものもありますが、変更できないものもあります。属性を修正するには、cloudhsm_mgmt_util の[setAttribute] コマンドを使用します。また、cloudhsm_mgmt_util の listAttribute コマンドを使用して属性一覧とそれを表す定数を取得することも可能です。

次のリストで、オブジェクトの作成後に変更が許可荒れている許可されている属性が表示されます。

- CKA_LABEL
- CKA_TOKEN

> **Note**
>
> 変更が許可されるには、セッションキーをトークンキーに変更する場合のみです。key_mgmt_util の setAttribute コマンドを使用して属性値を変更します。

- CKA_ENCRYPT
- CKA_DECRYPT
- CKA_SIGN
- CKA_VERIFY
- CKA_WRAP
- CKA_UNWRAP
- CKA_LABEL
- CKA_SENSITIVE
- CKA_DERIVE

> **Note**
>
> この属性ではキー取得がサポートされています。すべてのパブリックキーで False を指定する必要があります。True に設定することはできません。シークレットキーまたは EC プライベートキーに対しては、True または False に設定できます。

- CKA_TRUSTED

> **Note**
>
> この属性は Crypto Officer (CO) のみによって `True` または `False` に設定できます。

- `CKA_WRAP_WITH_TRUSTED`

> **Note**
>
> この属性をエクスポート可能なデータキーに適用して、このキーを `CKA_TRUSTED` としてマークされたキーでのみラップできるように指定します。1度 `CKA_WRAP_WITH_TRUSTED` を true に設定すると属性は読み取り専用になり、属性を変更または削除することはできません。

エラーコードの解釈

特定のキーでサポートされていない属性をテンプレートで指定すると、エラーが発生します。次の表には、仕様に違反した場合に生成されるエラーコードが含まれています。

エラーコード	説明
`CKR_TEMPLATE_INCONSISTENT`	PKCS#11 仕様に準拠しているが、CloudHSM でサポートされていない属性を属性テンプレートで指定した場合に、このエラーが発生します。
`CKR_ATTRIBUTE_TYPE_INVALID`	PKCS#11 仕様に準拠しているが、CloudHSM でサポートされていない属性の値を取得すると、このエラーが発生します。
`CKR_ATTRIBUTE_INCOMPLETE`	このエラーは、属性テンプレートで必須属性を指定しなかった場合に発生します。
`CKR_ATTRIBUTE_READ_ONLY`	このエラーは、属性テンプレートで読み取り専用属性を指定した場合に発生します。

PKCS #11 ライブラリのコードサンプル

のコードサンプルは、PKCS #11 ライブラリを使用して基本的なタスクを実行する方法 GitHub を示しています。

前提条件

サンプルを実行する前に、以下のステップを実行して環境をセットアップします。

- Client SDK 5 の PKCS #11 ライブラリ のインストールと設定をします。
- 暗号化ユーザー (CU) の設定をします。アプリケーションは、この HSM アカウントを使用して HSM でコードサンプルを実行します。

コードサンプル

PKCS#11 用 AWS CloudHSM ソフトウェアライブラリのコードサンプルは、 で入手できますGitHub。このリポジトリには、暗号化、復号化、署名、検証など、PKCS #11 を使用して一般的な操作を行う方法の例が含まれています。

- キーの生成 (AES、RSA、EC)
- キー属性のリスト化
- AES GCM を使用したデータの暗号化および復号
- AES_CTR を使用したデータの暗号化および復号
- 3DES を使用したデータの暗号化および復号
- RSAを使用したデータの署名と検証
- HMAC KDFを使用したキーの取得
- PKCS #5 パディングありの AES を使用したキーのラップとラップ解除
- パディングなしの AES を使用したキーのラップとラップ解除
- ゼロパディングありの AES を使用したキーのラップとラップ解除
- AES-GCM を使用したキーのラップとラップ解除
- RSA を使用したキーのラップとラップ解除

PKCS #11 ライブラリをクライアント SDK 3 からクライアント SDK 5 に移行する

このトピックでは、PKCS #11 ライブラリをクライアント SDK 3 からクライアント SDK 5 に移行します。移行の利点については、「」を参照してくださいClient SDK 5 の利点。

では AWS CloudHSM、顧客アプリケーションは AWS CloudHSM クライアントソフトウェア開発キット (SDK) を使用して暗号化オペレーションを実行します。Client SDK 5 は、引き続き新機能とプラットフォームサポートが追加されているプライマリ SDK です。

すべてのプロバイダーの移行手順を確認するには、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。

重大な変更に対処して準備する

これらの重大な変更を確認し、それに応じて開発環境でアプリケーションを更新します。

ラップメカニズムが変更されました

Client SDK 3 のメカニズム	同等の Client SDK 5 メカニズム
CKM_AES_KEY_WRAP	CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD
CKM_AES_KEY_WRAP_PAD	CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD
CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD	CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD
CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD	CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD
CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD	CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD

ECDH

Client SDK 3 では、ECDH を使用して KDF を指定できます。この機能は現在、クライアント SDK 5 では使用できません。アプリケーションにこの機能が必要な場合は、 に連絡してサポートを依頼してください。

キーハンドルがセッション固有になりました

クライアント SDK 5 でキーハンドルを正常に使用する場合、アプリケーションを実行するたびにキーハンドルを取得する必要があります。異なるセッションで同じキーハンドルを使用する既存のアプリケーションがある場合は、アプリケーションを実行するたびにキーハンドルを取得するようにコードを変更する必要があります。キーハンドルの取得については、この AWS CloudHSM PKCS #11 サンプル を参照してください。この変更は、PKCS #11 2.40 仕様 に準拠したものです。

Client SDK 5 への移行

このセクションの手順に従って、Client SDK 3 から Client SDK 5 に移行します。

> **Note**
>
> Amazon Linux、Ubuntu 16.04、Ubuntu 18.04、CentOS 6、CentOS 8、および RHEL 6 は現在、クライアント SDK 5 ではサポートされていません。現在、クライアント SDK 3 でこれらのプラットフォームのいずれかを使用している場合は、クライアント SDK 5 に移行するときに別のプラットフォームを選択する必要があります。

1. Client SDK 3 用の PKCS #11 ライブラリをアンインストールします。

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-pkcs11
   ```

   CentOS 7

   ```
   $ sudo yum remove cloudhsm-pkcs11
   ```

   RHEL 7

   ```
   $ sudo yum remove cloudhsm-pkcs11
   ```

   RHEL 8

   ```
   $ sudo yum remove cloudhsm-pkcs11
   ```

2. Client SDK 3 のクライアントデーモンをアンインストールします。

Amazon Linux 2

```
$ sudo yum remove cloudhsm-client
```

CentOS 7

```
$ sudo yum remove cloudhsm-client
```

RHEL 7

```
$ sudo yum remove cloudhsm-client
```

RHEL 8

```
$ sudo yum remove cloudhsm-client
```

> **Note**
>
> カスタム設定を再度有効にする必要があります。

3. 「」の手順に従って、Client SDK PKCS #11 ライブラリをインストールしますClient SDK 5 用の PKCS #11 ライブラリをインストールする。
4. Client SDK 5 では、新しい設定ファイル形式とコマンドラインブートストラップツールが導入されています。Client SDK 5 PKCS #11 ライブラリをブートストラップするには、 のユーザーガイドに記載されている手順に従ってくださいクライアント SDK をブートストラップする。
5. 開発環境で、アプリケーションをテストします。既存のコードを更新して、最終的な移行前に重大な変更を解決します。

関連トピック

- のベストプラクティス AWS CloudHSM

PKCS #11 の詳細設定

AWS CloudHSM PKCS #11 プロバイダーには、ほとんどのお客様が使用する一般的な設定の一部ではない以下の高度な設定が含まれています。これらの設定には追加機能があります。

- PKCS #11 による複数のスロットへの接続
- PKCS #11 の設定を再試行します

PKCS #11 を使用した複数のスロットへの接続

Client SDK 5 PKCS #11 ライブラリ内の 1 つのスロットは、 AWS CloudHSM内のクラスターへの 1 つの接続を表します。Client SDK 5 では、1 つの PKCS #11 アプリケーションから複数のスロットでユーザーを複数の CloudHSM クラスターに接続できるように PKCS11 ライブラリを設定できます。

このトピックで説明されている手順に従って、アプリケーションがマルチスロット機能を使用して複数のクラスターに接続するようにします。

トピック

- マルチスロットの前提条件
- PKCS #11 ライブラリをマルチスロット機能用に設定
- pkcs11 の設定とクラスターの追加
- configure-pkcs11 remove-cluster

マルチスロットの前提条件

- 接続先の 2 つ以上の AWS CloudHSM クラスターとそのクラスター証明書。
- セキュリティグループが上記のすべてのクラスターに接続するように正しく設定された EC2 インスタンス。クラスターとクライアントインスタンスのセットアップ方法の詳細については、「 の開始方法 AWS CloudHSM」を参照してください。
- マルチスロット機能を設定するには、PKCS #11 ライブラリを事前にダウンロードしてインストールしておく必要があります。これをまだ確認していない場合は、「???」の手順を参照してください。

PKCS #11 ライブラリをマルチスロット機能用に設定

PKCS #11 ライブラリをマルチスロット機能用に設定するには、次の手順に従います。

1. マルチスロット機能を使用して接続するクラスターを特定します。
2. ??? の手順に従って、これらのクラスターを PKCS #11 設定に追加します。
3. 次回 PKCS #11 アプリケーションを実行するときには、マルチスロット機能が使用できるようになります。

pkcs11 の設定とクラスターの追加

PKCS #11 を使用して複数のスロットに接続する場合は、configure-pkcs11 add-cluster コマンドを使用してクラスターを設定に追加します。

構文

```
configure-pkcs11 add-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [--region <REGION>]
        [--endpoint <ENDPOINT>]
        [--hsm-ca-cert <HSM CA CERTIFICATE FILE>]
        [--server-client-cert-file <CLIENT CERTIFICATE FILE>]
        [--server-client-key-file <CLIENT KEY FILE>]
        [-h, --help]
```

例

`cluster-id` パラメータを使用してクラスターを追加する

Example

configure-pkcs11 add-cluster とともに `cluster-id` パラメータを使用して、クラスター (`cluster-1234567` の ID) を設定に追加します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 add-cluster --cluster-id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-pkcs11.exe add-cluster --cluster-
id cluster-1234567
```

> **Tip**
>
> configure-pkcs11 add-cluster を `cluster-id` パラメータと一緒に使用してもクラスターが追加されない場合は、以下の例を参照して、追加するクラスターを識別するための `--region` と `--endpoint` パラメータも必要な、より長いバージョンのこのコマンドを参照してください。例えば、クラスターのリージョンが AWS CLI のデフォルトとして設定されているものと異なる場合、適切なリージョンを使用するように `--region` パラメータを使用する必要があります。さらに、呼び出しに使用する AWS CloudHSM API エンドポイントを指定することもできます。これは、 のデフォルトの DNS ホスト名を使用しない VPC インターフェイスエンドポイントを使用するなど、さまざまなネットワーク設定で必要になる場合があります AWS CloudHSM。

cluster-id、**endpoint**、および **region** パラメータを使用してクラスターを追加する

Example

configure-pkcs11 add-cluster とともに `cluster-id`、`endpoint`、`region` のパラメータを使用して、クラスター (`cluster-1234567` の ID) を設定に追加します。

Linux

```
        $ sudo /opt/cloudhsm/bin/configure-pkcs11 add-cluster --cluster-
id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-
east-1.amazonaws.com
```

Windows

```
        C:\Program Files\Amazon\CloudHSM\> .\configure-pkcs11.exe add-cluster --
cluster-id cluster-1234567--region us-east-1 --endpoint https://cloudhsmv2.us-
east-1.amazonaws.com
```

`--cluster-id`、`--region`、`--endpoint` パラメータの詳細については、the section called “パラメータ”を参照してください。

パラメータ

--cluster-id ***<Cluster ID>***

`DescribeClusters` を呼び出して、クラスターIDに関連付けられたクラスターのすべての HSM Elastic Network Interface（ENI）IPアドレスを検索します。システムは ENI IP アドレスを設定 AWS CloudHSM ファイルに追加します。

> **Note**
>
> パブリックインターネットにアクセスできない VPC 内の EC2 インスタンスから `--cluster-id`パラメータを使用する場合は、インターフェイス VPC エンドポイントを作成して に接続する必要があります AWS CloudHSM。VPC エンドポイントの詳細については、「???」を参照してください。

必須：はい

--endpoint ***<Endpoint>***

`DescribeClusters` 呼び出しに使用する AWS CloudHSM API エンドポイントを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

必須：いいえ

--hsm-ca-cert ***<HsmCA ##########***

HSM CA 証明書へのファイルパスを指定します。

必須：いいえ

--region ***<Region>***

クラスターのリージョンを指定します。このオプションは `--cluster-id` と組み合わせて設定する必要があります。

この `--region` パラメータを指定しない場合、システムは `AWS_DEFAULT_REGION` または `AWS_REGION` の環境変数の読み取りを試みてリージョンを選択します。これらの変数が設定さ

れていない場合、環境変数で別のファイルを指定しない限り、AWS Config (通常は ~/.aws/config) のプロファイルに関連付けられたリージョンをチェックしますAWS_CONFIG_FILE。いずれも設定されていない場合は、us-east-1 デフォルトでリージョンが設定されます。

必須：いいえ

--server-client-cert-file ##################

TLS クライアント・サーバー相互認証に使用するクライアント証明書へのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは --server-client-key-file と組み合わせて設定する必要があります。

必須：いいえ

--server-client-key-file ##################

TLS クライアントとサーバーの相互認証に使用されるクライアントキーへのパス。

クライアント SDK 5 に含まれるデフォルトのキーと SSL/TLS 証明書を使用しない場合のみ、このオプションを使用します。このオプションは --server-client-cert-file と組み合わせて設定する必要があります。

必須：いいえ

configure-pkcs11 remove-cluster

PKCS #11 を使用して複数のスロットに接続する場合は、configure-pkcs11 remove-cluster コマンドを使用して使用可能な PKCS#11 スロットからクラスターを削除します。

構文

```
configure-pkcs11 remove-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [-h, --help]
```

例

cluster-id パラメータを使用してクラスターを削除します

Example

configure-pkcs11 remove-cluster とともに cluster-id パラメータを使用して、クラスター (cluster-1234567 の ID) を設定から削除します。

Linux

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 remove-cluster --cluster-
id cluster-1234567
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-pkcs11.exe remove-cluster --cluster-
id cluster-1234567
```

--cluster-id パラメータの詳細については、「the section called “パラメータ”」をご参照ください。

パラメータ

--cluster-id *<Cluster ID>*

設定から削除するクラスターの ID

必須：はい

PKCS #11 の再試行コマンド

Client SDK 5.8.0 以降には、HSM スロットリングされたオペレーションをクライアント側から再試行する自動再試行戦略が組み込まれています。HSM が以前のオペレーションが多すぎてそれ以上リクエストを受け付けられないためにオペレーションをスロットリングすると、Client SDK はスロットリングされたオペレーションを最大 3 回再試行しますが、その間、エクスポネンシャルバック

オフします。この自動再試行戦略は、オフとスタンダードの 2 つのモードのいずれかに設定できます。

- オフ: Client SDK は、HSM によってスロットリングされたオペレーションに対しては再試行戦略を一切実行しません。
- スタンダード: これは Client SDK 5.8.0 以降のデフォルトモードです。このモードでは、Client SDK はエクスポネンシャルバックオフすることで、スロットリングされた操作を自動的に再試行します。

詳細については、「HSM スロットリング」を参照してください。

再試行コマンドをオフモードに設定する

Linux

Linux 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 次のコマンドを使用して再試行設定を off モードに設定できます。

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --default-retry-mode off
```

Windows

Windows 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 次のコマンドを使用して再試行設定を off モードに設定できます。

```
C:\Program Files\Amazon\CloudHSM\bin\ .\configure-pkcs11.exe --default-retry-
mode off
```

OpenSSL Dynamic Engine

AWS CloudHSM OpenSSL Dynamic Engine を使用すると、OpenSSL API を使用して暗号化オペレーションを CloudHSM OpenSSL クラスターにオフロードできます。

AWS CloudHSM は OpenSSL Dynamic Engine を提供します。このエンジンは、 で確認できますLinux 上の SSL/TLS オフロード。OpenSSL AWS CloudHSM で を使用する例については、このAWS セキュリティブログ を参照してください。SDK のプラットフォームサポートトの詳細につい

ては、「the section called “サポートされているプラットフォーム”」を参照してください。トラブルシューティングについては、「」を参照してくださいOpenSSL Dynamic Engine SDK の既知の問題。

Client SDK 3 の使用の詳細については、「以前のクライアント SDK (クライアント SDK 3)」を参照してください。

詳細については、以下のトピックを参照してください。

トピック

- OpenSSL Dynamic Engine のインストール
- OpenSSL Dynamic Engine キータイプ
- OpenSSL Dynamic Engine のメカニズム
- OpenSSL 動的エンジンをクライアント SDK 3 からクライアント SDK 5 に移行する
- OpenSSL の詳細設定

OpenSSL Dynamic Engine のインストール

> **Note**
>
> クライアント SDK 5 で単一の HSM クラスターを実行するには、まず `disable_key_availability_check` を `True` に設定してクライアントキーの耐久性の設定を管理する必要があります。詳細については、キーの同期 と クライアント SDK 5 設定ツール を参照してください。

OpenSSL Dynamic Engine をインストールして設定するには

1. 以下のコマンドを使用して、OpenSSL エンジンをダウンロードしてインストールします。

   Amazon Linux 2

   Amazon Linux 2 用の OpenSSL Dynamic Engine を x86_64 アーキテクチャにインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-dyn-latest.el7.x86_64.rpm
   ```

```
$ sudo yum install ./cloudhsm-dyn-latest.el7.x86_64.rpm
```

Amazon Linux 2 用の OpenSSL Dynamic Engine を ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-dyn-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el7.aarch64.rpm
```

Amazon Linux 2023

Amazon Linux 2023 用 OpenSSL Dynamic Engine を x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-dyn-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.amzn2023.x86_64.rpm
```

Amazon Linux 2023 用 OpenSSL Dynamic Engine を ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-dyn-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.amzn2023.aarch64.rpm
```

CentOS 7 (7.8+)

CentOS 7 用の OpenSSL Dynamic Engine を x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el7.x86_64.rpm
```

RHEL 7 (7.8+)

x86_64 アーキテクチャに RHEL 7 用の OpenSSL Dynamic Engine をインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

x86_64 アーキテクチャに RHEL 8 用の OpenSSL Dynamic Engine をインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-dyn-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el8.x86_64.rpm
```

RHEL 9 (9.2+)

x86_64 アーキテクチャに RHEL 9 用の OpenSSL Dynamic Engine をインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-dyn-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el9.x86_64.rpm
```

OpenSSL Dynamic Engine for RHEL 9 を ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-dyn-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el9.aarch64.rpm
```

Ubuntu 20.04 LTS

OpenSSL Dynamic Engine for Ubuntu 20.04 LTS を x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/
cloudhsm-dyn_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u20.04_amd64.deb
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine for Ubuntu 22.04 LTS を x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-dyn_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u22.04_amd64.deb
```

OpenSSL Dynamic Engine for Ubuntu 22.04 LTS を ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-dyn_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u22.04_arm64.deb
```

ダイナミックエンジン用の共有ライブラリが `/opt/cloudhsm/lib/libcloudhsm_openssl_engine.so` にインストールされました。

2. Client SDK 5 をブートストラップします。ブートストラップの詳細については、クライアント SDK をブートストラップする を参照してください。
3. Crypto User (CU) の認証情報を使用して環境変数を設定します。CU の詳細については、「CMU を使用したユーザーの管理」を参照してください。

```
$ export CLOUDHSM_PIN=<HSM user name>:<password>
```

> **Note**
>
> クライアント SDK 5 では CU の認証情報を保存するための CLOUDHSM_PIN 環境変数が導入されています。Client SDK 3 では、CU の認証情報を n3fips_password 環境変数に保存します。クライアント SDK 5 は両方の環境変数をサポートしますが、CLOUDHSM_PIN を使用することを推奨します。

4. OpenSSL Dynamic Engine のインストールをクラスターに接続します。詳細については、クラスターへの接続 を参照してください。
5. Client SDK 5 をブートストラップします。詳細については、「the section called “クライアント SDK をブートストラップする”」を参照してください。

クライアント SDK 5 用 [OpenSSL 動的エンジン] を確認します。

次のコマンドを使用して [OpenSSL 動的エンジン] のインストールを確認します。

```
$ openssl engine -t cloudhsm
```

次の出力により設定が検証されます。

```
(cloudhsm) CloudHSM OpenSSL Engine
     [ available ]
```

OpenSSL Dynamic Engine キータイプ

AWS CloudHSM OpenSSL Dynamic Engine では、次のキータイプがサポートされています。

キータイプ	説明
EC	P-256、P-384、および secp256k1 キータイプの ECDSA 署名/検証。OpenSSL エンジンと相互運用可能な EC キーを生成するには、key generate-file を参照してください。

キータイプ	説明
RSA	2048、3072、および 4096 ビットキーの RSA キー生成。RSA 署名/検証。検証は OpenSSL ソフトウェアにオフロードされます。

OpenSSL Dynamic Engine のメカニズム

AWS CloudHSM OpenSSL Dynamic Engine メカニズムを使用する方法について説明します。

署名および検証

AWS CloudHSM OpenSSL Dynamic Engine では、署名および検証関数に次のメカニズムを使用できます。

Client SDK 5 では、データはソフトウェアでローカルにハッシュされます。つまり、ハッシュできるデータのサイズに制限はありません。

RSA 署名タイプ

- SHA1withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

ECDSA 署名タイプ

- SHA1 (ECDSA 搭載)
- SHA224 (ECDSA 搭載)
- SHA256 (ECDSA 搭載)
- SHA384 (ECDSA 搭載)
- SHA512 (ECDSA 搭載)

OpenSSL 動的エンジンをクライアント SDK 3 からクライアント SDK 5 に移行する

このトピックでは、OpenSSL Dynamic Engine を Client SDK 3 から Client SDK 5 に移行します。移行のメリットについては、「」を参照してくださいClient SDK 5 の利点。

では AWS CloudHSM、顧客アプリケーションは AWS CloudHSM クライアントソフトウェア開発キット (SDK) を使用して暗号化オペレーションを実行します。Client SDK 5 は、引き続き新機能とプラットフォームサポートが追加されているプライマリ SDK です。

> **Note**
>
> OpenSSL Dynamic Engine を使用した Client SDK 5 では、乱数生成は現在サポートされていません。

すべてのプロバイダーの移行手順を確認するには、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。

Client SDK 5 への移行

このセクションの手順に従って、Client SDK 3 から Client SDK 5 に移行します。

> **Note**
>
> Amazon Linux、Ubuntu 16.04、Ubuntu 18.04、CentOS 6、CentOS 8、および RHEL 6 は現在、クライアント SDK 5 ではサポートされていません。現在、クライアント SDK 3 でこれらのプラットフォームのいずれかを使用している場合は、クライアント SDK 5 に移行するときに別のプラットフォームを選択する必要があります。

1. クライアント SDK 3 用の OpenSSL Dynamic Engine をアンインストールします。

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-dyn
   ```

CentOS 7

```
$ sudo yum remove cloudhsm-dyn
```

RHEL 7

```
$ sudo yum remove cloudhsm-dyn
```

RHEL 8

```
$ sudo yum remove cloudhsm-dyn
```

2. Client SDK 3 のクライアントデーモンをアンインストールします。

Amazon Linux 2

```
$ sudo yum remove cloudhsm-client
```

CentOS 7

```
$ sudo yum remove cloudhsm-client
```

RHEL 7

```
$ sudo yum remove cloudhsm-client
```

RHEL 8

```
$ sudo yum remove cloudhsm-client
```

> **Note**
>
> カスタム設定を再度有効にする必要があります。

3. 「」の手順に従って、Client SDK OpenSSL Dynamic Engine をインストールしますOpenSSL Dynamic Engine のインストール。

4. Client SDK 5 では、新しい設定ファイル形式とコマンドラインブートストラップツールが導入されています。Client SDK 5 OpenSSL 動的エンジンをブートストラップするには、 のユーザーガイドに記載されている手順に従ってくださいクライアント SDK をブートストラップする。
5. 開発環境で、アプリケーションをテストします。既存のコードを更新して、最終的な移行前に重大な変更を解決します。

関連トピック

- のベストプラクティス AWS CloudHSM

OpenSSL の詳細設定

AWS CloudHSM OpenSSL プロバイダーには、次の詳細設定が含まれていますが、これはほとんどのお客様が使用する一般的な設定の一部ではありません。これらの設定には追加機能があります。

- OpenSSL の再試行コマンド

OpenSSL の再試行コマンド

Client SDK 5.8.0 以降には、HSM スロットリングされたオペレーションをクライアント側から再試行する自動再試行戦略が組み込まれています。HSM が以前のオペレーションが多すぎてそれ以上リクエストを受け付けられないためにオペレーションをスロットリングすると、Client SDK はスロットリングされたオペレーションを最大 3 回再試行しますが、その間、エクスポネンシャルバックオフします。この自動再試行戦略は、オフとスタンダードの 2 つのモードのいずれかに設定できます。

- オフ: Client SDK は、HSM によってスロットリングされたオペレーションに対しては再試行戦略を一切実行しません。
- スタンダード: これは Client SDK 5.8.0 以降のデフォルトモードです。このモードでは、Client SDK はエクスポネンシャルバックオフすることで、スロットリングされた操作を自動的に再試行します。

詳細については、「HSM スロットリング」を参照してください。

再試行コマンドをオフモードに設定する

Linux

Linux 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 再試行コマンドを off モードに設定するには以下のコマンドを使用できます。

```
$ sudo /opt/cloudhsm/bin/configure-dyn --default-retry-mode off
```

Windows

Windows 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 再試行コマンドを off モードに設定するには以下のコマンドを使用できます。

```
C:\Program Files\Amazon\CloudHSM\bin\ .\configure-dyn.exe --default-retry-mode
 off
```

JCE プロバイダー

AWS CloudHSM JCE プロバイダーは、Java 暗号化拡張 (JCE) プロバイダーフレームワークから構築されたプロバイダー実装です。JCE では、Java 開発キット (JDK) を使用して暗号化操作を実行できます。このガイドでは、 AWS CloudHSM JCE プロバイダーは JCE プロバイダーと呼ばれることがあります。JCE プロバイダーと JDK を使用して、HSM に暗号化操作をオフロードします。トラブルシューティングについては、「」を参照してくださいJCE SDK の既知の問題。

Client SDK 3 の使用の詳細については、「以前のクライアント SDK (クライアント SDK 3)」を参照してください。

トピック

- Client SDK 5 の AWS CloudHSM JCE プロバイダーをインストールして使用する
- JCE プロバイダーでサポートされているキータイプ
- JCE プロバイダーでサポートされているメカニズム
- サポートされている Java キー属性
- Java 用 AWS CloudHSM ソフトウェアライブラリのコードサンプル
- AWS CloudHSM JCE プロバイダーの Javadocs

- AWS CloudHSM KeyStore Java クラスの使用
- JCE プロバイダーをクライアント SDK 3 からクライアント SDK 5 に移行する
- JCE の詳細設定

Client SDK 5 の AWS CloudHSM JCE プロバイダーをインストールして使用する

JCE プロバイダーは、OpenJDK 8、OpenJDK 11、OpenJDK 17、および OpenJDK 21 と互換性があります。どちらも OpenJDK のウェブサイトからダウンロードできます。

> **Note**
>
> クライアント SDK 5 で単一の HSM クラスターを実行するには、まず `disable_key_availability_check` を `True` に設定してクライアントキーの耐久性の設定を管理する必要があります。詳細については、キーの同期 と クライアント SDK 5 設定ツール を参照してください。

トピック

- JCE プロバイダーをインストールする
- JCE プロバイダーに認証情報を提供する
- JCE プロバイダー中のキー管理の基本

JCE プロバイダーをインストールする

1. 以下のコマンドを使用して、JCE プロバイダーをダウンロードし,インストールします。

   Amazon Linux 2

   Amazon Linux 2 用の JCE プロバイダーを x86_64 アーキテクチャにインストールします。

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-jce-latest.el7.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-jce-latest.el7.x86_64.rpm
   ```

Amazon Linux 2 用の JCE プロバイダーを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-jce-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el7.aarch64.rpm
```

Amazon Linux 2023

Amazon Linux 2023 用の JCE プロバイダーを x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-jce-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.amzn2023.x86_64.rpm
```

Amazon Linux 2023 用の JCE プロバイダーを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-jce-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.amzn2023.aarch64.rpm
```

CentOS 7 (7.8+)

CentOS 7 用の JCE プロバイダーを x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el7.x86_64.rpm
```

RHEL 7 (7.8+)

RHEL 7 用の JCE プロバイダーを x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

RHEL 8 用の JCE プロバイダーを x86_64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el8.x86_64.rpm
```

RHEL 9 (9.2+)

x86_64 アーキテクチャに RHEL 9 (9.2+) 用の JCE プロバイダーをインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-jce-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el9.x86_64.rpm
```

RHEL 9 (9.2+) 用の JCE プロバイダーを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-jce-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el9.aarch64.rpm
```

Ubuntu 20.04 LTS

x86_64 アーキテクチャに Ubuntu 20.04 LTS 用の JCE プロバイダーをインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/
cloudhsm-jce_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u20.04_amd64.deb
```

Ubuntu 22.04 LTS

x86_64 アーキテクチャに Ubuntu 22.04 LTS 用の JCE プロバイダーをインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-jce_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u22.04_amd64.deb
```

Ubuntu 22.04 LTS 用の JCE プロバイダーを ARM64 アーキテクチャにインストールします。

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-jce_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u22.04_arm64.deb
```

Windows Server 2016

x86_64 アーキテクチャに Windows Server 2016 用の JCE プロバイダーをインストールし、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/
Windows/AWSCloudHSMJCE-latest.msi -Outfile C:\AWSCloudHSMJCE-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMJCE-
latest.msi /quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2019

x86_64 アーキテクチャに Windows Server 2019 用の JCE プロバイダーをインストールし、管理者 PowerShell として を開き、次のコマンドを実行します。

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/
Windows/AWSCloudHSMJCE-latest.msi -Outfile C:\AWSCloudHSMJCE-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMJCE-
latest.msi /quiet /norestart /log C:\client-install.txt' -Wait
```

2. Client SDK 5 をブートストラップします。ブートストラップの詳細については、クライアント SDK をブートストラップする を参照してください。
3. 次の JCE プロバイダーファイルを見つけます。

   Linux

   - `/opt/cloudhsm/java/cloudhsm-version.jar`
   - `/opt/cloudhsm/bin/configure-jce`
   - `/opt/cloudhsm/bin/jce-info`

   Windows

   - `C:\Program Files\Amazon\CloudHSM\java\cloudhsm-version.jar>`
   - `C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe`
   - `C:\Program Files\Amazon\CloudHSM\bin\jce_info.exe`

JCE プロバイダーに認証情報を提供する

HSM では、Java アプリケーションが HSM を使用する前に、Java アプリケーションを認証する必要があります。HSM は、明示的なログインと暗黙的なログイン方法のいずれかを使用して認証します。

明示的なログイン - この方法では、 AWS CloudHSM 認証情報をアプリケーションに直接渡すことができます。CU ユーザー名とパスワードを PIN パターンで渡す `AuthProvider` を使用します。詳細については、「Login to an HSM」のコード例を参照してください。

暗黙的なログイン - この方法では、 AWS CloudHSM 認証情報を、新しいプロパティファイルまたはシステムプロパティで設定するか、環境変数として設定することができます。

- System properties - アプリケーションの実行時に、システムプロパティを通して認証情報を設定します。次の例は、これを行うための 2 つの異なる方法を示しています。

  Linux

  ```
  $ java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
  ```

  ```
  System.setProperty("HSM_USER","<HSM user name>");
  System.setProperty("HSM_PASSWORD","<password>");
  ```

  Windows

  ```
  PS C:\> java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
  ```

  ```
  System.setProperty("HSM_USER","<HSM user name>");
  System.setProperty("HSM_PASSWORD","<password>");
  ```

- Environment variables - 認証情報を環境変数として設定します。

  Linux

  ```
  $ export HSM_USER=<HSM user name>
  $ export HSM_PASSWORD=<password>
  ```

  Windows

  ```
  PS C:\> $Env:HSM_USER="<HSM user name>"
  PS C:\> $Env:HSM_PASSWORD="<password>"
  ```

アプリケーションで設定されない場合、または HSM でセッションを認証する前にユーザーが操作を行った場合は、認証情報を使用できない場合があります。このような場合は、Java 用の CloudHSM ソフトウェアライブラリによって、次の順序で認証情報が検索されます。

1. システムプロパティ
2. 環境変数

JCE プロバイダー中のキー管理の基本

JCE プロバイダー中のキー管理の基本には、キーのインポート、キーのエクスポート、ハンドルによるキーのロード、またはキーの削除などがあります。キーの管理の詳細については、「キーの管理」のサンプルコードを参照してください。

また、JCE プロバイダーののサンプルコードについては、コードサンプル で参照できます。

JCE プロバイダーでサポートされているキータイプ

Java 用の AWS CloudHSM ソフトウェアライブラリでは、次のキータイプを生成できます。

キータイプ	説明
AES	128、192、256 ビットの AES キーを生成します。
Triple DES (3DES、DESede)	192 ビットトリプル DES キーを生成する[1]。今後の変更の脚注を参照してください。
EC	EC キーペア — NIST 曲線 secp224r1 (P-224)、secp256r1 (P-256)、secp256k1 (ブロックチェーン)、secp384r1 (P-384)、secp521r1 (P-521) を生成します。
[GENERIC_SECRET]	1 ~ 800 バイトの汎用シークレットを生成します。
HMAC	SHA1、SHA224、SHA256、SHA384、SHA512 のハッシュサポート。
RSA	256 ビットの増分で、2048 ~ 4096 ビットの RSA キーを生成します。

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

JCE プロバイダーでサポートされているメカニズム

このトピックでは、クライアント SDK 5 で JCE プロバイダーがサポートするメカニズムについて説明します。でサポートされている Java 暗号化アーキテクチャ (JCA) インターフェイスとエンジンクラスについては AWS CloudHSM、以下のトピックを参照してください。

トピック

キーとキーペアの関数を生成する

Java 用の AWS CloudHSM ソフトウェアライブラリでは、次のオペレーションを使用してキーとキーペア関数を生成できます。

- `RSA`
- `EC`
- `AES`
- `DESede (Triple DES)`[1]
- `GenericSecret`

暗号関数

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のアルゴリズム、モード、パディングの組み合わせをサポートしています。

アルゴリズム	モード	[Padding] (パディング)	メモ
AES	CBC	AES/CBC/NoPadding AES/CBC/PKCS5Padding	Cipher.ENCRYPT_MODE および Cipher.DECRYPT_MODE を実装します。 Cipher.UNWRAP_MODE for AES/CBC NoPadding を実装します
AES	ECB	AES/ECB/PKCS5Padding AES/ECB/NoPadding	Cipher.ENCRYPT_MODE および Cipher.DECRYPT_MODE を実装します。
AES	CTR	AES/CTR/NoPadding	Cipher.ENCRYPT_MODE および Cipher.DECRYPT_MODE を実装します。
AES	GCM	AES/GCM/NoPadding	Cipher.WRAP_MODE 、Cipher.UNWRAP_MODE 、Cipher.ENCRYPT_MODE 、および Cipher.DECRYPT_MODE を実装します。 AES-GCM 暗号化の実行時に、HSM はリ

アルゴリズム	モード	[Padding] (パディング)	メモ
			クエスト内の初期化ベクトル (IV) を無視し、独自に IV を生成して使用します。オペレーションが完了したら、`Cipher.getIV()` を呼び出して IV を取得する必要があります。

AESWrap	ECB	`AESWrap/ECB/NoPadding`
		`AESWrap/ECB/PKCS5Padding`
		`AESWrap/ECB/ZeroPadding`
			`Cipher.WRAP_MODE` および `Cipher.UNWRAP_MODE` を実装します。

DESede (Triple DES)	CBC	`DESede/CBC/PKCS5Padding` `DESede/CBC/NoPadding`	`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。今後の変更については、以下の注記「1」を参照してください。

アルゴリズム	モード	[Padding] (パディング)	メモ

DESede (Triple DES)	ECB	`DESede/ECB/NoPadding`
		
		`DESede/ECB/PKCS5Padding`
			`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。今後の変更については、以下の注記「1」を参照してください。

アルゴリズム	モード	[Padding] (パディング)	メモ
RSA	ECB	RSA/ECB/PKCS1Padding 注記「1」を参照 RSA/ECB/OAEPPadding RSA/ECB/OAEPWithSHA-1ANDMGF1Padding RSA/ECB/OAEPWithSHA-224ANDMGF1Padding RSA/ECB/OAEPWithSHA-256ANDMGF1Padding RSA/ECB/OAEPWithSHA-384ANDMGF1Padding RSA/ECB/OAEPWithSHA-512ANDMGF1Padding	Cipher.WRAP_MODE 、Cipher.UNWRAP_MODE 、Cipher.ENCRYPT_MODE 、および Cipher.DECRYPT_MODE を実装します。

アルゴリズム	モード	[Padding] (パディング)	メモ
RSA	ECB	RSA/ECB/NoPadding	Cipher.ENCRYPT_MODE および Cipher.DECRYPT_MODE を実装します。
RSAAESWrap	ECB	RSAAESWrap/ECB/OAEPPadding RSAAESWrap/ECB/OAEPWithSHA-1ANDMGF1Padding RSAAESWrap/ECB/OAEPWithSHA-224ANDMGF1Padding RSAAESWrap/ECB/OAEPWithSHA-256ANDMGF1Padding RSAAESWrap/ECB/OAEPWithSHA-384ANDMGF1Padding RSAAESWrap/ECB/OAEPWithSHA-512ANDMGF1Padding	Cipher.WRAP_MODE および Cipher.UNWRAP_MODE を実装します。

署名および検証

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のタイプの署名と検証をサポートしています。Client SDK 5 とハッシュ機能付きの署名アルゴリズムでは、データはソフトウェアでローカルにハッシュされてから、署名/検証のために HSM に送信されます。つまり、SDK でハッシュできるデータのサイズに制限はありません。

RSA 署名タイプ

- NONEwithRSA
- RSASSA-PSS
- SHA1withRSA
- SHA1withRSA/PSS
- SHA1withRSAandMGF1
- SHA224withRSA
- SHA224withRSAandMGF1
- SHA224withRSA/PSS
- SHA256withRSA
- SHA256withRSAandMGF1
- SHA256withRSA/PSS
- SHA384withRSA
- SHA384withRSAandMGF1
- SHA384withRSA/PSS
- SHA512withRSA
- SHA512withRSAandMGF1
- SHA512withRSA/PSS

ECDSA 署名タイプ

- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA

- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA

ダイジェスト関数

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のメッセージダイジェストをサポートしています。Client SDK 5 では、データはソフトウェアでローカルにハッシュされます。つまり、SDK でハッシュできるデータのサイズに制限はありません。

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

Hash-based Message Authentication Code (HMAC) 関数

Java 用の AWS CloudHSM ソフトウェアライブラリは、次の HMAC アルゴリズムをサポートしています。

- HmacSHA1 (最大データサイズ (バイト): 16288)
- HmacSHA224 (最大データサイズ (バイト): 16256)
- HmacSHA256 (最大データサイズ (バイト): 16288)
- HmacSHA384 (最大データサイズ (バイト): 16224)
- HmacSHA512 (最大データサイズ (バイト): 16224)

暗号ベースのメッセージ認証コード (CMAC) 関数

CMAC (暗号ベースのメッセージ認証コード) は、ブロック暗号とシークレットキーを使用してメッセージ認証コード (MAC) を作成します。HMAC とは異なり、MAC にはハッシュ方式ではなくブロック対称キーメソッドを使用します。

Java 用の AWS CloudHSM ソフトウェアライブラリは、次の CMAC アルゴリズムをサポートしています。

- AESCMAC

キーファクトリを使用してキーをキー仕様に変換します

キーファクトリを使用してキーをキー仕様に変換できます。 AWS CloudHSM には JCE 用の 2 種類のキーファクトリがあります。

SecretKeyFactory： 対称キーをインポートまたは取得するために使用されます。を使用すると SecretKeyFactory、サポートされているキーまたはサポートされている KeySpec を渡して、対称キーを にインポートまたは取得できます AWS CloudHSM。でサポートされている仕様は次のとおりです KeyFactory。

- SecretKeyFactoryの `generateSecret`メソッドでは、次のKeySpecクラスがサポートされています。
  - KeyAttributesMapを使用して、追加属性を持つキーバイトを CloudHSM キーとしてインポートできます。例はこちらからご覧いただけます。
  - SecretKeySpecを使用して、対称キー仕様を CloudHSM キーとしてインポートできます。
  - AesCmacKdfParameterSpecは、別の CloudHSM AES キーを使用して対称キーを導出するために使用できます。

> **Note**
>
> SecretKeyFactoryの `translateKey`メソッドは、キーインターフェイスを実装するすべてのキーを取得します。

KeyFactory： 非対称キーのインポートに使用されます。を使用すると KeyFactory、サポートされているキーを渡すか、サポートされているキー KeySpec を にインポートできます AWS CloudHSM。詳細については、以下のリソースを参照してください。

- KeyFactoryの `generatePublic`メソッドでは、次のKeySpecクラスがサポートされています。
- RSA と EC の両方の CloudHSM KeyAttributesMap KeyTypes。以下が含まれます。
  - RSA と EC パブリック の両方の CloudHSM KeyAttributesMap KeyTypes。例はこちらからご覧いただけます
  - RSA と EC パブリックキーの両方の X509EncodedKeySpec
  - RSAPublicKeySpec for RSA パブリックキー

  - ECPublicKeySpec パブリックキーの EC
- KeyFactoryの `generatePrivate`メソッドでは、次のKeySpecクラスがサポートされています。
- RSA と EC の両方の CloudHSM KeyAttributesMap KeyTypes。以下が含まれます。
  - RSA と EC パブリック の両方の CloudHSM KeyAttributesMap KeyTypes。例はこちらからご覧いただけます
  - EC プライベートキーと RSA プライベートキーの両方の PKCS8EncodedKeySpec
  - RSAPrivateCrtKeySpec for RSA プライベートキー
  - ECPrivateKeySpec プライベートキーの EC

KeyFactoryの `translateKey`メソッドでは、キーインターフェイス を実装する任意のキーを取り込みます。

メカニズムの注釈

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

サポートされている Java キー属性

このトピックでは、Client SDK 5 でサポートされている Java キー属性について説明します。このトピックでは、JCE プロバイダーの独自の拡張機能を使用してキーの属性を設定する方法について説明します。この拡張機能を使用して、これらのオペレーション中にサポートされるキー属性とその値を設定します。

- キー生成
- キーのインポート

キーアトリビュートの使用方法の例については、「the section called “コードサンプル”」を参照してください。

トピック

- 属性について
- サポートされている属性
- キーの属性設定

属性について

キー属性を使用して、パブリックキー、プライベートキー、シークレットキーなど、キーオブジェクトで許可されるアクションを指定します。キー属性と値は、キーオブジェクトの作成オペレーション中に定義されます。

Java Cryptography Extension (JCE) では、キー属性に値を設定する方法が指定されていないため、ほとんどのアクションがデフォルトで許可されていました。これに対して、PKCS # 11 標準では、より制限の厳しいデフォルトのある包括的な属性のセットが定義されています。JCE プロバイダー 3.1 以降、 は一般的に使用される属性に対してより制限の厳しい値を設定できる独自の拡張機能 AWS CloudHSM を提供します。

サポートされている属性

次の表に示す属性の値を設定できます。ベストプラクティスとして、制限する属性の値のみを設定してください。値を指定しない場合、 は以下の表で指定されたデフォルト値 AWS CloudHSM を使用します。デフォルト値の列のセルが空の場合は、属性に割り当てられている特定のデフォルト値がないことを示します。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
DECRYPT	TRUE		TRUE	True は、キーを使用して任意のバッファを復号できることを示します。通常、WRAP「true」に設定されているキーに対しては、これを「FALSE」に設定します。
DERIVE				キーを使用して他のキーを派生

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
				させることができます。
ENCRYPT	TRUE	TRUE		True は、キーを使用して任意のバッファを暗号化できることを示します。
EXTRACTABLE	TRUE		TRUE	True は、このキーを HSM からエクスポートできることを示します。
ID				キーを識別するために使用されるユーザー定義の値。
KEY_TYPE				キーのタイプ (AES、DESede、ジェネリックシークレット、EC、RSA) を識別するために使用されます。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
LABEL				HSM 上のキーを簡単に識別できるユーザー定義の文字列。ベストプラクティスに従うには、後で見つけやすいように、各キーに固有のラベルを付けてください。
LOCAL				HSM によって生成されたキーを示します。
OBJECT_CLASS				キー (SecretKey、 PublicKey または) のオブジェクトクラスを識別するために使用されます PrivateKey。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
PRIVATE	TRUE	TRUE	TRUE	True は、ユーザーが認証されるまで、ユーザーがキーにアクセスできないことを示します。わかりやすくするために、この属性が FALSE に設定されている場合でも、ユーザーは認証 AWS CloudHSM されるまで のキーにアクセスできません。
SIGN	TRUE		TRUE	True は、キーを使用してメッセージダイジェストに署名できることを示します。パブリックキーおよびアーカイブしたプライベートキーの場合、通常 FALSE に設定されます。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
SIZE				キーのサイズを定義する属性。サポートされているキーサイズの詳細については、「Client SDK 5 でサポートされているメカニズム」を参照してください。
TOKEN	FALSE	FALSE	FALSE	クラスター内のすべての HSM にレプリケートされ、バックアップに含まれる永続的なキー。TOKEN = FALSE は、HSM への接続が切断されるかログアウトされると自動的に消去される一時キーを意味します。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
UNWRAP	TRUE		TRUE	True は、キーを使用して別のキーをラップ解除 (インポート) できることを示します。
VERIFY	TRUE	TRUE		True は、キーを使用して署名を検証できることを示します。これは通常、プライベートキーの場合、FALSE に設定されます。
WRAP	TRUE	TRUE		True は、キーを使用して別のキーをラップできることを示します。通常、プライベートキーの場合、これを FALSE に設定します。

属性	デフォルト値			メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	
WRAP_WITH_TRUSTED	FALSE		FALSE	True は、TRUSTED 属性が true に設定されているキーでのみキーのラップとラップ解除ができることを示します。キーが true に WRAP_WITH_TRUSTED 設定されると、その属性は読み取り専用になり、false に設定できなくなります。トラストラッピングについて詳しくは、「信頼できるキーを使ってキーのアンラップを制御する」をご覧ください。

> **Note**
>
> PKCS #11 ライブラリでは、より広範な属性がサポートされます。詳細については、「サポートされている PKCS #11 属性」を参照してください。

キーの属性設定

`KeyAttributesMap` は Java Map のようなオブジェクトで、キーオブジェクトの属性値を設定するために使用できます。`KeyAttributesMap` 関数のメソッドは、Java マップ操作のメソッドと同様です。

属性にカスタム値を設定するには、次の 2 つのオプションがあります。

- 次の表に示す方法を使用します。
- このドキュメントの後半で説明するビルダーパターンの使用

属性マップオブジェクトは、属性を設定するための次のメソッドをサポートしています。

操作	戻り値	**`KeyAttributesMap`** 方法
既存のキーのキー属性の値を取得する	オブジェクト (値を含む) または null	get(keyAttribute)
1 つのキー属性の値を入力します。	キー属性のマッピングがなかった場合、キー属性に関連付けられた以前の値、または null	put(keyAttribute, value)
複数のキー属性の値を設定する	該当なし	putAll (keyAttributesMap)
属性マップからキーと値のペアを削除する	キー属性のマッピングがなかった場合、キー属性に関連付けられた以前の値、または null	remove(keyAttribute)

> **Note**
>
> 明示的に指定しない属性は、上記の the section called “サポートされている属性” の表に示したデフォルトに設定されます。

キーペアの属性設定

Java クラス KeyPairAttributesMap を使用して、キーペアのキー属性を処理します。KeyPairAttributesMap は、2 つの KeyAttributesMap オブジェクトをカプセル化します。1 つはパブリックキー用ともう 1 つはプライベートキー用です。

パブリックキーとプライベートキーの個々の属性を個別に設定するには、そのキーの対応する KeyAttributes マップオブジェクトで put() メソッドを使用できます。getPublic() メソッドを使用してパブリックキーの属性マップを取得し、getPrivate() を使用してプライベートキーの属性マップを取得します。引数としてキーペア属性マップを使用する putAll() を使用して、パブリックキーペアとプライベートキーペアの両方に、複数のキー属性の値を一緒に入力します。

Java 用 AWS CloudHSM ソフトウェアライブラリのコードサンプル

このトピックでは、Client SDK 5 の Java コードサンプルに関するリソースと情報を提供します。

前提条件

サンプルを実行する前に、環境をセットアップする必要があります。

- Java Cryptographic Extension (JCE) provider をインストールして設定します。
- 有効な HSM ユーザー名とパスワードを設定します。これらのタスクには、暗号化ユーザー (CU) のアクセス権限で十分です。アプリケーションは、それぞれの例でこの認証情報を使用して HSM にログインします。
- JCE provider へのクレデンシャルを提供する方法を決定します。

コードサンプル

次のコードサンプルでは、基本タスクを実行するために、AWS CloudHSM JCE provider を使用する方法を示します。その他のコードサンプルは、 で入手できますGitHub。

- HSM へのログイン
- キーの管理
- 対称キーの生成
- 非対称キーの生成
- AES GCM による暗号化と復号
- Encrypt and decrypt with AES-CTR

- DESede-ECB による暗号化と復号化 注記 1 参照
- Sign and Verify with RSA Keys
- Sign and Verify with EC Keys
- サポートされているキー属性の使用
- CloudHSM キーストアの使用

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

AWS CloudHSM JCE プロバイダーの Javadocs

JCE プロバイダーの Javadocs を使用して、AWS CloudHSM JCE SDK で定義されている Java タイプとメソッドに関する使用情報を取得します。の最新の Javadocs をダウンロードするには AWS CloudHSM、ダウンロードページの 最新リリースセクションを参照してください。

Javadocs は統合開発環境 (IDE) にインポートしたり、ウェブブラウザで表示することができます。

AWS CloudHSM KeyStore Java クラスの使用

クラスは AWS CloudHSM `KeyStore`、専用 PKCS12 キーストアを提供します。このキーストアでは、証明書をキーデータとともに保存し、 AWS CloudHSMに保存されているキーデータに関連付けることができます。`KeyStore` クラスは、Java AWS CloudHSM Cryptography Extension (JCE) の`KeyStore`サービスプロバイダーインターフェイス (SPI) を実装します。の使用の詳細については`KeyStore`、「クラスKeyStore」を参照してください。

> **Note**
>
> 証明書は公開情報であり、暗号化キーのストレージ容量を最大化するため、 AWS CloudHSM は HSMsへの証明書の保存をサポートしていません。

適切なキーストアの選択

AWS CloudHSM Java Cryptographic Extension (JCE) プロバイダーは、専用の AWS CloudHSM を提供しています KeyStore。クラスは、HSM AWS CloudHSM `KeyStore`へのキーオペレーションのオ

フロード、証明書のローカルストレージ、および証明書ベースのオペレーションをサポートしています。

専用 CloudHSM KeyStore を次のようにロードします。

```
KeyStore ks = KeyStore.getInstance("CloudHSM")
```

初期化中 AWS CloudHSM KeyStore

JCE プロバイダーにログインするの AWS CloudHSM KeyStore と同じ方法で にログインします。環境変数またはシステムプロパティファイルのいずれかを使用できます。CloudHSM KeyStoreの使用を開始する前にログインする必要があります。JCE プロバイダーを使用して HSM にログインする例については、Login to an HSM を参照してください。

必要に応じて、パスワードを指定して、キーストアデータを保持するローカル PKCS12 ファイルを暗号化できます。 AWS CloudHSM キーストアを作成するときは、ロード、設定、取得の方法を使用するときにパスワードを設定し、指定します。

次のように新しい CloudHSM KeyStore オブジェクトをインスタンス化します。

```
ks.load(null, null);
```

`store` メソッドを使用して、キーストアデータをファイルに書き込みます。その後は、次のように、ソースファイルとパスワードを使用し、`load` メソッドを使用して既存のキーストアをロードできます。

```
ks.load(inputStream, password);
```

の使用 AWS CloudHSM KeyStore

AWS CloudHSM KeyStore は JCE クラスKeyStore仕様に準拠しており、以下の機能を提供します。

- `load`

  指定された入力ストリームからキーストアをロードします。キーストアの保存時にパスワードが設定されている場合、ロードを成功させるには、この同じパスワードを指定する必要があります。新しい空のキーストアを初期化するには、両方のパラメータを null に設定します。

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  ks.load(inputStream, password);
  ```

- `aliases`

  指定されたキーストアインスタンス内に含まれるすべてのエントリのエイリアス名の列挙を返します。結果には、PKCS12 ファイルにローカルに保存されたオブジェクトと、HSM 上に存在するオブジェクトが含まれます。

  サンプルコード:

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
for(Enumeration<String> entry = ks.aliases(); entry.hasMoreElements();) {
    String label = entry.nextElement();
    System.out.println(label);
}
```

- `containsalias`

  キーストアが、指定されたエイリアスを持つ少なくとも 1 つのオブジェクトにアクセスできる場合は true を返します。キーストアは、PKCS12 ファイルにローカルに保存されているオブジェクトと、HSM 上に存在するオブジェクトをチェックします。

- `deleteEntry`

  ローカル PKCS12 ファイルから証明書エントリを削除します。HSM に保存されているキーデータの削除は、 ではサポートされていません AWS CloudHSM KeyStore。Destroyable インターフェースの `destroy` メソッドを使用してキーを削除できます。

```
((Destroyable) key).destroy();
```

- `getCertificate`

  使用可能な場合、エイリアスに関連付けられた証明書を返します。エイリアスが存在しないか、証明書ではないオブジェクトを参照している場合、関数は NULL を返します。

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
Certificate cert = ks.getCertificate(alias);
```

- `getCertificateAlias`

  指定された証明書とデータが一致する最初のキーストアエントリの名前 (エイリアス) を返します。

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
```

```
String alias = ks.getCertificateAlias(cert);
```

- `getCertificateChain`

  指定されたエイリアスに関連付けられた証明書チェーンを返します。エイリアスが存在しないか、証明書ではないオブジェクトを参照している場合、関数は NULL を返します。

- `getCreationDate`

  指定されたエイリアスによって識別されるエントリの作成日を返します。作成日が使用できない場合、この関数は証明書が有効になった日付を返します。

- `getKey`

  GetKey は HSM に渡され、指定されたラベルに対応するキーオブジェクトを返します。HSM `getKey`に直接クエリを実行すると、 によって生成されたかどうかにかかわらず、HSM 上の任意のキーに使用できます KeyStore。

  ```
  Key key = ks.getKey(keyLabel, null);
  ```

- `isCertificateEntry`

  指定されたエイリアスを持つエントリが証明書エントリを表すかどうかをチェックします。

- `isKeyEntry`

  指定されたエイリアスを持つエントリがキーエントリを表すかどうかをチェックします。このアクションは、PKCS12 ファイルと HSM の両方でエイリアスを検索します。

- `setCertificateEntry`

  指定された証明書を指定されたエイリアスに割り当てます。指定されたエイリアスがキーまたは証明書の識別にすでに使用されている場合は、`KeyStoreException` がスローされます。JCE コードを使用してキーオブジェクトを取得し、 メソッドを使用して KeyStore `SetKeyEntry`証明書をキーに関連付けることができます。

- `byte[]` キーのある `setKeyEntry`

  この API は現在、Client SDK 5 ではサポートされていません。

- `Key` オブジェクトのある `setKeyEntry`

  指定されたキーを指定されたエイリアスに割り当て、HSM 内に保存します。キーが HSM 内にまだ存在しない場合は、抽出可能なセッションキーとして HSM にインポートされます。

Key オブジェクトが PrivateKey のタイプの場合、対応する証明書チェーンが添付されている必要があります。

エイリアスが既に存在する場合、SetKeyEntry 呼び出しは KeyStoreException をスローし、キーが上書きされるのを防ぎます。キーを上書きする必要がある場合は、そのために KMU または JCE を使用します。

- engineSize

  キーストア内のエントリの数を返します。

- store

  キーストアを指定された出力ストリームに PKCS12 ファイルとして保存し、指定されたパスワードで保護します。さらに、ロードされたすべてのキー (setKey 呼び出しを使用して設定される) が保持されます。

JCE プロバイダーをクライアント SDK 3 からクライアント SDK 5 に移行する

このトピックでは、JCE プロバイダーを Client SDK 3 から Client SDK 5 に移行します。移行の利点については、「」を参照してくださいClient SDK 5 の利点。

では AWS CloudHSM、顧客アプリケーションは AWS CloudHSM クライアントソフトウェア開発キット (SDK) を使用して暗号化オペレーションを実行します。Client SDK 5 は、引き続き新機能とプラットフォームサポートが追加されているプライマリ SDK です。

Client SDK 3 JCE プロバイダーは、標準 JCE 仕様に含まれていないカスタムクラスと APIs を使用します。JCE プロバイダーのクライアント SDK 5 は JCE 仕様に準拠しており、特定の領域ではクライアント SDK 3 と下位互換性がありません。お客様のアプリケーションでは、クライアント SDK 5 への移行の一環として変更が必要になる場合があります。このセクションでは、移行を成功させるために必要な変更点の概要を説明します。

すべてのプロバイダーの移行手順を確認するには、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。

トピック

- 重大な変更に対処して準備する
- Client SDK 5 への移行

- 関連トピック

重大な変更に対処して準備する

これらの重大な変更を確認し、それに応じて開発環境でアプリケーションを更新します。

プロバイダークラスと名前が変更されました

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
プロバイダークラスと名前	Client SDK 3 の JCE プロバイダークラスは と呼ばれCaviumProvider 、プロバイダー名は ですCavium。	クライアント SDK 5 では、プロバイダークラス が呼び出CloudHsmProvider され、プロバイダー名は ですCloudHSM。	CloudHsmProvider オブジェクトを初期化する方法の例は、AWS CloudHSM GitHub サンプルリポジトリ にあります。

明示的なログインが変更されましたが、暗黙的な は変更されていません

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
明示的なログイン	クライアント SDK 3 は、明示的なログイン に LoginManager クラスを使用します[1]。	クライアント SDK 5 では、CloudHSMプロバイダーは明示的なログインAuthProvider のために を実装します。 AuthProvider は標準の Java クラスであり、Java のイディオマティックな方法に従ってプロバイダーにログインします。Client SDK	Client SDK 5 で明示的なログインを使用する方法の例については、AWS CloudHSM LoginRunner サンプルリポジトリ のサンプルを参照してください。 AWS CloudHSM GitHub

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
		5 のログイン状態管理が改善されたため、アプリケーションは再接続中にログインをモニタリングして実行する必要がなくなりました[2]。	
暗黙的なログイン	暗黙的なログインに変更は必要ありません。Client SDK 3 から Client SDK 5 に移行する場合、暗黙的なログインでは、同じプロパティファイルとすべての環境変数が引き続き機能します。		Client SDK 5 で暗黙的なログインを使用する方法の例については、LoginRunner サンプル AWS CloudHSM GitHub リポジトリのサンプルを参照してください。

- [1] クライアント SDK 3 コードスニペット：

```
LoginManager lm = LoginManager.getInstance();

lm.login(partition, user, pass);
```

- [2] クライアント SDK 5 コードスニペット：

```
// Construct or get the existing provider object
AuthProvider provider = new CloudHsmProvider();

// Call login method on the CloudHsmProvider object
// Here loginHandler is a CallbackHandler
provider.login(null, loginHandler);
```

Client SDK 5 で明示的なログインを使用する方法の例については、LoginRunner サンプル AWS CloudHSM GitHub リポジトリのサンプルを参照してください。

キー生成が変更されました

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
キー生成	Client SDK 3 では、`Cavium[Key-type]AlgorithmParameterSpec` を使用してキー生成パラメータを指定します。コードスニペットについては、「脚注」を参照してください[1]。	Client SDK 5 では、`KeyAttributesMap` を使用してキー生成属性を指定します。コードスニペットについては、「脚注」を参照してください[2]。	`KeyAttributesMap` を使用して対称キーを生成する方法の例については、AWS CloudHSM Github Symmetric Keys サンプルリポジトリのサンプルを参照してください。
キーペアの生成	Client SDK 3 では、`Cavium[Key-type]Algorithmpa rameterSpec` を使用してキーペア生成パラメータを指定します。コードスニペットについては、「脚注」を参照してください[3]。	クライアント SDK 5 では、`KeyPairAttributesMap` を使用してこれらのパラメータを指定します。コードスニペットについては、「脚注」を参照してください[4]。	`KeyAttributesMap` を使用して非対称キーを生成する方法の例については、AsymmetricKeys サンプル AWS CloudHSM GitHub リポジトリのサンプルを参照してください。

- [1] クライアント SDK 3 キー生成コードスニペット：

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES", "Cavium");
CaviumAESKeyGenParameterSpec aesSpec = new CaviumAESKeyGenParameterSpec(
keySizeInBits,
keyLabel,
isExtractable,
isPersistent);
```

```
keyGen.init(aesSpec);
SecretKey aesKey = keyGen.generateKey();
```

- [2] クライアント SDK 5 キー生成コードスニペット：

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES",
CloudHsmProvider.PROVIDER_NAME);

final KeyAttributesMap aesSpec = new KeyAttributesMap();
aesSpec.put(KeyAttribute.LABEL, keyLabel);
aesSpec.put(KeyAttribute.SIZE, keySizeInBits);
aesSpec.put(KeyAttribute.EXTRACTABLE, isExtractable);
aesSpec.put(KeyAttribute.TOKEN, isPersistent);

keyGen.init(aesSpec);
SecretKey aesKey = keyGen.generateKey();
```

- [3] クライアント SDK 3 キーペア生成コードスニペット::

```
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("rsa", "Cavium");
CaviumRSAKeyGenParameterSpec spec = new CaviumRSAKeyGenParameterSpec(
keySizeInBits,
new BigInteger("65537"),
label + ":public",
label + ":private",
isExtractable,
isPersistent);

keyPairGen.initialize(spec);

keyPairGen.generateKeyPair();
```

- [4] クライアント SDK 5 キーペア生成コードスニペット：

```
KeyPairGenerator keyPairGen =
KeyPairGenerator.getInstance("RSA", providerName);

// Set attributes for RSA public key
final KeyAttributesMap publicKeyAttrsMap = new KeyAttributesMap();
publicKeyAttrsMap.putAll(additionalPublicKeyAttributes);
publicKeyAttrsMap.put(KeyAttribute.LABEL, label + ":Public");
publicKeyAttrsMap.put(KeyAttribute.MODULUS_BITS, keySizeInBits);
publicKeyAttrsMap.put(KeyAttribute.PUBLIC_EXPONENT,
```

```
new BigInteger("65537").toByteArray());

// Set attributes for RSA private key
final KeyAttributesMap privateKeyAttrsMap = new KeyAttributesMap();
privateKeyAttrsMap.putAll(additionalPrivateKeyAttributes);
privateKeyAttrsMap.put(KeyAttribute.LABEL, label + ":Private");

// Create KeyPairAttributesMap and use that to initialize the
// keyPair generator
KeyPairAttributesMap keyPairSpec =
new KeyPairAttributesMapBuilder()
.withPublic(publicKeyAttrsMap)
.withPrivate(privateKeyAttrsMap)
.build();

keyPairGen.initialize(keyPairSpec);
keyPairGen.generateKeyPair();
```

キーの検索、削除、参照が変更されました

AWS CloudHSM を使用して、 で既に生成されたキーを検索する KeyStore。Client SDK 3 には、Caviumと の 2 つの KeyStore タイプがありますCloudHSM。クライアント SDK 5 には KeyStore 、タイプ が 1 つだけありますCloudHSM。

から Cavium KeyStore に移行するには、 KeyStore タイプの変更CloudHSM KeyStore が必要です。さらに、クライアント SDK 3 はキーを参照するためにキーハンドルを使用し、クライアント SDK 5 はキーラベルを使用します。結果として生じる動作の変更を以下に示します。

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
キーリファレンス	Client SDK 3 では、アプリケーションはキーラベルまたはキーハンドルを使用して HSM 内のキーを参照します。でラベルを使用してキー KeyStore を検索するか、ハンドルを使用	Client SDK 5 では、アプリケーションはを使用してラベルでキーAWS CloudHSM KeyStore Java クラスの使用を検索できます。ハンドルでキーを検索するには、 AWS CloudHSM	

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
	してCaviumKey オブジェクトを作成します。	KeyStoreWithAttributes で を使用します AWS CloudHSM KeyReferenceSpec 。	
複数のエントリの検索	getEntry、、または を使用してキーを検索するgetCertificate 場合getKey、同じ基準を持つ複数の項目が に存在するシナリオではCavium KeyStore、見つかった最初のエントリのみが返されます。	AWS CloudHSM KeyStore と ではKeyStoreWithAttributes 、同じシナリオで例外がスローされます。この問題を解決するには、CloudHSM CLI の key set-attribute コマンドを使用してキーに一意のラベルを設定することをお勧めします。またはKeyStoreWithAttributes#getKeys 、を使用して、基準に一致するすべてのキーを返します。	

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
すべてのキーを検索する	Client SDK 3 では、を使用して HSM 内のすべてのキーを検索できますUtil.findAllKeys() 。	Client SDK 5 では、 KeyStoreWithAttributes クラスを使用することで、検索キーがより簡単かつ効率的になります。可能であれば、レイテンシーを最小限に抑えるためにキーをキャッシュします。詳細については、「アプリケーションのキーを効果的に管理する」を参照してください。HSM からすべてのキーを取得する必要がある場合は、空の KeyStoreWithAttributes#getKeys で を使用しますKeyAttributesMap 。	KeyStoreWithAttributes クラスを使用してキーを検索する例は AWS CloudHSM Github サンプルリポジトリで利用でき、コードスニペットはに示されています1。

変更点	クライアント SDK 3 の内容	Client SDK 5 の内容	例
キー削除	Client SDK 3 は `Util.deleteKey()` を使用してキーを削除します。	Client SDK 5 の `Key` オブジェクトは、この`Destroyable` インターフェイスの `destroy()` メソッドを使用してキーを削除できるインターフェイスを実装します。	削除キー機能を示すサンプルコードは、CloudHSM Github サンプルリポジトリ にあります。各 SDK のサンプルスニペットを に示します2。

- [1] スニペットを以下に示します。

```
KeyAttributesMap findSpec = new KeyAttributesMap();
findSpec.put(KeyAttribute.LABEL, label);
findSpec.put(KeyAttribute.KEY_TYPE, keyType);
KeyStoreWithAttributes keyStore = KeyStoreWithAttributes.getInstance("CloudHSM");

keyStore.load(null, null);
keyStore.getKey(findSpec);
```

- [2] Client SDK 3 でキーを削除する：

```
Util.deleteKey(key);
```

  Client SDK 5 でキーを削除する：

```
((Destroyable) key).destroy();
```

暗号アンラップオペレーションが変更され、他の暗号オペレーションは変更されていません

> **Note**
>
> 暗号の暗号化/復号/ラップオペレーションに変更は必要ありません。

アンラップオペレーションでは、Client SDK 3 CaviumUnwrapParameterSpec クラスを、リストされている暗号化オペレーションに固有の次のいずれかのクラスに置き換える必要があります。

- GCMUnwrapKeySpec AES/GCM/NoPaddingアンラップ用
- IvUnwrapKeySpec AESWrap unwrapおよび 用 AES/CBC/NoPadding unwrap
- RSA OAEP unwrap の場合は OAEPUnwrapKeySpec

のス二ペットの例OAEPUnwrapkeySpec：

```
OAEPParameterSpec oaepParameterSpec =
new OAEPParameterSpec(
        "SHA-256",
        "MGF1",
        MGF1ParameterSpec.SHA256,
        PSpecified.DEFAULT);

KeyAttributesMap keyAttributesMap =
        new KeyAttributesMap(KeyAttributePermissiveProfile.KEY_CREATION);
keyAttributesMap.put(KeyAttribute.TOKEN, true);
keyAttributesMap.put(KeyAttribute.EXTRACTABLE, false);

OAEPUnwrapKeySpec spec = new OAEPUnwrapKeySpec(oaepParameterSpec,
        keyAttributesMap);

Cipher hsmCipher =
        Cipher.getInstance(
                "RSA/ECB/OAEPPadding",
                CloudHsmProvider.PROVIDER_NAME);
hsmCipher.init(Cipher.UNWRAP_MODE, key, spec);
```

署名オペレーションが変更されていない

署名オペレーションに変更は必要ありません。

Client SDK 5 への移行

このセクションの手順に従って、Client SDK 3 から Client SDK 5 に移行します。

> **Note**
>
> Amazon Linux、Ubuntu 16.04、Ubuntu 18.04 CentOS 6、CentOS 8、および RHEL 6 は現在、クライアント SDK 5 ではサポートされていません。現在、クライアント SDK 3 でこれらのプラットフォームのいずれかを使用している場合は、クライアント SDK 5 に移行するときに別のプラットフォームを選択する必要があります。

1. Client SDK 3 の JCE プロバイダーをアンインストールします。

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-jce
   ```

   CentOS 7

   ```
   $ sudo yum remove cloudhsm-jce
   ```

   RHEL 7

   ```
   $ sudo yum remove cloudhsm-jce
   ```

   RHEL 8

   ```
   $ sudo yum remove cloudhsm-jce
   ```

2. Client SDK 3 のクライアントデーモンをアンインストールします。

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-client
   ```

   CentOS 7

   ```
   $ sudo yum remove cloudhsm-client
   ```

RHEL 7

```
$ sudo yum remove cloudhsm-client
```

RHEL 8

```
$ sudo yum remove cloudhsm-client
```

> **Note**
>
> カスタム設定を再度有効にする必要があります。

3. 「」の手順に従って、クライアント SDK JCE プロバイダーをインストールしますClient SDK 5 の AWS CloudHSM JCE プロバイダーをインストールして使用する。
4. Client SDK 5 では、新しい設定ファイル形式とコマンドラインブートストラップツールが導入されています。Client SDK 5 JCE プロバイダーをブートストラップするには、 のユーザーガイドに記載されている手順に従ってくださいクライアント SDK をブートストラップする。
5. 開発環境で、アプリケーションをテストします。既存のコードを更新して、最終的な移行前に重大な変更を解決します。

関連トピック

- のベストプラクティス AWS CloudHSM

JCE の詳細設定

AWS CloudHSM JCE プロバイダーには、次の高度な設定が含まれていますが、これはほとんどのお客様が使用する一般的な設定の一部ではありません。

- 複数のクラスターへの接続
- JCE を使用したキー抽出
- JCE の設定を再試行

JCE プロバイダーによる複数のクラスターへの接続

この構成では、1 つのクライアントインスタンスが複数のクラスターと通信できます。1 つのインスタンスが 1 つのクラスターとしか通信しない場合と比較して、これは一部のユースケースではコスト削減機能となる可能性があります。CloudHsmProvider クラス は、Java セキュリティのプロバイダークラス の の実装 AWS CloudHSMです。このクラスの各インスタンスは、 AWS CloudHSM クラスター全体への接続を表します。このクラスをインスタンス化して Java セキュリティプロバイダのリストに追加すると、標準 JCE クラスを使用して操作できるようになります。

次の例では、このクラスをインスタンス化して Java セキュリティプロバイダのリストに追加します。

```
if (Security.getProvider(CloudHsmProvider.PROVIDER_NAME) == null) {
    Security.addProvider(new CloudHsmProvider());
}
```

CloudHsmProvider の設定

CloudHsmProvider は 2 つの方法で設定できます。

1. ファイルによる設定 (デフォルト設定)
2. コードを使用して設定

ファイルによる設定 (デフォルト設定)

デフォルトのコンストラクタを使用して CloudHsmProvider をインスタンス化すると、デフォルトでは、Linux の /opt/cloudhsm/etc/cloudhsm-jce.cfg パスで構成ファイルが検索されます。この設定ファイルは、configure-jce を使用して設定できます。

デフォルトコンストラクターを使用して作成されたオブジェクトは、デフォルトの CloudHSM プロバイダー名 CloudHSM を使用します。プロバイダー名は JCE とやり取りして、さまざまなオペレーションにどのプロバイダーを使用するかを判断するのに役立ちます。Cipher オペレーションに CloudHSM プロバイダー名を使用する例は次のとおりです。

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", "CloudHSM");
```

コードを使用して設定

Client SDK バージョン 5.8.0 以降では、Java コードを使用して CloudHsmProvider を設定することもできます。そのための方法は、CloudHsmProviderConfig クラスのオブジェクトを使用する

ことです。CloudHsmProviderConfigBuilder を使用してこのオブジェクトを構築することができます。

CloudHsmProvider には、次の例のように、CloudHsmProviderConfig オブジェクトを取得する別のコンストラクターがあります。

Example

```
CloudHsmProviderConfig config = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
                                            .withHsmCAFilePath(hsmCAFilePath)

 .withClusterUniqueIdentifier("CloudHsmCluster1")
        .withServer(CloudHsmServer.builder().withHostIP(hostName).build())
                        .build())
        .build();
CloudHsmProvider provider = new CloudHsmProvider(config);
```

この例では、JCE プロバイダー名は CloudHsmCluster1 で、アプリケーションが JCE とやり取りするときに使用できる名前です。

Example

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", "CloudHsmCluster1");
```

あるいは、アプリケーションは上記で作成したプロバイダーオブジェクトを使用して、そのプロバイダーをオペレーションに使用することを JCE に知らせることもできます。

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider);
```

withClusterUniqueIdentifier メソッドで一意の識別子が指定されていない場合は、ランダムに生成されたプロバイダー名が自動的に作成されます。このランダムに生成された識別子を取得するには、アプリケーションは provider.getName() を呼び出して識別子を取得できます。

複数のクラスターへの接続

前述のように、それぞれ CloudHsmProvider が CloudHSM クラスターへの接続を表します。同じアプリケーションから別のクラスターと通信したい場合は、別のクラスターの設定を使用して CloudHsmProvider のオブジェクトをもう1つ作成し、次の例に示すように、プロバイダーオブジェクトまたはプロバイダー名を使用してこの他のクラスターとやり取りできます。

Example

```
CloudHsmProviderConfig config = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
                                            .withHsmCAFilePath(hsmCAFilePath)

 .withClusterUniqueIdentifier("CloudHsmCluster1")
        .withServer(CloudHsmServer.builder().withHostIP(hostName).build())
                        .build())
        .build();
CloudHsmProvider provider1 = new CloudHsmProvider(config);

if (Security.getProvider(provider1.getName()) == null) {
    Security.addProvider(provider1);
}

CloudHsmProviderConfig config2 = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
                                            .withHsmCAFilePath(hsmCAFilePath2)

 .withClusterUniqueIdentifier("CloudHsmCluster2")
        .withServer(CloudHsmServer.builder().withHostIP(hostName2).build())
                        .build())
        .build();
CloudHsmProvider provider2 = new CloudHsmProvider(config2);

if (Security.getProvider(provider2.getName()) == null) {
    Security.addProvider(provider2);
}
```

上記の両方のプロバイダー (両方のクラスター) を設定したら、プロバイダーオブジェクトまたはプロバイダー名を使用してプロバイダーを操作できます。

との通信方法を示すこの例を拡張するとcluster1、AES/GCM/NoPadding オペレーションに次のサンプルを使用できます。

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider1);
```

同じアプリケーションで、プロバイダー名を使用して 2 番目のクラスターで「AES」キー生成を行う場合は、次のサンプルを使用することもできます。

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider2.getName());
```

JCE の再試行コマンド

Client SDK 5.8.0 以降には、HSM スロットリングされたオペレーションをクライアント側から再試行する自動再試行戦略が組み込まれています。HSM が以前のオペレーションが多すぎてそれ以上リクエストを受け付けられないためにオペレーションをスロットリングすると、Client SDK はスロットリングされたオペレーションを最大 3 回再試行しますが、その間、エクスポネンシャルバックオフします。この自動再試行戦略は、オフとスタンダードの 2 つのモードのいずれかに設定できます。

- オフ: Client SDK は、HSM によってスロットリングされたオペレーションに対しては再試行戦略を一切実行しません。
- スタンダード: これは Client SDK 5.8.0 以降のデフォルトモードです。このモードでは、Client SDK はエクスポネンシャルバックオフすることで、スロットリングされた操作を自動的に再試行します。

詳細については、「HSM スロットリング」を参照してください。

再試行コマンドをオフモードに設定する

Linux

Linux 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 次のコマンドを使用して再試行設定を off モードに設定できます。

```
$ sudo /opt/cloudhsm/bin/configure-jce --default-retry-mode off
```

Windows

Windows 上の Client SDK 5 向けに再試行コマンドを off に設定するには

- 次のコマンドを使用して再試行設定を off モードに設定できます。

```
C:\Program Files\Amazon\CloudHSM\bin\ .\configure-jce.exe --default-retry-mode
 off
```

JCE を使用したキー抽出

Java Cryptography Extension (JCE) は、さまざまな暗号化実装をプラグインできるアーキテクチャを使用します。 は、暗号化オペレーションを HSM にオフロードする 1 つの JCE プロバイダーを AWS CloudHSM 出荷します。他のほとんどの JCE プロバイダーが AWS CloudHSM に保存されているキーを操作するには、HSM のキーバイトをクリアテキストでマシンのメモリに抽出して使用する必要があります。HSM では通常、キーを ラップされたオブジェクト としてのみ抽出でき、クリアテキストとして抽出することはできません。ただし、プロバイダー間の統合のユースケースをサポートするために、 はオプトイン設定オプション AWS CloudHSM を許可して、クリア内のキーバイトの抽出を可能にします。

> **Important**
>
> JCE は、AWS CloudHSM プロバイダーが指定されるか、 AWS CloudHSM キーオブジェクトが使用される AWS CloudHSM たびに、オペレーションを にオフロードします。HSM 内でオペレーションが行われることが予想される場合は、キーを明確に抽出する必要はありません。クリアテキストでのキー抽出が必要なのは、サードパーティのライブラリや JCE プロバイダーの制限により、アプリケーションがキーのラップやアンラップなどの安全なメカニズムを使用できない場合のみです。

AWS CloudHSM JCE プロバイダーは、デフォルトで外部 JCE プロバイダーと連携するパブリックキーの抽出を許可します。以下の方法は常に許可されています。

Class	方法	Format (getEncoded)
EcPublicKey	getEncoded ()	X.509
	getW()	該当なし
RSAPublicKey	getEncoded ()	X.509
	getPublicExponent()	該当なし
CloudHsmRsaPrivateCrtKey	getPublicExponent()	該当なし

AWS CloudHSM JCE プロバイダーは、デフォルトではプライベートキーまたはシークレットキーのキーバイトのクリアな抽出を許可しません。ユースケースで必要な場合は、以下の条件で プライベート または シークレット キーのキーバイトを消去して抽出できます。

1. プライベートまたはシークレットキーの EXTRACTABLE 属性は「true」に設定されています。
   - デフォルトでは、プライベートキーとシークレットキーの EXTRACTABLE 属性は「true」に設定されています。EXTRACTABLE キーは HSM からのエクスポートが許可されているキーです。詳細については、「Client SDK 5 向けサポートされている Java 属性」を参照してください。
2. プライベートキーとシークレットキーの WRAP_WITH_TRUSTED 属性は「false」に設定されます。
   - getEncoded、getPrivateExponent、getSおよびクリアでエクスポートできないプライベートキーでは使用できません。WRAP_WITH_TRUSTED は、プライベートキーを HSM からクリアにエクスポートすることはできません。詳細については、「信頼できるキーを使ったキーのアンラップの制御」を参照してください。

AWS CloudHSM JCE プロバイダーが からプライベートキーシークレットを抽出することを許可する AWS CloudHSM

> **Important**
>
> この設定変更により、HSM クラスターからすべてのクリアの EXTRACTABLE キーバイトを抽出できるようになります。セキュリティを高めるには、キーラッピング方法を使用して HSM から安全にキーを抽出することを検討してください。これにより、HSM からキーバイトが意図せず抽出されるのを防ぐことができます。

1. 以下のコマンドを使用して、プライベート キーと シークレット キーを JCE で抽出できるようにします。

   Linux

   ```
   $ /opt/cloudhsm/bin/configure-jce --enable-clear-key-extraction-in-software
   ```

   Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-jce.exe --enable-clear-key-
extraction-in-software
```

2. クリアキー抽出を有効にすると、以下の方法でプライベートキーをメモリーに抽出できるようになります。

Class	方法	Format (getEncoded)
Key	getEncoded ()	RAW
ECPrivateKey	getEncoded ()	PKCS#8
	getS()	該当なし
RSAPrivateCrtKey	getEncoded ()	X.509
	getPrivateExponent()	該当なし
	getPrimeP()	該当なし
	getPrimeQ()	該当なし
	getPrimeExponentP()	該当なし
	getPrimeExponentQ ()	該当なし
	getCrtCoefficient()	該当なし

JCE がキーをクリアで エクスポートできないようにして、デフォルトの動作に戻したい場合は、以下のコマンドを実行します。

Linux

```
$ /opt/cloudhsm/bin/configure-jce --disable-clear-key-extraction-in-software
```

Windows

```
C:\Program Files\Amazon\CloudHSM\> .\configure-jce.exe --disable-clear-key-
extraction-in-software
```

暗号化 API: Microsoft Windows 用の Next Generation (CNG) とキーストレージプロバイダー (KSP)

Windows 用の AWS CloudHSM クライアントには、CNG プロバイダーと KSP プロバイダーが含まれています。現在、CNG および KSP プロバイダーをサポートしているのは、Client SDK 3 だけです。

Key storage providers (KSPs)により、キーの格納と取得が可能になります。たとえば、Microsoft の Active Directory Certificate Services (AD CS) の役割を Windows サーバーに追加し、認証機関 (CA) の新しいプライベートキーを作成するのを選択した場合は、キーストレージを管理する KSP を選択できます。AD CS のロールを設定するときは、KSP を選択できます。詳細については、「Windows Server CA の作成」を参照してください。

Cryptography API: Next Generation (CNG) は、Microsoft Windows オペレーティングシステム固有の暗号化 API です。CNG を使用すると、開発者は暗号化技術を使用して Windows ベースのアプリケーションを保護できます。大まかに言うと、CNG の AWS CloudHSM 実装には次の機能があります。

- 暗号化プリミティブ - 基本的な暗号化オペレーションを実行することができます。
- Key Import and Export - 対称キーと非対称キーのインポートとエクスポートを可能にします。
- Data Protection API (CNG DPAPI) - データの暗号化と復号化を簡単にすることができます。
- キーの保管と取り出し - 非対称キーペアのプライベートキーを安全に保管し隔離することができます。

トピック

- Windows 用 KSP および CNG プロバイダーを確認する
- Windows の AWS CloudHSM 前提条件
- AWS CloudHSM キーを証明書に関連付ける
- CNG プロバイダーのコード例

Windows 用 KSP および CNG プロバイダーを確認する

KSP および CNG プロバイダーは、Windows AWS CloudHSM クライアントをインストールするときにインストールされます。クライアントは、「クライアントのインストール (Windows)」の手順に従ってインストールします。

Windows AWS CloudHSM クライアントを設定して実行する

Windows CloudHSM クライアントを開始する前に、前提条件 を満たす必要があります。次に、プロバイダーが使用する設定ファイルを更新し、以下の手順を実行してクライアントを起動します。これらのステップは、KSP および CNG プロバイダーの初回使用時と、クラスターの HSM の追加または削除を行った後に必要です。これにより、クラスター内のすべての HSM AWS CloudHSM でデータを同期し、一貫性を維持できます。 HSMs

ステップ 1: AWS CloudHSM クライアントを停止する

プロバイダーが使用する設定ファイルを更新する前に、 AWS CloudHSM クライアントを停止します。クライアントが停止済みである場合、stop コマンドを実行しても影響はありません。

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

  AWS CloudHSM クライアントを起動したコマンドウィンドウで Ctrl +C を使用します。

ステップ 2: AWS CloudHSM 設定ファイルを更新する

この手順では、-aConfigure ツールの パラメータを使用して、クラスター内の HSM の 1 つの Elastic Network Interface (ENI) IP アドレスを設定ファイルに追加します。

```
C:\Program Files\Amazon\CloudHSM configure.exe -a <HSM ENI IP>
```

クラスター内の HSM の ENI IP アドレスを取得するには、 AWS CloudHSM コンソールに移動し、クラスター を選択し、目的のクラスターを選択します。DescribeClusters オペレーション、describe-clusters コマンド、または Get-HSM2Cluster PowerShell コマンドレットを使用することもできます。1 つの ENI IP アドレスのみを入力します。どの ENI IP アドレスでも使用できます。

ステップ 3: AWS CloudHSM クライアントを起動する

次に、 AWS CloudHSM クライアントを起動または再起動します。 AWS CloudHSM クライアントは起動時に、設定ファイル内の ENI IP アドレスを使用してクラスターをクエリします。次に、クラスター内のすべての HSM の ENI IP アドレスを、クラスター情報ファイルに追加します。

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

KSP および CNG プロバイダーの確認

次のいずれかのコマンドを使用して、システムにインストールするプロバイダーを決定します。コマンドは、登録された KSP および CNG プロバイダーをリスト表示。 AWS CloudHSM クライアントを実行する必要はありません。

```
C:\Program Files\Amazon\CloudHSM>ksp_config.exe -enum
```

```
C:\Program Files\Amazon\CloudHSM>cng_config.exe -enum
```

KSP および CNG プロバイダーが Windows Server EC2 インスタンスにインストールされていることを確認するには、リスト中に次のエントリが表示されているのを見る必要があります。

```
Cavium CNG Provider
Cavium Key Storage Provider
```

CNG プロバイダーが見つからない場合は、次のコマンドを実行します。

```
C:\Program Files\Amazon\CloudHSM>cng_config.exe -register
```

CNG プロバイダーが見つからない場合は、次のコマンドを実行します。

```
C:\Program Files\Amazon\CloudHSM>ksp_config.exe -register
```

Windows の AWS CloudHSM 前提条件

Windows AWS CloudHSM クライアントを起動して KSP および CNG プロバイダーを使用する前に、システムで HSM のログイン認証情報を設定する必要があります。Windows Credential Manager またはシステム環境変数を使用して、認証情報を設定できます。認証情報の保存には、Windows Credential Manager を使用することをお勧めします。このオプションは、 AWS CloudHSM クライアントバージョン 2.0.4 以降で使用できます。環境変数を使用すると設定が簡単になりますが、Windows Credential Manager を使用するよりも安全性が低くなります。

Windows Credential Manager

`set_cloudhsm_credentials` ユーティリティまたは Windows Credential Manager インターフェイスのいずれかを使用できます。

- **`set_cloudhsm_credentials` ユーティリティの使用:**

  `set_cloudhsm_credentials` ユーティリティは Windows インストーラに含まれています。このユーティリティを使用して、HSM ログイン認証情報を Windows Credential Manager に簡単に渡すことができます。このユーティリティをソースからコンパイルする場合は、インストーラに含まれている Python コードを使用できます。

  1. `C:\Program Files\Amazon\CloudHSM\tools\` フォルダに移動します。
  2. CU ユーザー名とパスワードのパラメータを使用して `set_cloudhsm_credentials.exe` ファイルを実行します。

     ```
     set_cloudhsm_credentials.exe --username <CU USER> --password <CU PASSWORD>
     ```

- Credential Manager インターフェイスの使用:

  Credential Manager インターフェイスを使用して、認証情報を手動で管理できます。

  1. Credential Manager を開くには、タスクバーの検索ボックスに「`credential manager`」と入力し、[Credential Manager] を選択します。
  2. [Windows 資格情報] を選択して、Windows 認証情報を管理します。
  3. [汎用資格情報の追加] を選択し、以下のように詳細を入力します。
     - [インターネットまたはネットワークアドレス] にターゲット名として「`cloudhsm_client`」と入力します。
     - [ユーザー名] と [パスワード] に CU 認証情報を入力します。
     - [OK] をクリックします。

システム環境変数

Windows アプリケーションの HSM および暗号化ユーザー (CU) を識別するシステム環境変数を設定できます。setx command マンドを使用して、システム環境変数を設定するか、permanentシステム環境変数を programmaticallyに設定するか 、あるいは Windows の System Properties コントロールパネルの Advanced タブ中に設定します。

> **Warning**
>
> システム環境変数を使用して認証情報を設定すると、ユーザーのシステムでパスワードがプレーンテキストで入手可能になります。この問題を解決するには、Windows Credential Manager を使用します。

次のシステム環境変数を設定します。

`n3fips_password=`*`CU USERNAME`*`:`*`CU PASSWORD`*

HSM の 暗号化ユーザー (CU) を識別し、必要なすべての口グイン情報を提供します。アプリケーションはこの CU として認証および実行します。このアプリケーションには、この CU のアクセス権限があり、CU が所有および共有しているキーのみを表示および管理できます。新しい CU を作成するには、createUser を使用します。既存の CU を検索するには、listUsers を使用します。

例:

```
setx /m n3fips_password test_user:password123
```

AWS CloudHSM キーを証明書に関連付ける

Microsoft の などのサードパーティーツールで AWS CloudHSM キーを使用する前にSignTool、キーのメタデータをローカル証明書ストアにインポートし、メタデータを証明書に関連付ける必要があります。キーのメタデータをインポートするには、CloudHSM バージョン 3.0 以降に含まれている import_key.exe ユーティリティを使用します。次の手順では、追加情報とサンプル出力を示します。

ステップ 1: 証明書をインポートする

Windows では、証明書をダブルクリックするとローカルの証明書ストアにインポートできます。

ただし、ダブルクリックしてもインポートできない場合は、Microsoft Certreq ツールを使用して証明書マネージャーに証明書をインポートします。例えば:

```
certreq -accept certificatename
```

この操作が失敗し、エラー Key not found が表示された場合は、手順 2 に進みます。証明書がキーストアに表示される場合は、タスクは完了しているため、これ以上の操作は必要ありません。

ステップ 2: 証明書識別情報を収集する

前の手順が成功しなかった場合は、プライベートキーを証明書に関連付ける必要があります。ただし、関連付けを作成する前に、まず証明書の一意のコンテナ名とシリアル番号を検索する必要があります。certutil などのユーティリティを使用して、必要な証明書情報を表示します。certutil からの次の出力例は、コンテナ名とシリアル番号を示しています。

```
================ Certificate 1 ================ Serial Number:
   72000000047f7f7a9d41851b4e000000000004Issuer: CN=Enterprise-CANotBefore: 10/8/2019
 11:50
   AM NotAfter: 11/8/2020 12:00 PMSubject: CN=www.example.com, OU=Certificate
 Management,
   O=Information Technology, L=Seattle, S=Washington, C=USNon-root CertificateCert
   Hash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45 75 bc 65No key
 provider
   information Simple container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
 Unique
   container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
```

ステップ 3: AWS CloudHSM プライベートキーを証明書に関連付ける

キーを証明書に関連付けるには、まずAWS CloudHSM クライアントデーモン を起動してください。次に、import_key.exe (CloudHSM バージョン 3.0 以降に含まれています) を使用して、プライベートキーを証明書に関連付けます。証明書を指定するときは、その単純なコンテナ名を使用します。次の例は、コマンドと応答を示しています。このアクションでは、キーのメタデータのみがコピーされます。キーは HSM に残ります。

```
$> import_key.exe -RSA CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c

Successfully opened Microsoft Software Key Storage Provider : 0NCryptOpenKey failed :
 80090016
```

ステップ 4: 証明書ストアを更新する

AWS CloudHSM クライアントデーモンがまだ実行されていることを確認します。次に、certutil 動詞の -repairstore を使用して、証明書のシリアル番号を更新します。次のサンプルは、コマンドと出力の例を示しています。-repairstore 動詞の詳細については、Microsoft のドキュメントを参照してください。

```
C:\Program Files\Amazon\CloudHSM>certutil -f -csp "Cavium Key Storage Provider"-
repairstore my "72000000047f7f7a9d41851b4e000000000004"
my "Personal"
================ Certificate 1 ================
Serial Number: 72000000047f7f7a9d41851b4e000000000004
Issuer: CN=Enterprise-CA
NotBefore: 10/8/2019 11:50 AM
NotAfter: 11/8/2020 12:00 PM
Subject: CN=www.example.com, OU=Certificate Management, O=Information Technology,
 L=Seattle, S=Washington, C=US
Non-root CertificateCert Hash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45
 75 bc 65
SDK Version: 3.0
Key Container = CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
Provider = Cavium Key Storage ProviderPrivate key is NOT exportableEncryption test
 passedCertUtil: -repairstore command completed successfully.
```

証明書のシリアル番号を更新したら、Windows の任意のサードパーティー署名ツールでこの証明書と対応する AWS CloudHSM プライベートキーを使用できます。

CNG プロバイダーのコード例

⚠ ** コード例のみ - 本稼働環境で使用しないでください **
このサンプルコードは、例示のみを目的としています。本稼働環境でこのコードを実行しないでください。

次の例では、Windows のための CloudHSM を用いてインストールされた CNG プロバイダーを見つけるために、登録された暗号化プロバイダーをシステム上で列挙する方法について示しています。この例では、非対称キーペアの作成方法と、キーペアを使用してデータに署名する方法も示しています。

> **Important**
>
> この例を実行する前に、前提条件の説明に従って HSM 認証情報を設定する必要があります。詳細については、「Windows の AWS CloudHSM 前提条件」を参照してください。

```
// CloudHsmCngExampleConsole.cpp : Console application that demonstrates CNG
 capabilities.
// This example contains the following functions.
//
//   VerifyProvider()          - Enumerate the registered providers and retrieve Cavium
 KSP and CNG providers.
//   GenerateKeyPair()         - Create an RSA key pair.
//   SignData()                - Sign and verify data.
//

#include "stdafx.h"
#include <Windows.h>

#ifndef NT_SUCCESS
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)
#endif

#define CAVIUM_CNG_PROVIDER L"Cavium CNG Provider"
#define CAVIUM_KEYSTORE_PROVIDER L"Cavium Key Storage Provider"

// Enumerate the registered providers and determine whether the Cavium CNG provider
// and the Cavium KSP provider exist.
//
bool VerifyProvider()
{
  NTSTATUS status;
  ULONG cbBuffer = 0;
  PCRYPT_PROVIDERS pBuffer = NULL;
  bool foundCng = false;
  bool foundKeystore = false;

  // Retrieve information about the registered providers.
  //   cbBuffer - the size, in bytes, of the buffer pointed to by pBuffer.
  //   pBuffer - pointer to a buffer that contains a CRYPT_PROVIDERS structure.
```

```
  status = BCryptEnumRegisteredProviders(&cbBuffer, &pBuffer);

  // If registered providers exist, enumerate them and determine whether the
  // Cavium CNG provider and Cavium KSP provider have been registered.
  if (NT_SUCCESS(status))
  {
    if (pBuffer != NULL)
    {
      for (ULONG i = 0; i < pBuffer->cProviders; i++)
      {
        // Determine whether the Cavium CNG provider exists.
        if (wcscmp(CAVIUM_CNG_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
        {
          printf("Found %S\n", CAVIUM_CNG_PROVIDER);
          foundCng = true;
        }

        // Determine whether the Cavium KSP provider exists.
        else if (wcscmp(CAVIUM_KEYSTORE_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
        {
          printf("Found %S\n", CAVIUM_KEYSTORE_PROVIDER);
          foundKeystore = true;
        }
      }
    }
  }
  else
  {
    printf("BCryptEnumRegisteredProviders failed with error code 0x%08x\n", status);
  }

  // Free memory allocated for the CRYPT_PROVIDERS structure.
  if (NULL != pBuffer)
  {
    BCryptFreeBuffer(pBuffer);
  }

  return foundCng == foundKeystore == true;
}

// Generate an asymmetric key pair. As used here, this example generates an RSA key
 pair
// and returns a handle. The handle is used in subsequent operations that use the key
 pair.
```

```
// The key material is not available.
//
// The key pair is used in the SignData function.
//
NTSTATUS GenerateKeyPair(BCRYPT_ALG_HANDLE hAlgorithm, BCRYPT_KEY_HANDLE *hKey)
{
  NTSTATUS status;

  // Generate the key pair.
  status = BCryptGenerateKeyPair(hAlgorithm, hKey, 2048, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptGenerateKeyPair failed with code 0x%08x\n", status);
    return status;
  }

  // Finalize the key pair. The public/private key pair cannot be used until this
  // function is called.
  status = BCryptFinalizeKeyPair(*hKey, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptFinalizeKeyPair failed with code 0x%08x\n", status);
    return status;
  }

  return status;
}

// Sign and verify data using the RSA key pair. The data in this function is hardcoded
// and is for example purposes only.
//
NTSTATUS SignData(BCRYPT_KEY_HANDLE hKey)
{
  NTSTATUS status;
  PBYTE sig;
  ULONG sigLen;
  ULONG resLen;
  BCRYPT_PKCS1_PADDING_INFO pInfo;

  // Hardcode the data to be signed (for demonstration purposes only).
  PBYTE message = (PBYTE)"d83e7716bed8a20343d8dc6845e57447";
  ULONG messageLen = strlen((char*)message);

  // Retrieve the size of the buffer needed for the signature.
```

```
  status = BCryptSignHash(hKey, NULL, message, messageLen, NULL, 0, &sigLen, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptSignHash failed with code 0x%08x\n", status);
    return status;
  }

  // Allocate a buffer for the signature.
  sig = (PBYTE)HeapAlloc(GetProcessHeap(), 0, sigLen);
  if (sig == NULL)
  {
    return -1;
  }

  // Use the SHA256 algorithm to create padding information.
  pInfo.pszAlgId = BCRYPT_SHA256_ALGORITHM;

  // Create a signature.
  status = BCryptSignHash(hKey, &pInfo, message, messageLen, sig, sigLen, &resLen,
 BCRYPT_PAD_PKCS1);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptSignHash failed with code 0x%08x\n", status);
    return status;
  }

  // Verify the signature.
  status = BCryptVerifySignature(hKey, &pInfo, message, messageLen, sig, sigLen,
 BCRYPT_PAD_PKCS1);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptVerifySignature failed with code 0x%08x\n", status);
    return status;
  }

  // Free the memory allocated for the signature.
  if (sig != NULL)
  {
    HeapFree(GetProcessHeap(), 0, sig);
    sig = NULL;
  }

  return 0;
}
```

```
// Main function.
//
int main()
{
  NTSTATUS status;
  BCRYPT_ALG_HANDLE hRsaAlg;
  BCRYPT_KEY_HANDLE hKey = NULL;

  // Enumerate the registered providers.
  printf("Searching for Cavium providers...\n");
  if (VerifyProvider() == false) {
    printf("Could not find the CNG and Keystore providers\n");
    return 1;
  }

  // Get the RSA algorithm provider from the Cavium CNG provider.
  printf("Opening RSA algorithm\n");
  status = BCryptOpenAlgorithmProvider(&hRsaAlg, BCRYPT_RSA_ALGORITHM,
 CAVIUM_CNG_PROVIDER, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptOpenAlgorithmProvider RSA failed with code 0x%08x\n", status);
    return status;
  }

  // Generate an asymmetric key pair using the RSA algorithm.
  printf("Generating RSA Keypair\n");
  GenerateKeyPair(hRsaAlg, &hKey);
  if (hKey == NULL)
  {
    printf("Invalid key handle returned\n");
    return 0;
  }
  printf("Done!\n");

  // Sign and verify [hardcoded] data using the RSA key pair.
  printf("Sign/Verify data with key\n");
  SignData(hKey);
  printf("Done!\n");

  // Remove the key handle from memory.
  status = BCryptDestroyKey(hKey);
  if (!NT_SUCCESS(status))
```

```
  {
    printf("BCryptDestroyKey failed with code 0x%08x\n", status);
    return status;
  }

  // Close the RSA algorithm provider.
  status = BCryptCloseAlgorithmProvider(hRsaAlg, NULL);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptCloseAlgorithmProvider RSA failed with code 0x%08x\n", status);
    return status;
  }

  return 0;
}
```

以前のクライアント SDK (クライアント SDK 3)

AWS CloudHSM には、2 つの主要な Client SDK バージョンが含まれています。

- Client SDK 5: これは最新かつデフォルトの Client SDK です。Client SDK 5 による利点については、「Client SDK 5 の利点」を参照してください。
- Client SDK 3: これは古い Client SDK です。プラットフォームおよび言語ベースのアプリケーションの互換性および管理ツール用のコンポーネントの完全なセットが含まれています。

Client SDK 3 から Client SDK 5 に移行する手順については、「」を参照してくださいClient SDK 3 から Client SDK 5 への移行。

Client SDK 3 のドキュメントはこのトピックに記載されています。

ダウンロードするには、「ダウンロード」を参照してください。

Client SDK バージョンをチェックする

Amazon Linux

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

Amazon Linux 2

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

CentOS 6

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

CentOS 7

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

CentOS 8

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

RHEL 6

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

RHEL 7

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

RHEL 8

以下のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

Ubuntu 16.04 LTS

以下のコマンドを使用します。

```
apt list --installed | grep ^cloudhsm
```

Ubuntu 18.04 LTS

以下のコマンドを使用します。

```
apt list --installed | grep ^cloudhsm
```

Ubuntu 20.04 LTS

以下のコマンドを使用します。

```
apt list --installed | grep ^cloudhsm
```

Windows Server

以下のコマンドを使用します。

```
wmic product get name,version
```

Client SDK コンポーネントの比較

Client SDK 3 には、コマンドラインツールに加え、さまざまなプラットフォームまたは言語ベースのアプリケーションから HSM に暗号化オペレーションをオフロードできるコンポーネントが含まれています。Client SDK 5 は Client SDK 3 と同等ですが、CNG および KSP プロバイダーはまだサポートしていません。次の表では、クライアント SDK 3 と クライアント SDK 5 のコンポーネントの可用性を比較します。

コンポーネント	クライアント SDK 5	クライアント SDK 3
PKCS #11 ライブラリ	はい	はい
JCE プロバイダー	はい	はい
OpenSSL Dynamic Engine	はい	はい

コンポーネント	クライアント SDK 5	クライアント SDK 3
CNG および KSP プロバイダー		はい
CloudHSM 管理ユーティリティ (CMU)[1]	はい	はい
キー管理ユーティリティ (KMU)[1]	はい	はい
設定ツール	はい	はい

[1] CMU と KMU コンポーネントは、Client SDK 5 を搭載した CloudHSM CLI に含まれています。

トピック

- Client SDK 3 がサポートするプラットフォーム
- Linux での Client SDK 3 のアップグレード
- Client SDK 3 用の PKCS #11 ライブラリ
- OpenSSL Dynamic Engine 用の Client SDK 3 をインストールする
- JCE プロバイダー用Client SDK 3

Client SDK 3 がサポートするプラットフォーム

クライアント SDK 3 ではクライアントデーモンが必要で、CloudHSM 管理ユーティリティ (CMU)、キー管理ユーティリティ (KMU)、構成ツールなどのコマンドラインツールを提供しています。

基本サポートは、 AWS CloudHSM クライアント SDK のバージョンごとに異なります。通常、SDK 内のコンポーネントのプラットフォームのサポートは基本サポートと一致しますが、必ずしもそうとは限りません。特定のコンポーネントのプラットフォームサポートを決定するには、まず目的のプラットフォームが SDK のベースセクションに表示されていることを確認してから、コンポーネントセクションで除外やその他の関連情報がないか確認します。

プラットフォームのサポートは時間の経過とともに変化します。以前のバージョンの CloudHSM クライアント SDK では、ここに記載されているすべてのオペレーティングシステムがサポートされていない場合があります。リリースノートを使用して、以前のバージョンの CloudHSM クライアン

ト SDK に対するオペレーティングシステムサポートを確認します。詳細については、「Client SDK AWS CloudHSM のダウンロード」を参照してください。

AWS CloudHSM は 64 ビットオペレーティングシステムのみをサポートします。

目次

Linux サポート

- Amazon Linux
- Amazon Linux 2
- CentOS 6.10+ [2]
- CentOS 7.3+
- CentOS 8 [1、4]
- Red Hat Enterprise Linux (RHEL) 6.10+ [2]
- Red Hat Enterprise Linux (RHEL) 7.3+
- Red Hat Enterprise Linux (RHEL) 8 [1]
- Ubuntu 16.04 LTS [3]
- Ubuntu 18.04 LTS [1]

[1] OpenSSL Dynamic Engine はサポートされていません。詳細については、「OpenSSL Dynamic Engine」を参照してください。

[2] Client SDK 3.3.0 以降はサポートされていません。

[3] SDK 3.4 は Ubuntu 16.04 でサポートされる最後のリリースです。

[4] SDK 3.4 は CentOS 8.3+ でサポートされる最後のリリースです。

Windows サポート

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Client SDK 3 の HSM 互換性

hsm1.medium	hsm2m.medium
クライアントバージョン SDK 3.1.0 以降と互換性があります。	サポート外。

コンポーネントのサポート

PKCS #11 ライブラリ

PKCS #11 ライブラリは Linux の基本サポートに一致する Linux のみのコンポーネントです。詳細については、「the section called “Linux サポート”」を参照してください。

CloudHSM 管理ユーティリティ (CMU)

CloudHSM 管理ユーティリティ (CMU) コマンドラインツールは、Crypto Officer が HSMs内のユーザーを管理するのに役立ちます。これには、ユーザーの作成、削除および一覧表示とユーザーパスワードの変更を行うツールが含まれています。詳細については、「CloudHSM 管理ユーティリティ (CMU)」を参照してください。

キー管理ユーティリティ (KMU)

Key Management Utility (KMU) は、Crypto User (CU) がハードウェアセキュリティモジュール (HSM) のキーを管理するのに役立つコマンドラインツールです。詳細については、「キー管理ユーティリティ (KMU)」を参照してください。

JCE プロバイダー

JCE プロバイダーは Linux の基本サポートに一致する Linux のみのコンポーネントです。詳細については、「the section called “Linux サポート”」を参照してください。

- OpenJDK 1.8 が必要です。

OpenSSL Dynamic Engine

OpenSSL Dynamic Engine は Linux の基本サポートと一致 しない Linux のみのコンポーネントです。以下の除外項目を参照してください。

- OpenSSL 1.0.2[f+] が必要です。

サポートされていないプラットフォーム

- CentOS 8
- Red Hat Enterprise Linux (RHEL) 8
- Ubuntu 18.04 LTS

これらのプラットフォームは、クライアント SDK 3 の OpenSSL Dynamic Engine と互換性がないバージョンの OpenSSL が付属しています。 AWS CloudHSM は、クライアント SDK 5 の OpenSSL Dynamic Engine でこれらのプラットフォームをサポートしています。

CNG および KSP プロバイダー

CNG および KSP プロバイダーは、Windows の基本サポートに一致する Windows のみのコンポーネントです。詳細については、「Windows サポート」を参照してください。

Linux での Client SDK 3 のアップグレード

AWS CloudHSM Client SDK 3.1 以降では、アップグレードするには、クライアントデーモンのバージョンとインストールするコンポーネントが一致している必要があります。すべての Linux ベースのシステムでは、1 つのコマンドを使用して、同じバージョンの PKCS #11 ライブラリ、Java 暗号化拡張機能 (JCE) プロバイダー、または OpenSSL Dynamic Engine を使用してクライアントデーモンを一括更新する必要があります。CNG および KSP プロバイダーのバイナリーがすでにクライアントデーモンパッケージに含まれているため、この要件は Windows ベースのシステムには適用されません。

クライアントデーモンバージョンをチェックするには

- Red Hat ベースの Linux システム (Amazon Linux および CentOS を含む) では、次のコマンドを使用します。

```
rpm -qa | grep ^cloudhsm
```

- Debian ベースの Linux システムでは、次のコマンドを使用します。

```
apt list --installed | grep ^cloudhsm
```

- Windows システムでは、次のコマンドを使用します。

```
wmic product get name,version
```

トピック

前提条件

最新バージョンの AWS CloudHSM クライアントデーモンをダウンロードし、コンポーネントを選択します。

> **Note**
>
> すべてのコンポーネントをインストールする必要はありません。インストールしたコンポーネントごとに、クライアントデーモンのバージョンに合わせてコンポーネントをアップグレードする必要があります。

最新の Linux クライアントデーモン

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

最新の PKCS #11 ライブラリ

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

最新の OpenSSL Dynamic Engine

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-dyn_latest_amd64.deb
```

最新の JCE プロバイダー

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

ステップ 1: クライアントデーモンを停止する

クライアントデーモンを停止するには、次のコマンドを使用します。

Amazon Linux

```
$ sudo stop cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client stop
```

CentOS 7

```
$ sudo service cloudhsm-client stop
```

CentOS 8

```
$ sudo service cloudhsm-client stop
```

RHEL 7

```
$ sudo service cloudhsm-client stop
```

RHEL 8

```
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client stop
```

ステップ 2: Client SDK をアップグレードする

次のコマンドは、クライアントデーモンとコンポーネントのアップグレードに必要な構文を示しています。コマンドを実行する前に、アップグレードしないコンポーネントをすべて削除します。

Amazon Linux

```
$ sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el6.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el6.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el6.x86_64.rpm>
```

Amazon Linux 2

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 7

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 8

```
$ sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm> \
```

```
                <./cloudhsm-client-jce-latest.el8.x86_64.rpm>
```

RHEL 7

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
                <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
                <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
                <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

RHEL 8

```
$ sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm \
                <./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm> \
                <./cloudhsm-client-jce-latest.el8.x86_64.rpm>
```

Ubuntu 16.04 LTS

```
$ sudo apt install ./cloudhsm-client_latest_amd64.deb \
                <cloudhsm-client-pkcs11_latest_amd64.deb> \
                <cloudhsm-client-dyn_latest_amd64.deb> \
                <cloudhsm-client-jce_latest_amd64.deb>
```

Ubuntu 18.04 LTS

```
$ sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb \
                <cloudhsm-client-pkcs11_latest_amd64.deb> \
                <cloudhsm-client-jce_latest_amd64.deb>
```

ステップ 3: クライアントデーモンを起動する

クライアントデーモンを起動するには、以下のコマンドを使用します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

Client SDK 3 用の PKCS #11 ライブラリ

PKCS #11 は、ハードウェアセキュリティモジュール (HSM) で暗号化オペレーションを実行するための標準です。

ブートストラップの詳細については、「クラスターへの接続」を参照してください。

トピック

Client SDK 3 用の PKCS #11 ライブラインストールする

クライアント SDK 3 の前提条件

PKCS #11 ライブラリには AWS CloudHSM クライアントが必要です。

AWS CloudHSM クライアントをインストールして設定していない場合は、「」の手順に従って実行しますクライアント (Linux) のインストール。クライアントのインストールと設定が完了したら、次のコマンドを使用して起動します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

CentOS 8

```
$ sudo systemctl cloudhsm-client start
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 8

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Client SDK 3 用の PKCS #11 ライブラリのインストール

次のコマンドでは、PKCS #11 ライブラリをダウンロードしてインストールします。

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

- PKCS #11 ライブラリをインストールした EC2 インスタンスに、クライアント SDK 3 の他のコンポーネントがインストールされていない場合は、クライアント SDK 3 をブートストラップする必要があります。クライアント SDK 3 のコンポーネントを使用して、各インスタンスで 1 回だけ実行する必要があります。
- PKCS #11 ライブラリのファイルは、次の場所にあります。

  Linuxのバイナリ、設定スクリプト、証明書およびログファイル:

```
/opt/cloudhsm/lib
```

PKCS #11 ライブラリへの認証 (Client SDK 3)

PKCS #11 を使用すると、アプリケーションは HSMs で特定の [Crypto User (CU)] として実行されます。アプリケーションは、CU が所有して共有するキーのみを表示および管理できます。既存の CU を HSM で使用することも、新しい CU を作成することもできます。CU の管理については、「Managing HSM users with CloudHSM CLI」および「Managing HSM users with CloudHSM Management Utility (CMU)」を参照してください。

PKCS #11 に CU を指定するには、PKCS #11 [C_Login 関数] のピンパラメーターを使用します。の場合 AWS CloudHSM、ピンパラメータの形式は次のとおりです。

```
<CU_user_name>:<password>
```

たとえば、次のコマンドでユーザー名 `CryptoUser` とパスワード `CUPassword123!` を使用して PKCS #11 ライブラリのピンを CU に設定します。

```
CryptoUser:CUPassword123!
```

サポートされているキータイプ (Client SDK 3)

PKCS #11 ライブラリは、次のキータイプをサポートしています。

キータイプ	説明
RSA	256 ビットの増分で、2048 ~ 4096 ビットの RSA キーを生成します。

キータイプ	説明
EC	secp224r1 (P-224)、secp256r1 (P-256)、secp256k1 (ブロックチェーン)、secp384r1 (P-384)、secp521r1 (P-521) のカーブを使用してキーを生成します。
AES	128、192、256 ビットの AES キーを生成します。
3DES (Triple DES)	192 ビットの DES3 キーを生成します。今後の変更については、以下の注記「1」を参照してください。
[GENERIC_SECRET]	1 ~ 64 バイトの汎用シークレットを生成します。

- [1] NIST ガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

サポートされているメカニズム (Client SDK 3)

PKCS #11 ライブラリは、次のアルゴリズムをサポートしています。

- [暗号化と復号化] AES-CBC、AES-CTR、AES-ECB、AES-GCM、DES3-CBC、DES3-ECB、RSA-OAEP、RSA-PKCS
- [署名と確認] RSA、HMAC、ECDSA (ハッシュあり、なし)
- [ハッシュ/ダイジェスト] SHA1、SHA224、SHA256、SHA384、SHA512
- [キーラップ] AES キーラップ、[4] AES-GCM、RSA-AES、RSA-OAEP
- [キーの導出] — ECDH、[5] SP800-108 CTR KDF

PKCS #11 ライブラリのメカニズムと関数を示す表

PKCS #11 ライブラリは PKCS #11 仕様のバージョン 2.40 に準拠しています。PKCS#11 を使用して暗号化機能を呼び出すには、指定されたメカニズムで関数を呼び出します。次の表は、 AWS CloudHSMでサポートされている関数とメカニズムの組み合わせをまとめたものです。

サポートされている PKCS#11 メカニズムと関数を示すテーブルの解釈

✓ マークは、 が関数のメカニズム AWS CloudHSM をサポートしていることを示します。PKCS #11 仕様に一覧表示されている利用可能な関数がすべてサポートされているわけではありません。✖ マークは、PKCS #11 標準で許可されている場合でも、 が指定された関数のメカニズムをまだサポート AWS CloudHSM していないことを示します。空のセルは、PKCS #11 標準で特定の関数のメカニズムがサポートされていないことを示します。

サポートされている PKCS#11 ライブラリのメカニズムと関数

メカニズム	関数						
	キーの生成またはキーペア	署名と検証	SR と VR	ダイジェスト	暗号化と復号	派生キー	ラップと UnWrap
CKM_RSA_PKCS_KEY_PAIR_GEN	✓						
CKM_RSA_X9_31_KEY_PAIR_GEN	✓[2]						
CKM_RSA_X_509		✓			✓		
CKM_RSA_PKCS [注]		✓[1]	✖		✓[1]		✓[1]

メカニズム	関数						
記「8」を参照							
CKM_RSA_PKCS_OAEP					✓1		✓6
CKM_SHA1_RSA_PKCS		✓3.2					
CKM_SHA224_RSA_PKCS		✓3.2					
CKM_SHA256_RSA_PKCS		✓3.2					
CKM_SHA384_RSA_PKCS		✓2,3.2					
CKM_SHA512_RSA_PKCS		✓3.2					
CKM_RSA_PKCS_PSS		✓1					
CKM_SHA1_RSA_PKCS_PSS		✓3.2					
CKM_SHA224_RSA_PKCS_PSS		✓3.2					

メカニズム	関数						
CKM_SHA256_RSA_PKCS_PSS		✓[3.2]					
CKM_SHA384_RSA_PKCS_PSS		✓[2,3.2]					
CKM_SHA512_RSA_PKCS_PSS		✓[3.2]					
CKM_EC_KEY_PAIR_GEN	✓						
CKM_ECDSA		✓[1]					
CKM_ECDSA_SHA1		✓[3.2]					
CKM_ECDSA_SHA224		✓[3.2]					
CKM_ECDSA_SHA256		✓[3.2]					
CKM_ECDSA_SHA384		✓[3.2]					
CKM_ECDSA_SHA512		✓[3.2]					
CKM_ECDH1_DERIVE						✓[5]	

メカニズム	関数						
CKM_SP800_108_COUNTER_KDF						✓	
CKM_GENERIC_SECRET_KEY_GEN	✓						
CKM_AES_KEY_GEN	✓						
CKM_AES_ECB					✓		✖
CKM_AES_CTR					✓		✖
CKM_AES_CBC					✓[3.3]		✖
CKM_AES_CBC_PAD					✓		✖
CKM_DES3_KEY_GEN 注記「8」を参照	✓						
CKM_DES3_CBC 注記「8」を参照					✓[3.3]		✖

メカニズム	関数						
CKM_DES3_CBC_PAD 注記「8」を参照					✓		✖
CKM_DES3_ECB 注記「8」を参照					✓		✖
CKM_AES_GCM					✓[3.3, 4]		✓[7.1]
CKM_CLOUDHSM_AES_GCM					✓[7.1]		✓[7.1]
CKM_SHA_1				✓[3.1]			
CKM_SHA_1_HMAC		✓[3.3]					
CKM_SHA224				✓[3.1]			
CKM_SHA224_HMAC		✓[3.3]					
CKM_SHA256				✓[3.1]			
CKM_SHA256_HMAC		✓[3.3]					

メカニズム	関数						
CKM_SHA384				✓[3.1]			
CKM_SHA384_HMAC		✓[3.3]					
CKM_SHA512				✓[3.1]			
CKM_SHA512_HMAC		✓[3.3]					
CKM_RSA_AES_KEY_WRAP							✓
CKM_AES_KEY_WRAP							✓
CKM_AES_KEY_WRAP_PAD							✓
CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD							✓[7.1]
CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD							✓[7.1]

メカニズム	関数						
CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD							✓[7.1]

メカニズムの注釈

- [1] シングルパートのオペレーションのみ
- [2] メカニズムは機能的には CKM_RSA_PKCS_KEY_PAIR_GEN のメカニズムと似ていますが、p と q の生成に関してより強力な保証を提供します。
- [3.1] クライアント SDK に基づいてハッシュを異なる方法で AWS CloudHSM アプローチします。クライアント SDK 3 では、ハッシュを行う場所はデータのサイズと、シングルパートオペレーションとマルチパートオペレーションのどちらを使用するかによって異なります。

  クライアント SDK 3 のシングルパートのオペレーション

  表 3.1 に、クライアント SDK 3 の各メカニズムの最大のデータ設定サイズを表します。ハッシュ全体が HSM 内で計算されます。16KB を超えるデータサイズはサポートされません。

  表 3.1 シングルパートオペレーションの最大のデータ設定サイズ

[メカニズム]	[最大データサイズ]
CKM_SHA_1	16296
CKM_SHA224	16264
CKM_SHA256	16296
CKM_SHA384	16232
CKM_SHA512	16232

  クライアント SDK 3 のマルチパートオペレーション

16 KB を超えるデータサイズのサポートについては、データサイズによってハッシュが行われる場所が決まります。16 KB 未満のデータバッファは HSM 内でハッシュされます。16 KBからシステムの最大のデータサイズまでのバッファは、ソフトウェアでローカルにハッシュされます。[留意点]: ハッシュ関数は機密情報の暗号化を必要としないため、HSM の外部で安全にコンピューティングすることができます。

- [3.2] クライアント SDK に基づいてハッシュを異なる方法で AWS CloudHSM アプローチします。クライアント SDK 3 では、ハッシュを行う場所はデータのサイズと、シングルパートオペレーションとマルチパートオペレーションのどちらを使用するかによって異なります。

  クライアント SDK 3 のシングルパートのオペレーション

  表 3.2 に、クライアント SDK 3 の各メカニズムの最大のデータ設定サイズを表します。16KB を超えるデータサイズはサポートされません。

  表 3.2 シングルパートオペレーションの最大のデータ設定サイズ

[メカニズム]	[最大データサイズ]
CKM_SHA1_RSA_PKCS	16296
CKM_SHA224_RSA_PKCS	16264
CKM_SHA256_RSA_PKCS	16296
CKM_SHA384_RSA_PKCS	16232
CKM_SHA512_RSA_PKCS	16232
CKM_SHA1_RSA_PKCS_PSS	16296
CKM_SHA224_RSA_PKCS_PSS	16264
CKM_SHA256_RSA_PKCS_PSS	16296
CKM_SHA384_RSA_PKCS_PSS	16232
CKM_SHA512_RSA_PKCS_PSS	16232
CKM_ECDSA_SHA1	16296

[メカニズム]	[最大データサイズ]
CKM_ECDSA_SHA224	16264
CKM_ECDSA_SHA256	16296
CKM_ECDSA_SHA384	16232
CKM_ECDSA_SHA512	16232

クライアント SDK 3 のマルチパートオペレーション

16 KB を超えるデータサイズのサポートについては、データサイズによってハッシュが行われる場所が決まります。16 KB 未満のデータバッファは HSM 内でハッシュされます。16 KBからシステムの最大のデータサイズまでのバッファは、ソフトウェアでローカルにハッシュされます。[留意点]: ハッシュ関数は機密情報の暗号化を必要としないため、HSM の外部で安全にコンピューティングすることができます。

- [3.3] 以下のいずれかのメカニズムを使用してデータを操作する際、データバッファが最大データサイズを超えるとエラーになります。これらのメカニズムでは、すべてのデータ処理が HSM 内で行われる必要があります。次の表は、各メカニズムに設定されている最大データサイズを示します:

  表 3.3 最大のデータ設定サイズ

[メカニズム]	[最大データサイズ]
CKM_SHA_1_HMAC	16288
CKM_SHA224_HMAC	16256
CKM_SHA256_HMAC	16288
CKM_SHA384_HMAC	16224
CKM_SHA512_HMAC	16224
CKM_AES_CBC	16272
CKM_AES_GCM	16224

[メカニズム]	[最大データサイズ]
CKM_CLOUDHSM_AES_GCM	16224
CKM_DES3_CBC	16280

- [4] AES-GCM の暗号化を実行している際、HSM はアプリケーションからの初期化ベクトル (IV) データを受け入れません。HSM が生成した IV を使用する必要があります。HSM で生成された 12 バイトの IV は、指定した CK_GCM_PARAMS パラメータ構造の pIV 要素が指すメモリ参照に書き込まれます。ユーザーが混乱しないよう、バージョン 1.1.1 以降の PKCS#11 SDK では、AES-GCM 暗号化が初期化されると、pIV はゼロ化されたバッファを指し示すようになっています。
- [5]クライアント SDK 3 のみ。メカニズムは SSL/TLS オフロードのケースをサポートするために実装されており、HSM 内の一部でのみ実行されます。このメカニズムを使用する前に、「PKCS#11 ライブラリの既知の問題」の「Issue: ECDH key derivation is executed only partially within the HSM」を参照してください。CKM_ECDH1_DERIVE では、 secp521r1 (P-521) カーブはサポートされません。
- [6] 次の CK_MECHANISM_TYPE および CK_RSA_PKCS_MGF_TYPE は、CK_RSA_PKCS_OAEP_PARAMS の CKM_RSA_PKCS_OAEP としてサポートされています:
  - CKG_MGF1_SHA1 を使用する CKM_SHA_1
  - CKG_MGF1_SHA224 を使用する CKM_SHA224
  - CKG_MGF1_SHA256 を使用する CKM_SHA256
  - CKM_MGF1_SHA384 を使用する CKM_SHA384
  - CKM_MGF1_SHA512 を使用する CKM_SHA512
- [7.1] ベンダー定義のメカニズム。CloudHSM ベンダー定義のメカニズムを使用するには、コンパイル時に PKCS #11 アプリケーションに /opt/cloudhsm/include/pkcs11t.h を含める必要があります。

  CKM_CLOUDHSM_AES_GCM: この独自のメカニズムは、標準 CKM_AES_GCM よりもプログラム的に安全な代替手段です。これは、HSM によって生成された IV を、暗号の初期化中に提供される CK_GCM_PARAMS 構造体に書き戻すのではなく、暗号文の先頭に付加します。このメカニズムは C_Encrypt、C_WrapKey、C_Decrypt、C_UnwrapKey 関数で使用できます。このメカニズムを使用する場合は、CK_GCM_PARAMS 構造体内の pIV 変数を NULL に設定する必要があります。このメカニズムを C_Decrypt および C_UnwrapKey と共に使用する場合、IV は、ラップ解除される暗号文の前に付加されることが想定されます。

  CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD: PKCS #5 パディングを使用する AES キーラップ

CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD: ゼロパディングを使用する AES キーラップ

AES キーラップに関する追加情報については、[AES キーラップ] を参照してください。

- [8] NIST ガイダンスに従い、これは 2023 年以降の FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

サポートされている API オペレーション (Client SDK 3)

PKCS #11 ライブラリは、次の PKCS #11 API オペレーションをサポートしています。

- `C_CloseAllSessions`
- `C_CloseSession`
- `C_CreateObject`
- `C_Decrypt`
- `C_DecryptFinal`
- `C_DecryptInit`
- `C_DecryptUpdate`
- `C_DeriveKey`
- `C_DestroyObject`
- `C_Digest`
- `C_DigestFinal`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_Encrypt`
- `C_EncryptFinal`
- `C_EncryptInit`
- `C_EncryptUpdate`
- `C_Finalize`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_FindObjectsInit`

- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignRecover(クライアント SDK 3 のサポートのみ)
- C_SignRecoverInit(クライアント SDK 3 のサポートのみ)
- C_SignUpdate
- C_UnWrapKey
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyRecover(クライアント SDK 3 のサポートのみ)
- C_VerifyRecoverInit(クライアント SDK 3 のサポートのみ)
- C_VerifyUpdate
- C_WrapKey

サポートされているキー属性 (Client SDK 3)

キーオブジェクトには、パブリックキー、プライベートキー、またはシークレットキーを指定できます。キーオブジェクトで許可されているアクションは属性で指定されます。属性は、キーオブジェクトの作成時に定義されます。PKCS#11 ライブラリを使用する際、PKCS#11 標準で指定されたデフォルト値が割り当てられます。

AWS CloudHSM は、PKCS #11 仕様に記載されているすべての属性をサポートしているわけではありません。サポートするすべての属性の仕様に準拠しています。これらの属性は、それぞれのテーブルにリストされています。

オブジェクトを作成、変更、またはコピーする `C_CreateObject`、`C_GenerateKey`、`C_GenerateKeyPair`、`C_UnwrapKey`、`C_DeriveKey` などの暗号化関数は、属性テンプレートをパラメータの 1 つとして使用します。オブジェクトの作成中に属性テンプレートを渡す方法の詳細については、「Generate keys through PKCS #11 library」のサンプルを参照してください。

PKCS#11 ライブラリの属性テーブルの解釈

PKCS#11 ライブラリのテーブルには、キータイプによって異なる属性のリストが含まれています。で特定の暗号化関数を使用する場合、特定の属性が特定のキータイプでサポートされているかどうかを示します AWS CloudHSM。

凡例:

- ✓ CloudHSM が特定のキータイプの属性をサポートしていることを示します。
- ✖ CloudHSM が特定のキータイプの属性をサポートしていないこと示します。
- R は、属性値が特定のキータイプに対して読み取り専用に設定されていることを示します。
- S は、属性が機密であるため、`GetAttributeValue` で読み取れないことを示します。
- [Default Value] 列のセルが空の場合は、属性に割り当てられている特定のデフォルト値がないことを示します。

GenerateKeyPair

属性	キータイプ				デフォルト値
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	
CKA_CLASS	✓	✓	✓	✓	
CKA_KEY_TYPE	✓	✓	✓	✓	
CKA_LABEL	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	True
CKA_TOKEN	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✓	✖	✓	False
CKA_DECRYPT	✓	✖	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	True

属性	キータイプ				デフォルト値
CKA_DESTROYABLE	✓	✓	✓	✓	True
CKA_SIGN	✓	✗	✓	✗	False
CKA_SIGN_RECOVER	✗	✗	✓[3]	✗	
CKA_VERIFY	✗	✓	✗	✓	False
CKA_VERIFY_RECOVER	✗	✗	✗	✓[4]	
CKA_WRAP	✗	✓	✗	✓	False
CKA_WRAP_TEMPLATE	✗	✓	✗	✓	
CKA_TRUSTED	✗	✓	✗	✓	False
CKA_WRAP_WITH_TRUSTED	✓	✗	✓	✗	False
CKA_UNWRAP	✓	✗	✓	✗	False
CKA_UNWRAP_TEMPLATE	✓	✗	✓	✗	
CKA_SENSITIVE	✓	✗	✓	✗	True

属性	キータイプ				デフォルト値
CKA_ALWAYS_SENSITIVE	R	✖	R	✖	
CKA_EXTRACTABLE	✓	✖	✓	✖	True
CKA_NEVER_EXTRACTABLE	R	✖	R	✖	
CKA_MODULUS	✖	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✓[2]	
CKA_PRIME_1	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	✖	

属性	キータイプ				デフォルト値
CKA_PUBLIC_EXPONENT	✖	✖	✖	✓[2]	
CKA_EC_PARAMS	✖	✓[2]	✖	✖	
CKA_EC_POINT	✖	✖	✖	✖	
CKA_VALUE	✖	✖	✖	✖	
CKA_VALUE_LEN	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	

GenerateKey

属性	キータイプ			デフォルト値
	AES	DES3	汎用シークレット	
CKA_CLASS	✓	✓	✓	
CKA_KEY_TYPE	✓	✓	✓	
CKA_LABEL	✓	✓	✓	
CKA_ID	✓	✓	✓	

属性	キータイプ			デフォルト値
CKA_LOCAL	R	R	R	True
CKA_TOKEN	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✓	✓	✖	False
CKA_DECRYPT	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	True
CKA_SIGN	✓	✓	✓	True
CKA_SIGN_RECOVER	✖	✖	✖	
CKA_VERIFY	✓	✓	✓	True
CKA_VERIFY_RECOVER	✖	✖	✖	
CKA_WRAP	✓	✓	✖	False
CKA_WRAP_TEMPLATE	✓	✓	✖	

属性	キータイプ			デフォルト値
CKA_TRUSTED	✓	✓	✖	False
CKA_WRAP_WITH_TRUSTED	✓	✓	✓	False
CKA_UNWRAP	✓	✓	✖	False
CKA_UNWRAP_TEMPLATE	✓	✓	✖	
CKA_SENSITIVE	✓	✓	✓	True
CKA_ALWAYS_SENSITIVE	✖	✖	✖	
CKA_EXTRACTABLE	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	
CKA_MODULUS	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	
CKA_PRIME_1	✖	✖	✖	

属性	キータイプ			デフォルト値
CKA_PRIME_2	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	
CKA_EC_POINT	✖	✖	✖	
CKA_VALUE	✖	✖	✖	
CKA_VALUE_LEN	✓[2]	✖	✓[2]	
CKA_CHECK_VALUE	R	R	R	

CreateObject

属性	キータイプ							デフォルト値
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	AES	DES3	汎用シークレット	
CKA_CLASS	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✖	✖	✓	✓	✓	✖	False
CKA_DECRYPT	✖	✖	✓	✖	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓	False

属性	キータイプ							デフォルト値
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✖	✓	✖	✓	✓	✓	False
CKA_SIGN_RECOVER	✖	✖	✓[3]	✖	✖	✖	✖	False
CKA_VERIFY	✖	✓	✖	✓	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	✓[4]	✖	✖	✖	
CKA_WRAP	✖	✖	✖	✓	✓	✓	✖	False
CKA_WRAP_TEMPLATE	✖	✓	✖	✓	✓	✓	✖	
CKA_TRUSTED	✖	✓	✖	✓	✓	✓	✖	False
CKA_WRAP_WITH_TRUSTED	✓	✖	✓	✖	✓	✓	✓	False
CKA_UNWRAP	✖	✖	✓	✖	✓	✓	✖	False

属性	キータイプ							デフォルト値
CKA_UNWRAP_TEMPLATE	✓	✖	✓	✖	✓	✓	✖	
CKA_SENSITIVE	✓	✖	✓	✖	✓	✓	✓	True
CKA_ALWAYS_SENSITIVE	R	✖	R	✖	R	R	R	
CKA_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	✖	R	✖	R	R	R	
CKA_MODULUS	✖	✖	✓[2]	✓[2]	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✖	✖	✖	✖	
CKA_PRIME_1	✖	✖	✓	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✓	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✓	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✓	✖	✖	✖	✖	

属性	キータイプ							デフォルト値
CKA_EXPONENT_2	✖	✖	✓	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✓[2]	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✓[2]	✓[2]	✖	✖	✖	
CKA_EC_PARAMS	✓[2]	✓[2]	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✓[2]	✖	✖	✖	✖	✖	
CKA_VALUE	✓[2]	✖	✖	✖	✓[2]	✓[2]	✓[2]	
CKA_VALUE_LEN	✖	✖	✖	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	R	R	R	

UnwrapKey

属性	キータイプ						デフォルト値
		EC プライベート	RSA プライベート	AES	DES3	汎用シークレット	

属性	キータイプ					デフォルト値
CKA_CLASS	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT	✖	✖	✓	✓	✖	False
CKA_DECRYPT	✖	✓	✓	✓	✖	False
CKA_DERIVE	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✓	✓	✓	✓	False

属性	キータイプ					デフォルト値
CKA_SIGN_RECOVER	✖	✓[3]	✖	✖	✖	False
CKA_VERIFY	✖	✖	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	✖	✖	
CKA_WRAP	✖	✖	✓	✓	✖	False
CKA_UNWRAP	✖	✓	✓	✓	✖	False
CKA_SENSITIVE	✓	✓	✓	✓	✓	True
CKA_EXTRACTABLE	✓	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	R	R	
CKA_ALWAYS_SENSITIVE	R	R	R	R	R	
CKA_MODULUS	✖	✖	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✖	✖	

属性	キータイプ					デフォルト値
CKA_PRIME_1	✖	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✖	✖	✖	✖	
CKA_VALUE	✖	✖	✖	✖	✖	
CKA_VALUE_LEN	✖	✖	✖	✖	✖	

属性	キータイプ					デフォルト値
CKA_CHECK_VALUE		R	R	R	R	R	

DeriveKey

属性	キータイプ				デフォルト値
		AES	DES3	汎用シークレット	
CKA_CLASS		✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE		✓[2]	✓[2]	✓[2]	
CKA_LABEL		✓	✓	✓	
CKA_ID		✓	✓	✓	
CKA_LOCAL		R	R	R	True
CKA_TOKEN		✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✓	✓	✖	False
CKA_DECRYPT		✓	✓	✖	False
CKA_DERIVE		✓	✓	✓	False

属性	キータイプ			デフォルト値
CKA_MODIFIABLE	✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE	✓[1]	✓[1]	✓[1]	True
CKA_SIGN	✓	✓	✓	False
CKA_SIGN_RECOVER	✖	✖	✖	
CKA_VERIFY	✓	✓	✓	False
CKA_VERIFY_RECOVER	✖	✖	✖	
CKA_WRAP	✓	✓	✖	False
CKA_UNWRAP	✓	✓	✖	False
CKA_SENSITIVE	✓	✓	✓	True
CKA_EXTRACTABLE	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	
CKA_ALWAYS_SENSITIVE	R	R	R	

属性	キータイプ			デフォルト値
CKA_MODULUS	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	
CKA_PRIME_1	✖	✖	✖	
CKA_PRIME_2	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✖	
CKA_EC_PARAMS	✖	✖	✖	
CKA_EC_POINT	✖	✖	✖	
CKA_VALUE	✖	✖	✖	

属性	キータイプ			デフォルト値
CKA_VALUE_LEN		✓[2]	✖	✓[2]	
CKA_CHECK_VALUE		R	R	R	

GetAttributeValue

属性	キータイプ						
	EC プライベート	EC パブリック	RSA プライベート	RSA パブリック	AES	DES3	汎用シークレット
CKA_CLASS	✓	✓	✓	✓	✓	✓	✓
CKA_KEY_TYPE	✓	✓	✓	✓	✓	✓	✓
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓
CKA_ID	✓	✓	✓	✓	✓	✓	✓
CKA_LOCAL	✓	✓	✓	✓	✓	✓	✓
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓
CKA_PRIVATE	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]
CKA_ENCRYPT	✖	✖	✖	✓	✓	✓	✖

属性	キータイプ						
CKA_DECRYPT	✖	✖	✓	✖	✓	✓	✖
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓
CKA_MODIFIABLE	✓	✓	✓	✓	✓	✓	✓
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓
CKA_SIGN	✓	✖	✓	✖	✓	✓	✓
CKA_SIGN_RECOVER	✖	✖	✓	✖	✖	✖	✖
CKA_VERIFY	✖	✓	✖	✓	✓	✓	✓
CKA_VERIFY_RECOVER	✖	✖	✖	✓	✖	✖	✖
CKA_WRAP	✖	✖	✖	✓	✓	✓	✖
CKA_WRAP_TEMPLATE	✖	✓	✖	✓	✓	✓	✖
CKA_TRUSTED	✖	✓	✖	✓	✓	✓	✓
CKA_WRAP_WITH_TRUSTED	✓	✖	✓	✖	✓	✓	✓
CKA_UNWRAP	✖	✖	✓	✖	✓	✓	✖

属性	キータイプ						
CKA_UNWRAP_TEMPLATE	✓	✖	✓	✖	✓	✓	✖
CKA_SENSITIVE	✓	✖	✓	✖	✓	✓	✓
CKA_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓
CKA_NEVER_EXTRACTABLE	✓	✖	✓	✖	✓	✓	✓
CKA_ALWAYS_SENSITIVE	R	R	R	R	R	R	R
CKA_MODULUS	✖	✖	✓	✓	✖	✖	✖
CKA_MODULUS_BITS	✖	✖	✖	✓	✖	✖	✖
CKA_PRIME_1	✖	✖	S	✖	✖	✖	✖
CKA_PRIME_2	✖	✖	S	✖	✖	✖	✖
CKA_COEFFICIENT	✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_1	✖	✖	S	✖	✖	✖	✖

属性	キータイプ						
CKA_EXPONENT_2	✖	✖	S	✖	✖	✖	✖
CKA_PRIVATE_EXPONENT	✖	✖	S	✖	✖	✖	✖
CKA_PUBLIC_EXPONENT	✖	✖	✓	✓	✖	✖	✖
CKA_EC_PARAMS	✓	✓	✖	✖	✖	✖	✖
CKA_EC_POINT	✖	✓	✖	✖	✖	✖	✖
CKA_VALUE	S	✖	✖	✖	✓[2]	✓[2]	✓[2]
CKA_VALUE_LEN	✖	✖	✖	✖	✓	✖	✓
CKA_CHECK_VALUE	✓	✓	✓	✓	✓	✓	✖

属性注釈

- [1] この属性はファームウェアによって部分的にサポートされており、デフォルト値にのみ明示的に設定する必要があります。
- [2] 必須属性
- [3]クライアント SDK 3 のみ。CKA_SIGN_RECOVER の属性は CKA_SIGN の属性から派生します。設定される場合は、CKA_SIGN に設定されている値と同じ値にのみ設定できます。設定されない場合、CKA_SIGN のデフォルト値が導出されます。CloudHSM では RSA ベースの回復可能な署名メカニズムのみがサポートされるため、この属性は現在 RSA パブリックキーのみに適用されます。

- [4]クライアント SDK 3 のみ。CKA_VERIFY_RECOVER の属性は CKA_VERIFY の属性から派生します。設定される場合は、CKA_VERIFY に設定されている値と同じ値にのみ設定できます。設定されない場合、CKA_VERIFY のデフォルト値が導出されます。CloudHSM では RSA ベースの回復可能な署名メカニズムのみがサポートされるため、この属性は現在 RSA パブリックキーのみに適用されます。

属性の変更

オブジェクトの属性には、オブジェクトが作成された後に変更できるものもありますが、変更できないものもあります。属性を修正するには、cloudhsm_mgmt_util の[setAttribute] コマンドを使用します。また、cloudhsm_mgmt_util の listAttribute コマンドを使用して属性一覧とそれを表す定数を取得することも可能です。

次のリストで、オブジェクトの作成後に変更が許可荒れている許可されている属性が表示されます。

- CKA_LABEL
- CKA_TOKEN

> **Note**
>
> 変更が許可されるには、セッションキーをトークンキーに変更する場合のみです。key_mgmt_util の setAttribute コマンドを使用して属性値を変更します。

- CKA_ENCRYPT
- CKA_DECRYPT
- CKA_SIGN
- CKA_VERIFY
- CKA_WRAP
- CKA_UNWRAP
- CKA_LABEL
- CKA_SENSITIVE
- CKA_DERIVE

> **Note**
>
> この属性ではキー取得がサポートされています。すべてのパブリックキーで `False` を指定する必要があります。`True` に設定することはできません。シークレットキーまたは EC プライベートキーに対しては、`True` または `False` に設定できます。

- `CKA_TRUSTED`

> **Note**
>
> この属性は Crypto Officer (CO) のみによって `True` または `False` に設定できます。

- `CKA_WRAP_WITH_TRUSTED`

> **Note**
>
> この属性をエクスポート可能なデータキーに適用して、このキーを `CKA_TRUSTED` としてマークされたキーでのみラップできるように指定します。1度 `CKA_WRAP_WITH_TRUSTED` を true に設定すると属性は読み取り専用になり、属性を変更または削除することはできません。

エラーコードの解釈

特定のキーでサポートされていない属性をテンプレートで指定すると、エラーが発生します。次の表には、仕様に違反した場合に生成されるエラーコードが含まれています。

エラーコード	説明
`CKR_TEMPLATE_INCONSISTENT`	PKCS#11 仕様に準拠しているが、CloudHSM でサポートされていない属性を属性テンプレートで指定した場合に、このエラーが発生します。
`CKR_ATTRIBUTE_TYPE_INVALID`	PKCS#11 仕様に準拠しているが、CloudHSM でサポートされていない属性の値を取得すると、このエラーが発生します。

エラーコード	説明
CKR_ATTRIBUTE_INCOMPLETE	このエラーは、属性テンプレートで必須属性を指定しなかった場合に発生します。
CKR_ATTRIBUTE_READ_ONLY	このエラーは、属性テンプレートで読み取り専用属性を指定した場合に発生します。

PKCS #11 ライブラリのコードサンプル (Client SDK 3)

のコードサンプルは、PKCS #11 ライブラリを使用して基本的なタスクを実行する方法 GitHub を示しています。

サンプルコードの前提条件

サンプルを実行する前に、以下のステップを実行して環境をセットアップします。

- Client SDK 3 用の PKCS #11 ライブラリ のインストールと設定をします。
- 暗号化ユーザー (CU) の設定をします。アプリケーションは、この HSM アカウントを使用して HSM でコードサンプルを実行します。

コードサンプル

PKCS#11 用 AWS CloudHSM ソフトウェアライブラリのコードサンプルは、 で入手できますGitHub。このリポジトリには、暗号化、復号化、署名、検証など、PKCS #11 を使用して一般的な操作を行う方法の例が含まれています。

- キーの生成 (AES、RSA、EC)
- キー属性のリスト化
- AES GCM を使用したデータの暗号化および復号
- AES_CTR を使用したデータの暗号化および復号
- 3DES を使用したデータの暗号化および復号
- RSAを使用したデータの署名と検証
- HMAC KDFを使用したキーの取得
- PKCS #5 パディングありの AES を使用したキーのラップとラップ解除
- パディングなしの AES を使用したキーのラップとラップ解除

- ゼロパディングありの AES を使用したキーのラップとラップ解除
- AES-GCM を使用したキーのラップとラップ解除
- RSA を使用したキーのラップとラップ解除

OpenSSL Dynamic Engine 用の Client SDK 3 をインストールする

Client SDK 3 では、クラスターに接続するためにはクライアントデーモンが必要です。以下をサポートします。

- 2048、3072、および 4096 ビットキーの RSA キーの生成。
- RSA の署名/検証。
- RSA の暗号化/復号。
- 暗号化された安全で FIPS 検証済みの乱数生成。

トピック

- Client SDK 3 の OpenSSL Dynamic Engine の前提条件
- Client SDK 3 用の OpenSSL Dynamic Engine をインストールする
- クライアント SDK 3 の OpenSSL 動的エンジンを使用する

Client SDK 3 の OpenSSL Dynamic Engine の前提条件

サポートされるプラットフォームの詳細については、「Client SDK 3 がサポートするプラットフォーム」を参照してください。

OpenSSL の AWS CloudHSM 動的エンジンを使用する前に、 AWS CloudHSM クライアントが必要です。 OpenSSL

クライアントは、クラスター内の HSMs との end-to-end 暗号化された通信を確立するデーモンであり、OpenSSL エンジンはクライアントとローカルで通信します。 AWS CloudHSM クライアントをインストールして設定するには、「」を参照してくださいクライアント (Linux) のインストール。次のコマンドを使用して起動します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 6

```
$ sudo systemctl start cloudhsm-client
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 6

```
$ sudo systemctl start cloudhsm-client
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Client SDK 3 用の OpenSSL Dynamic Engine をインストールする

次の手順では、OpenSSL の AWS CloudHSM 動的エンジンをインストールして設定する方法について説明します。アップグレードの詳細については、「Client SDK 3 のアップグレード」を参照してください。

OpenSSL エンジンをインストールして設定するには

1. 以下のコマンドを使用して、OpenSSL エンジンをダウンロードしてインストールします。

   Amazon Linux

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

CentOS 6

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

RHEL 6

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/
cloudhsm-client-dyn_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-dyn_latest_amd64.deb
```

OpenSSL エンジンは /opt/cloudhsm/lib/libcloudhsm_openssl.so にインストールされています。

2. 次のコマンドを使用して、crypto user (CU) の認証情報が含まれている n3fips_password という名前の環境変数を設定します。

```
$ export n3fips_password=<HSM user name>:<password>
```

クライアント SDK 3 の OpenSSL 動的エンジンを使用する

OpenSSL 統合アプリケーションから OpenSSL の AWS CloudHSM 動的エンジンを使用するには、アプリケーションが という名前の OpenSSL 動的エンジンを使用していることを確認しますcloudhsm。動的エンジンの共有ライブラリは /opt/cloudhsm/lib/libcloudhsm_openssl.so にあります。

OpenSSL コマンドラインから OpenSSL の AWS CloudHSM 動的エンジンを使用するには、OpenSSL-engineオプションを使用して、 という名前の OpenSSL 動的エンジンを指定しますcloudhsm。例:

```
$ openssl s_server -cert server.crt -key server.key -engine cloudhsm
```

JCE プロバイダー用Client SDK 3

AWS CloudHSM JCE プロバイダーは、Java 暗号化拡張 (JCE) プロバイダーフレームワークから構築されたプロバイダー実装です。JCE では、Java 開発キット (JDK) を使用して暗号化操作を実行できます。このガイドでは、 AWS CloudHSM JCE プロバイダーは JCE プロバイダーと呼ばれることがあります。JCE プロバイダーと JDK を使用して、HSM に暗号化操作をオフロードします。

トピック

- Client SDK 3 の AWS CloudHSM JCE プロバイダーをインストールして使用する
- Client SDK 3 のためのサポートメカニズム
- Client SDK 3 の Java キー属性
- Java for Client SDK 3 の AWS CloudHSM ソフトウェアライブラリのコードサンプル
- Client SDK 3 での AWS CloudHSM KeyStore Java クラスの使用

Client SDK 3 の AWS CloudHSM JCE プロバイダーをインストールして使用する

JCE プロバイダーを使用する前に、 AWS CloudHSM クライアントが必要です。

クライアントは、クラスター内の HSMs との end-to-end 暗号化された通信を確立するデーモンです。JCE プロバイダは、クライアントとローカルに通信します。 AWS CloudHSM クライアントパッケージをインストールして設定していない場合は、「」のステップに従って実行しますクライアント (Linux) のインストール。クライアントのインストールと設定が完了したら、次のコマンドを使用して起動します。

JCE プロバイダーは、Linux および互換性のあるオペレーティングシステム上でのみサポートされています。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

CentOS 8

```
$ sudo systemctl cloudhsm-client start
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 8

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

トピック

JCE プロバイダーのインストール

以下のコマンドを使用して、JCE プロバイダーをダウンロードし,インストールします。このプロバイダーは、Linux および互換性のあるオペレーティングシステムでのみサポートされています。

> **Note**
>
> アップグレードについては、「Client SDK 3 のアップグレード」を参照してください。

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-jce-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_u18.04_amd64.deb
```

前述のコマンドを実行すると、次の JCE プロバイダーファイルが表示されます。

- /opt/cloudhsm/java/cloudhsm-*version*.jar
- /opt/cloudhsm/java/cloudhsm-test-*version*.jar
- /opt/cloudhsm/java/hamcrest-all-1.3.jar
- /opt/cloudhsm/java/junit.jar
- /opt/cloudhsm/java/log4j-api-2.17.1.jar
- /opt/cloudhsm/java/log4j-core-2.17.1.jar
- /opt/cloudhsm/lib/libcaviumjca.so

インストールの検証

インストールを検証するには、HSM で基本的なオペレーションを実行します。

JCE プロバイダーのインストールを検証するには

1. (オプション) 使用環境に Java がインストール済みでない場合は、次のコマンドを使用してインストールします。

   Linux (and compatible libraries)

   ```
   $ sudo yum install java-1.8.0-openjdk
   ```

   Ubuntu

   ```
   $ sudo apt-get install openjdk-8-jre
   ```

2. 次のコマンドを使用して、必要な環境変数を設定します。*<HSM user name>* と *<password>* では、crypto user (CU) の認証情報に置き換えます。

   ```
   $ export LD_LIBRARY_PATH=/opt/cloudhsm/lib
   ```

   ```
   $ export HSM_PARTITION=PARTITION_1
   ```

   ```
   $ export HSM_USER=<HSM user name>
   ```

   ```
   $ export HSM_PASSWORD=<password>
   ```

3. 基本的な機能のテストを実行するには、次のコマンドを使用します。成功すると、コマンドの出力は次のようになります。

   ```
   $ java8 -classpath "/opt/cloudhsm/java/*" org.junit.runner.JUnitCore
    TestBasicFunctionality

   JUnit version 4.11
   .2018-08-20 17:53:48,514 DEBUG [main] TestBasicFunctionality
    (TestBasicFunctionality.java:33) - Adding provider.
   2018-08-20 17:53:48,612 DEBUG [main] TestBasicFunctionality
    (TestBasicFunctionality.java:42) - Logging in.
   2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:104) -
    Looking for credentials in HsmCredentials.properties
   2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:122) -
    Looking for credentials in System.properties
   ```

```
2018-08-20 17:53:48,613 INFO [main] cfm2.LoginManager (LoginManager.java:130) -
 Looking for credentials in System.env
 SDK Version: 2.03
2018-08-20 17:53:48,655 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:54) - Generating AES Key with key size 256.
2018-08-20 17:53:48,698 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:63) - Encrypting with AES Key.
2018-08-20 17:53:48,705 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:84) - Deleting AES Key.
2018-08-20 17:53:48,707 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:92) - Logging out.

Time: 0.205

OK (1 test)
```

JCE プロバイダへの認証情報の提供

HSM では、アプリケーションがそれらを使用する前に、Java アプリケーションを認証する必要があります。アプリケーションごとに 1 つのセッションを使用できます。HSM は、明示的なログインと暗黙的なログイン方法のいずれかを使用して、セッションを認証します。

Explicit login - この方法では、CloudHSM 認証情報をアプリケーションに直接渡すことができます。また、`LoginManager.login()` メソッドを使用します。ここで、CU ユーザー名、パスワード、HSM パーティション ID を渡します。明示的なログイン方法の使用の詳細については、「HSM へのログイン」のサンプルコードを参照してください。

Implicit login - この方法では、CloudHSM 認証情報を、新しいプロパティファイルまたはシステムプロパティで設定するか、環境変数として設定することができます。

- New property file - `HsmCredentials.properties` という名前の新しいファイルを作成し、そのファイルをアプリケーションの CLASSPATH に追加します。ファイルには次の内容が含まれます。

```
HSM_PARTITION = PARTITION_1
HSM_USER = <HSM user name>
HSM_PASSWORD = <password>
```

- System properties - アプリケーションの実行時に、システムプロパティを通して認証情報を設定します。次の例は、これを行うための 2 つの異なる方法を示しています。

```
$ java -DHSM_PARTITION=PARTITION_1 -DHSM_USER=<HSM user name> -
DHSM_PASSWORD=<password>
```

```
System.setProperty("HSM_PARTITION","PARTITION_1");
System.setProperty("HSM_USER","<HSM user name>");
System.setProperty("HSM_PASSWORD","<password>");
```

- Environment variables - 認証情報を環境変数として設定します。

```
$ export HSM_PARTITION=PARTITION_1
$ export HSM_USER=<HSM user name>
$ export HSM_PASSWORD=<password>
```

アプリケーションで設定されない場合、または HSM でセッションを認証する前にユーザーが操作を行った場合は、認証情報を使用できない場合があります。このような場合は、Java 用の CloudHSM ソフトウェアライブラリによって、次の順序で認証情報が検索されます。

1. `HsmCredentials.properties`
2. システムプロパティ
3. 環境変数

エラー処理

暗黙的なログインよりも明示的なログインの方が、簡単にエラーを処理することができます。`LoginManager` クラスを使用すると、アプリケーションが障害に対応する方法をより細かく制御できます。暗黙的なログイン方法では、認証情報が無効な場合や、HSM でのセッションの認証に問題が発生したタイミングをエラー処理で把握するのが難しくなります。

JCE プロバイダー中のキー管理の基本

JCE プロバイダー中のキー管理の基本には、キーのインポート、キーのエクスポート、ハンドルによるキーのロード、またはキーの削除などがあります。キーの管理の詳細については、「キーの管理」のサンプルコードを参照してください。

また、JCE プロバイダーののサンプルコードについては、コードサンプル で参照できます。

Client SDK 3 のためのサポートメカニズム

でサポートされている Java 暗号化アーキテクチャ (JCA) インターフェイスとエンジンクラスについては AWS CloudHSM、以下のトピックを参照してください。

トピック

- サポートされるキー
- サポートされる暗号
- サポートされているダイジェスト
- サポートされている Hash-based Message Authentication Code (HMAC) アルゴリズム
- サポートされている署名/検証メカニズム
- メカニズムの注釈

サポートされるキー

Java 用の AWS CloudHSM ソフトウェアライブラリでは、次のキータイプを生成できます。

- AES – 128、192、256 ビットの AES キー。
- DESede — 92 ビット 3DES キー。今後の変更については、以下の注記「1」を参照してください。
- NIST 曲線 secp256r1 (P-256)、secp384r1 (P-384)、および secp256k1 (ブロックチェーン) を対象とした ECC キーペア。
- RSA – 2048 ~ 4096 ビットの RSA キー (256 ビットの増分)。

標準のパラメータに加えて、生成されるキーごとに以下のパラメータがサポートされています。

- Label: キーの検索に使用できるキーラベル。
- isExtractable: キーを HSM からエクスポートできるかどうかを示します。
- isPersistent: 現在のセッションの終了後、キーが HSM に残るかどうかを示します。

> **Note**
>
> Java ライブラリバージョン 3.1 では、パラメータをより詳細に指定することができます。詳細については、「サポートされている Java 属性」を参照してください。

サポートされる暗号

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のアルゴリズム、モード、パディングの組み合わせをサポートしています。

アルゴリズム	モード	[Padding] (パディング)	メモ
AES	CBC	`AES/CBC/NoPadding` `AES/CBC/PKCS5Padding`	`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。
AES	ECB	`AES/ECB/NoPadding` `AES/ECB/PKCS5Padding`	`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。AES 変換を使用します。
AES	CTR	`AES/CTR/NoPadding`	`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。
AES	GCM	`AES/GCM/NoPadding`	`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE`、`Cipher.WRAP_MODE` および `Cipher.UNWRAP_MODE` を実装します。

アルゴリズム	モード	[Padding] (パディング)	メモ
			AES-GCM 暗号化の実行時に、HSM はリクエスト内の初期化ベクトル (IV) を無視し、独自に IV を生成して使用します。オペレーションが完了したら、`Cipher.getIV()` を呼び出して IV を取得する必要があります。

AESWrap	ECB	`AESWrap/ECB/ZeroPadding`
		`AESWrap/ECB/NoPadding`
		`AESWrap/ECB/PKCS5Padding`
			`Cipher.WRAP_MODE` および `Cipher.UNWRAP_MODE` を実装します。AES 変換 を使用します。

アルゴリズム	モード	[Padding] (パディング)	メモ

DESede (Triple DES)	CBC	`DESede/CBC/NoPadding`
		
		`DESede/CBC/PKCS5Padding`
			`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。
			
			キー生成ルーチンは 168 ビットまたは 192 ビットのサイズを受け入れます。ただし、内部的に、すべての DESede キーは 192 ビットです。
			
			今後の変更については、以下の注記「1」を参照してください。

アルゴリズム	モード	[Padding] (パディング)	メモ

DESede (Triple DES)	ECB	`DESede/ECB/NoPadding`
		
		`DESede/ECB/PKCS5Padding`
			`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。
			
			キー生成ルーチンは 168 ビットまたは 192 ビットのサイズを受け入れます。ただし、内部的に、すべての DESede キーは 192 ビットです。
			
			今後の変更については、以下の注記「1」を参照してください。

RSA	ECB	`RSA/ECB/NoPadding`
		
		`RSA/ECB/PKCS1Padding`
			`Cipher.ENCRYPT_MODE` および `Cipher.DECRYPT_MODE` を実装します。
			
			今後の変更については、以下の注記「1」を参照してください。

アルゴリズム	モード	[Padding] (パディング)	メモ
RSA	ECB	RSA/ECB/OAEPPadding RSA/ECB/OAEPWithSHA-1ANDMGF1Padding RSA/ECB/OAEPWithSHA-224ANDMGF1Padding RSA/ECB/OAEPWithSHA-256ANDMGF1Padding RSA/ECB/OAEPWithSHA-384ANDMGF1Padding RSA/ECB/OAEPWithSHA-512ANDMGF1Padding	Cipher.ENCRYPT_MODE 、Cipher.DECRYPT_MODE 、Cipher.WRAP_MODE 、および Cipher.UNWRAP_MODE を実装します。 OAEPPadding は、SHA-1 パディングタイプの OAEP です。
RSAAESWrap	ECB	OAEPPADDING	Cipher.WRAP_Mode および Cipher.UNWRAP_MODE を実装します。

サポートされているダイジェスト

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のメッセージダイジェストをサポートしています。

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

> **Note**
>
> 16 KB 未満のデータは HSM でハッシュされ、それ以上のデータはソフトウェアでローカルにハッシュされます。

サポートされている Hash-based Message Authentication Code (HMAC) アルゴリズム

Java 用の AWS CloudHSM ソフトウェアライブラリは、次の HMAC アルゴリズムをサポートしています。

- HmacSHA1
- HmacSHA224
- HmacSHA256
- HmacSHA384
- HmacSHA512

サポートされている署名/検証メカニズム

Java 用の AWS CloudHSM ソフトウェアライブラリは、次のタイプの署名と検証をサポートしています。

RSA 署名タイプ

- NONEwithRSA
- SHA1withRSA

- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA
- SHA1withRSA/PSS
- SHA224withRSA/PSS
- SHA256withRSA/PSS
- SHA384withRSA/PSS
- SHA512withRSA/PSS

ECDSA 署名タイプ

- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA

メカニズムの注釈

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

Client SDK 3 の Java キー属性

このトピックでは、Java ライブラリバージョン 3.1 の独自の拡張機能を使用して、キー属性を設定する方法について説明します。この拡張機能を使用して、これらのオペレーション中にサポートされるキー属性とその値を設定します。

- キー生成
- キーのインポート
- キーのラップ解除

> **Note**
>
> カスタムキー属性を設定するための拡張機能は、オプション機能です。Java ライブラリバージョン 3.0 で機能するコードがすでにある場合、そのコードを変更する必要はありません。作成したキーには、以前と同じ属性が引き続き含まれます。

トピック

属性について

キー属性を使用して、パブリックキー、プライベートキー、シークレットキーなど、キーオブジェクトで許可されるアクションを指定します。キー属性と値は、キーオブジェクトの作成オペレーション中に定義します。

ただし、Java Cryptography Extension (JCE) では、キー属性に値を設定する方法が指定されていないため、ほとんどのアクションがデフォルトで許可されていました。これに対して、PKCS # 11 標準では、より制限の厳しいデフォルトのある包括的な属性のセットが定義されています。Java ライブラリバージョン 3.1 以降、CloudHSM は、一般的に使用される属性に対してより制限の厳しい値を設定できる独自の拡張機能を提供します。

サポートされている属性

次の表に示す属性の値を設定できます。ベストプラクティスとして、制限する属性の値のみを設定してください。値を指定しない場合、CloudHSM は次の表で指定されたデフォルト値を使用します。デフォルト値の列のセルが空の場合は、属性に割り当てられている特定のデフォルト値がないことを示します。

属性	デフォルト値		メモ
	対称キー	キーペアのパブリックキー	キーペアのプライベートキー	

属性	デフォルト値			メモ
CKA_TOKEN	FALSE	FALSE	FALSE	クラスター内のすべての HSM にレプリケートされ、バックアップに含まれる永続的なキー。CKA_TOKEN = FALSE は、セッションキーを意味します。セッションキーは 1 つの HSM にのみロードされ、HSM への接続が切断されると自動的に消去されます。
CKA_LABEL				ユーザー定義の文字列。これにより、HSM のキーを簡単に識別できます。
CKA_EXTRACTABLE	TRUE		TRUE	True は、このキーを HSM からエクスポートできることを示します。
CKA_ENCRYPT	TRUE	TRUE		True は、キーを使用して任意のバッファを暗号化できることを示します。

属性	デフォルト値			メモ
CKA_DECRYPT	TRUE		TRUE	True は、キーを使用して任意のバッファを復号できることを示します。通常は、CKA_WRAP が true に設定されているキーに対して、これを FALSE に設定します。
CKA_WRAP	TRUE	TRUE		True は、キーを使用して別のキーをラップできることを示します。通常、プライベートキーの場合、これを FALSE に設定します。
CKA_UNWRAP	TRUE		TRUE	True は、キーを使用して別のキーをラップ解除 (インポート) できることを示します。

属性	デフォルト値			メモ
CKA_SIGN	TRUE		TRUE	True は、キーを使用してメッセージダイジェストに署名できることを示します。パブリックキーおよびアーカイブしたプライベートキーの場合、通常 FALSE に設定されます。
CKA_VERIFY	TRUE	TRUE		True は、キーを使用して署名を検証できることを示します。これは通常、プライベートキーの場合、FALSE に設定されます。

属性	デフォルト値			メモ
CKA_PRIVATE	TRUE	TRUE	TRUE	True は、ユーザーが認証されるまで、ユーザーがキーにアクセスできないことを示します。わかりやすくするために、この属性が FALSE に設定されている場合でも、ユーザーが認証されるまで CloudHSM のどのキーにもアクセスできません。

> **Note**
>
> PKCS #11 ライブラリでは、より広範な属性がサポートされます。詳細については、「サポートされている PKCS #11 属性」を参照してください。

キーの属性設定

CloudHsmKeyAttributesMap は Java Map のようなオブジェクトで、キーオブジェクトの属性値を設定するために使用できます。CloudHsmKeyAttributesMap 関数のメソッドは、Java マップ操作のメソッドと同様です。

属性にカスタム値を設定するには、次の 2 つのオプションがあります。

- 次の表に示す方法を使用します。
- このドキュメントの後半で説明するビルダーパターンの使用

属性マップオブジェクトは、属性を設定するための次のメソッドをサポートしています。

操作	戻り値	**CloudHSMKeyAttributesMap** 方法
既存のキーのキー属性の値を取得する	オブジェクト (値を含む) または null	get(keyAttribute)
1 つのキー属性の値を入力します。	キー属性のマッピングがなかった場合、キー属性に関連付けられた以前の値、または null	put(keyAttribute, value)
複数のキー属性の値を設定する	該当なし	putAll (keyAttributesMap)
属性マップからキーと値のペアを削除する	キー属性のマッピングがなかった場合、キー属性に関連付けられた以前の値、または null	remove(keyAttribute)

> **Note**
>
> 明示的に指定しない属性は、上記の the section called “サポートされている属性” の表に示したデフォルトに設定されます。

ビルダーパターンの例

通常、開発者にとっては、ビルダーパターンを介してクラスを利用する方がより便利です。例:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

CloudHsmKeyAttributesMap keyAttributesSessionDecryptionKey =
   new CloudHsmKeyAttributesMap.Builder()
      .put(CloudHsmKeyAttributes.CKA_LABEL, "ExtractableSessionKeyEncryptDecrypt")
      .put(CloudHsmKeyAttributes.CKA_WRAP, false)
```

```
      .put(CloudHsmKeyAttributes.CKA_UNWRAP, false)
      .put(CloudHsmKeyAttributes.CKA_SIGN, false)
      .put(CloudHsmKeyAttributes.CKA_VERIFY, false)
      .build();

CloudHsmKeyAttributesMap keyAttributesTokenWrappingKey =
   new CloudHsmKeyAttributesMap.Builder()
      .put(CloudHsmKeyAttributes.CKA_LABEL, "TokenWrappingKey")
      .put(CloudHsmKeyAttributes.CKA_TOKEN, true)
      .put(CloudHsmKeyAttributes.CKA_ENCRYPT, false)
      .put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
      .put(CloudHsmKeyAttributes.CKA_SIGN, false)
      .put(CloudHsmKeyAttributes.CKA_VERIFY, false)
      .build();
```

開発者は、キーテンプレートのベストプラクティスを実施するための便利な方法として、事前に定義された属性セットを利用することもできます。例:

```
//best practice template for wrapping keys

CloudHsmKeyAttributesMap commonKeyAttrs = new CloudHsmKeyAttributesMap.Builder()
    .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, false)
    .put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
    .build();

// initialize a new instance of CloudHsmKeyAttributesMap by copying commonKeyAttrs
// but with an appropriate label

CloudHsmKeyAttributesMap firstKeyAttrs = new CloudHsmKeyAttributesMap(commonKeyAttrs);
firstKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, "key label");

// alternatively, putAll() will overwrite existing values to enforce conformance

CloudHsmKeyAttributesMap secondKeyAttrs = new CloudHsmKeyAttributesMap();
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_DECRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_ENCRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, “safe wrapping key”);
secondKeyAttrs.putAll(commonKeyAttrs); // will overwrite CKA_DECRYPT to be FALSE
```

キーペアの属性設定

Java クラス CloudHsmKeyPairAttributesMap を使用して、キーペアのキー属性を処理します。CloudHsmKeyPairAttributesMap は、2 つの CloudHsmKeyAttributesMap オブジェクトをカプセル化します。1 つはパブリックキー用ともう 1 つはプライベートキー用です。

パブリックキーとプライベートキーの個々の属性を個別に設定するには、そのキーの対応する CloudHsmKeyAttributes マップオブジェクトで put() メソッドを使用できます。getPublic() メソッドを使用してパブリックキーの属性マップを取得し、getPrivate() を使用してプライベートキーの属性マップを取得します。引数としてキーペア属性マップを使用する putAll() を使用して、パブリックキーペアとプライベートキーペアの両方に、複数のキー属性の値を一緒に入力します。

ビルダーパターンの例

通常、開発者にとっては、ビルダーパターンを介してキー属性を設定する方がより便利です。例:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

//specify attributes up-front
CloudHsmKeyAttributesMap keyAttributes =
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_SIGN, false)
        .put(CloudHsmKeyAttributes.CKA_LABEL, "PublicCertSerial12345")
        .build();

CloudHsmKeyPairAttributesMap keyPairAttributes =
    new CloudHsmKeyPairAttributesMap.Builder()
        .withPublic(keyAttributes)
        .withPrivate(
            new CloudHsmKeyAttributesMap.Builder() //or specify them inline
                .put(CloudHsmKeyAttributes.CKA_LABEL, "PrivateCertSerial12345")
                .put (CloudHSMKeyAttributes.CKA_WRAP, FALSE)
                .build()
        )
        .build();
```

> **Note**
>
> この独自の拡張機能の詳細については、「」の「Javadoc アーカイブとサンプル」を参照してください GitHub。Javadoc を調べるには、アーカイブをダウンロードして展開します。

まとめ

キーオペレーションでキー属性を指定するには、次の手順に従います。

1. 対称キーの CloudHsmKeyAttributesMap、またはキーペアの CloudHsmKeyPairAttributesMap をインスタンス化します。
2. 必要なキー属性と値を使用して、ステップ 1 からの属性オブジェクトを定義します。
3. 特定のキータイプに対応する Cavium*ParameterSpec クラスをインスタンス化し、この設定された属性オブジェクトをコンストラクタに渡します。
4. この Cavium*ParameterSpec オブジェクトを対応する暗号クラスまたはメソッドに渡します。

参考のために、次の表に、カスタムキー属性をサポートする Cavium*ParameterSpec クラスとメソッドを示します。

キータイプ	パラメータ仕様クラス	コンストラクタの例
基本クラス	CaviumKeyGenAlgorithmParameterSpec	CaviumKeyGenAlgorithmParameterSpec(CloudHsmKeyAttributesMap keyAttributesMap)
DES	CaviumDESKeyGenParameterSpec	CaviumDESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap keyAttributesMap)

キータイプ	パラメータ仕様クラス	コンストラクタの例
RSA	CaviumRSAKeyGenParameterSpec	CaviumRSAKeyGenParameterSpec(int keysize, BigInteger publicExponent, CloudHsmKeyPairAttributesMap keyPairAttributesMap)
シークレット	CaviumGenericSecretKeyGenParameterSpec	CaviumGenericSecretKeyGenParameterSpec(int size, CloudHsmKeyAttributesMap key AttributesMap)
AES	CaviumAESKeyGenParameterSpec	CaviumAESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap key AttributesMap)
EC	CaviumECGenParameterSpec	CaviumECGenParameterSpec(String stdName, CloudHsmKeyPairAttributesMap keyPairAttributesMap)

サンプルコード: キーの生成とラップ

次の簡単なコードサンプルは、キー生成とキーラップの 2 つの異なるオペレーションの手順を示しています。

```
// Set up the desired key attributes
```

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES", "Cavium");
CaviumAESKeyGenParameterSpec keyAttributes = new CaviumAESKeyGenParameterSpec(
    256,
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_LABEL, "MyPersistentAESKey")
        .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, true)
        .put(CloudHsmKeyAttributes.CKA_TOKEN, true)
        .build()
);

// Assume we already have a handle to the myWrappingKey
// Assume we already have the wrappedBytes to unwrap

// Unwrap a key using Custom Key Attributes

CaviumUnwrapParameterSpec unwrapSpec = new
 CaviumUnwrapParameterSpec(myInitializationVector, keyAttributes);

Cipher unwrapCipher = Cipher.getInstance("AESWrap", "Cavium");
unwrapCipher.init(Cipher.UNWRAP_MODE, myWrappingKey, unwrapSpec);
Key unwrappedKey = unwrapCipher.unwrap(wrappedBytes, "AES", Cipher.SECRET_KEY);
```

Java for Client SDK 3 の AWS CloudHSM ソフトウェアライブラリのコードサンプル

前提条件

サンプルを実行する前に、環境をセットアップする必要があります。

- Java Cryptographic Extension (JCE) provider と AWS CloudHSM client package をインストールします。
- 有効な HSM ユーザー名とパスワードを設定します。これらのタスクには、暗号化ユーザー (CU) のアクセス権限で十分です。アプリケーションは、それぞれの例でこの認証情報を使用して HSM にログインします。
- JCE provider へのクレデンシャルを提供する方法を決定します。

コードサンプル

次のコードサンプルでは、基本タスクを実行するために、AWS CloudHSM JCE provider を使用する方法を示します。その他のコードサンプルは、 で入手できますGitHub。

- HSM へのログイン
- キーの管理
- AES キーの生成
- AES GCM による暗号化と復号
- Encrypt and decrypt with AES-CTR
- D3DES-ECB による暗号化と復号[注記 1 参照]
- AES-GCM を使用したキーのラップとラップ解除
- AES を使用したキーのラップとラップ解除
- RSA を使用したキーのラップとラップ解除
- サポートされているキー属性の使用
- キーストアのキーの列挙
- CloudHSM キーストアの使用
- マルチスレッドでのメッセージの署名のサンプル
- Sign and Verify with EC Keys

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

Client SDK 3 での AWS CloudHSM KeyStore Java クラスの使用

クラスは、keytool AWS CloudHSM `KeyStore`や jarsigner などのアプリケーションを介して AWS CloudHSM キーへのアクセスを許可する、専用 PKCS12 キーストアを提供します。このキーストアでは、証明書をキーデータとともに保存し、 AWS CloudHSMに保存されているキーデータに関連付けることができます。

> **Note**
>
> 証明書は公開情報であり、暗号化キーのストレージ容量を最大化するため、 AWS CloudHSM は HSMsへの証明書の保存をサポートしていません。

クラスは、Java AWS CloudHSM `KeyStore`Cryptography Extension (JCE) の`KeyStore`サービスプロバイダーインターフェイス (SPI) を実装します。の使用の詳細については`KeyStore`、「クラス KeyStore」を参照してください。

適切なキーストアの選択

AWS CloudHSM Java 暗号化拡張 (JCE) プロバイダーには、すべてのトランザクションを HSM に渡すデフォルトのパススルー読み取り専用キーストアが付属しています。このデフォルトのキーストアは、専用 とは異なります AWS CloudHSM KeyStore。ほとんどの場合、デフォルトを使用することにより、ランタイムのパフォーマンスとスループットが向上します。は、HSM AWS CloudHSM KeyStore へのキーオペレーションのオフロードに加えて、証明書と証明書ベースのオペレーションのサポートが必要なアプリケーションにのみ使用してください。

どちらのキーストアも操作に Cavium JCE プロバイダを使用しますが、これらは独立したエンティティであり、相互に情報を交換しません。

Java アプリケーションのデフォルトのキーストアを次のようにロードします。

```
KeyStore ks = KeyStore.getInstance("Cavium");
```

専用 CloudHSM KeyStore を次のようにロードします。

```
KeyStore ks = KeyStore.getInstance("CloudHSM")
```

初期化中 AWS CloudHSM KeyStore

JCE プロバイダーにログインするの AWS CloudHSM KeyStore と同じ方法で にログインします。環境変数またはシステムプロパティファイルのいずれかを使用できます。CloudHSM KeyStoreの使用を開始する前にログインする必要があります。JCE プロバイダーを使用して HSM にログインする例については、Login to an HSM を参照してください。

必要に応じて、パスワードを指定して、キーストアデータを保持するローカル PKCS12 ファイルを暗号化できます。 AWS CloudHSM キーストアを作成するときは、ロード、設定、取得の方法を使用するときにパスワードを設定し、指定します。

次のように新しい CloudHSM KeyStore オブジェクトをインスタンス化します。

```
ks.load(null, null);
```

`store` メソッドを使用して、キーストアデータをファイルに書き込みます。その後は、次のように、ソースファイルとパスワードを使用し、`load` メソッドを使用して既存のキーストアをロードできます。

```
ks.load(inputStream, password);
```

の使用 AWS CloudHSM KeyStore

CloudHSM KeyStore オブジェクトは、通常、jarsigner や keytool などのサードパーティーアプリケーションを通じて使用されます。コードを使用してオブジェクトに直接アクセスすることもできます。

AWS CloudHSM KeyStore は JCE クラスKeyStore仕様に準拠しており、以下の機能を提供します。

- `load`

  指定された入力ストリームからキーストアをロードします。キーストアの保存時にパスワードが設定されている場合、ロードを成功させるには、この同じパスワードを指定する必要があります。新しい空のキーストアを初期化するには、両方のパラメータを null に設定します。

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  ks.load(inputStream, password);
  ```

- `aliases`

  指定されたキーストアインスタンス内に含まれるすべてのエントリのエイリアス名の列挙を返します。結果には、PKCS12 ファイルにローカルに保存されたオブジェクトと、HSM 上に存在するオブジェクトが含まれます。

  サンプルコード:

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  for(Enumeration<String> entry = ks.aliases(); entry.hasMoreElements();)
   {
    String label = entry.nextElement();
    System.out.println(label);
   }
  ```

- `ContainsAlias`

  キーストアが、指定されたエイリアスを持つ少なくとも 1 つのオブジェクトにアクセスできる場合は true を返します。キーストアは、PKCS12 ファイルにローカルに保存されているオブジェクトと、HSM 上に存在するオブジェクトをチェックします。

- `DeleteEntry`

ローカル PKCS12 ファイルから証明書エントリを削除します。HSM に保存されているキーデータの削除は、 ではサポートされていません AWS CloudHSM KeyStore。CloudHSM の key_mgmt_util ツールを使用してキーを削除できます。

- `GetCertificate`

  使用可能な場合、エイリアスに関連付けられた証明書を返します。エイリアスが存在しないか、証明書ではないオブジェクトを参照している場合、関数は NULL を返します。

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
Certificate cert = ks.getCertificate(alias)
```

- `GetCertificateAlias`

  指定された証明書とデータが一致する最初のキーストアエントリの名前 (エイリアス) を返します。

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
String alias = ks.getCertificateAlias(cert)
```

- `GetCertificateChain`

  指定されたエイリアスに関連付けられた証明書チェーンを返します。エイリアスが存在しないか、証明書ではないオブジェクトを参照している場合、関数は NULL を返します。

- `GetCreationDate`

  指定されたエイリアスによって識別されるエントリの作成日を返します。作成日が使用できない場合、この関数は証明書が有効になった日付を返します。

- `GetKey`

  GetKey は HSM に渡され、指定されたラベルに対応するキーオブジェクトを返します。HSM getKeyに直接クエリを実行すると、 によって生成されたかどうかにかかわらず、HSM 上の任意のキーに使用できます KeyStore。

```
Key key = ks.getKey(keyLabel, null);
```

- `IsCertificateEntry`

  指定されたエイリアスを持つエントリが証明書エントリを表すかどうかをチェックします。

- `IsKeyEntry`

指定されたエイリアスを持つエントリがキーエントリを表すかどうかをチェックします。このアクションは、PKCS12 ファイルと HSM の両方でエイリアスを検索します。

- `SetCertificateEntry`

  指定された証明書を指定されたエイリアスに割り当てます。指定されたエイリアスがキーまたは証明書の識別にすでに使用されている場合は、`KeyStoreException` がスローされます。JCE コードを使用してキーオブジェクトを取得し、 メソッドを使用して KeyStore `SetKeyEntry`証明書をキーに関連付けることができます。

- `byte[]` キーのある `SetKeyEntry`

  この API は現在、Client SDK 3 ではサポートされていません。

- `Key` オブジェクトのある `SetKeyEntry`

  指定されたキーを指定されたエイリアスに割り当て、HSM 内に保存します。`Key` オブジェクトが `CaviumKey` のタイプでない場合、キーは抽出可能なセッションキーとして HSM にインポートされます。

  `Key` オブジェクトが `PrivateKey` のタイプの場合、対応する証明書チェーンが添付されている必要があります。

  エイリアスが既に存在する場合、`SetKeyEntry` 呼び出しは `KeyStoreException` をスローし、キーが上書きされるのを防ぎます。キーを上書きする必要がある場合は、そのために KMU または JCE を使用します。

- `EngineSize`

  キーストア内のエントリの数を返します。

- `Store`

  キーストアを指定された出力ストリームに PKCS12 ファイルとして保存し、指定されたパスワードで保護します。さらに、ロードされたすべてのキー (`setKey` 呼び出しを使用して設定される) が保持されます。

AWS CloudHSMとサードパーティアプリケーションの統合

のユースケースには、サードパーティーのソフトウェアアプリケーションを AWS CloudHSM クラスター内の HSM と統合すること AWS CloudHSM が含まれます。サードパーティーのソフトウェアをと統合することで AWS CloudHSM、さまざまなセキュリティ関連の目標を達成できます。以下のトピックでは、これらの目標のいくつかを達成する方法について説明します。

トピック

- での SSL/TLS オフロードによるウェブサーバーセキュリティの向上 AWS CloudHSM
- AWS CloudHSMを使用した認証機関 (CA) として Windows Server を設定する
- AWS CloudHSMでの Oracle Database の透過的なデータ暗号化 (TDE)
- SignTool で Microsoft を使用してファイルに署名 AWS CloudHSM する
- Java キーツールとJarsigner
- その他のサードパーティベンダーとの統合

での SSL/TLS オフロードによるウェブサーバーセキュリティの向上 AWS CloudHSM

ウェブサーバーとそのクライアント (ウェブブラウザ) では、Secure Sockets Layer (SSL) または Transport Layer Security (TLS) プロトコルを使用して、ウェブサーバーのアイデンティティを確認し、インターネット上でウェブページやその他のデータを送受信するための安全な接続を確立するための安全な接続を確立するものです。これは HTTPS として知られています。このウェブサーバーでは、パブリック/プライベートのキーペアと SSL/TLS パブリックキー証明書を使用して、各クライアントとの HTTPS セッションを確立します。このプロセスにはウェブサーバーの多くの計算が含まれますが、その一部を SSL アクセラレーションと呼ばれる AWS CloudHSM クラスターにオフロードできます。オフロードすることで、ウェブサーバーの計算負荷が軽減され、サーバーのプライベートキーを HSM に保存することでセキュリティが強化されます。

以下のトピックでは、 での SSL/TLS オフロードの AWS CloudHSM 仕組みの概要と、以下のプラットフォーム AWS CloudHSM で で SSL/TLS オフロードを設定するためのチュートリアルについて説明します。

Linux の場合は、NGINX または Apache HTTP Server Web サーバーソフトウェアで OpenSSL Dynamic Engine を使用します

Windows の場合、Windows Server の Internet Information Services (IIS) のウェブサーバーソフトウェアを使用します

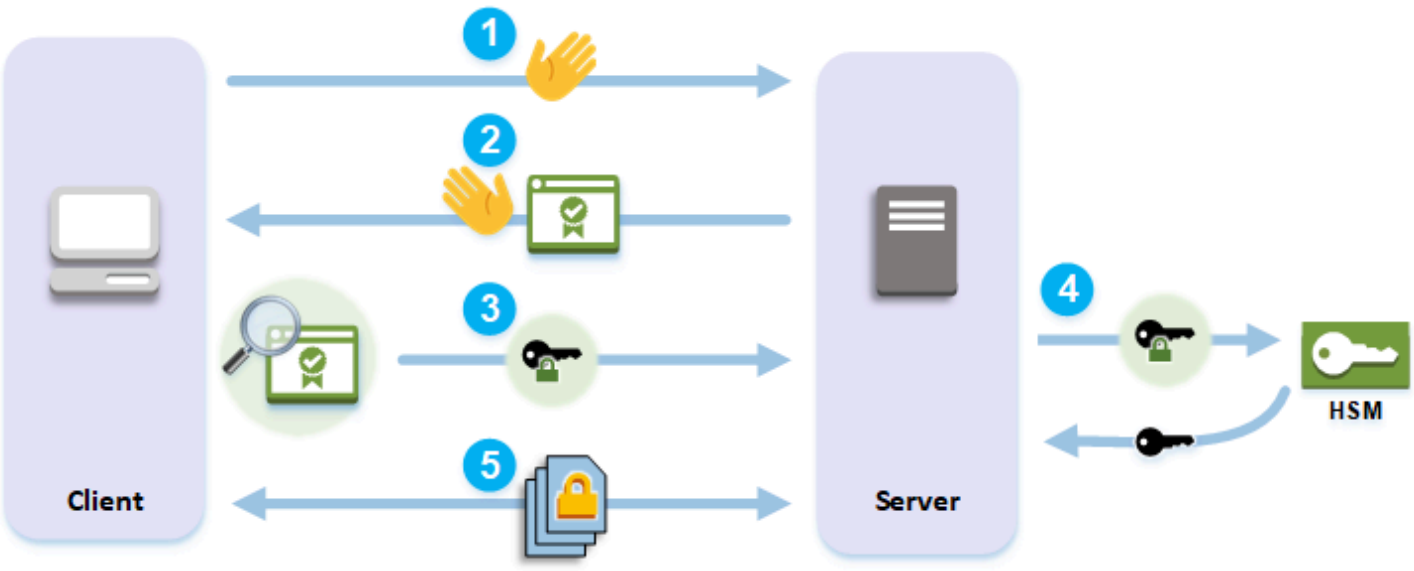
トピック

- での SSL/TLS オフロードの AWS CloudHSM 仕組み
- Linux 上の SSL/TLS オフロード
- Windows での SSL/TLS オフロードに IIS と CNG の使用
- Elastic Load Balancing を使用してロードバランサーを追加する (オプション)

での SSL/TLS オフロードの AWS CloudHSM 仕組み

HTTPS 接続を確立するため、ウェブサーバーはクライアントとハンドシェークプロセスを実行します。次の図に示すように、このプロセスの一環として、サーバーはいくつかの暗号化処理を HSM にオフロードします。プロセスの各ステップについて、図の下に説明があります。

> **Note**
>
> 次のイメージとプロセスでは、サーバーの検証とキーの交換に RSA を使用することを想定しています。RSA の代わりに Diffie-Hellman を使用する場合、手順が若干異なります。



1. クライアントはサーバーに Hello メッセージを送信します。
2. サーバーは Hello メッセージで応答し、サーバーの証明書を送信します。

3. クライアントは以下のアクションを実行します。
   a. SSL/TLS サーバーの証明書がクライアントの信頼するルート証明書により署名されていることを確認します。
   b. サーバーの証明書からパブリックキーを抽出します。
   c. プリマスタシークレットを生成し、サーバーのパブリックキーで暗号化します。
   d. 暗号化されたプリマスターシークレットをサーバーに送信します。
4. クライアントのプリマスターシークレットを復号するため、サーバーはそれを HSM に送信します。HSM は、HSM のプライベートキーを使用してプリマスターシークレットを復号し、プリマスターシークレットをサーバーに送信します。クライアントとサーバーはそれぞれ独立してプリマスターシークレットおよび Hello メッセージからのいくつかの情報を使用して、マスターシークレットを計算します。
5. ハンドシェイクプロセスは終了します。残りのセッションでは、クライアントとサーバーの間で送信されたすべてのメッセージは、マスターシークレットのデリバティブで暗号化されます。

で SSL/TLS オフロードを設定する方法については AWS CloudHSM、次のいずれかのトピックを参照してください。

- Linux 上の SSL/TLS オフロード
- Windows での SSL/TLS オフロードに IIS と CNG の使用

Linux 上の SSL/TLS オフロード

を使用すると AWS CloudHSM、NGINX、Apache、Tomcat を使用して Linux で SSL/TLS オフロードを実行できます。詳細については、以下の関連トピックを参照してください。

トピック

- Linux での SSL/TLS オフロードへの NGINX または OpenSSL 搭載の Apache の使用
- Linux での SSL/TLS オフロードに JSSE で Tomcat を使う

Linux での SSL/TLS オフロードへの NGINX または OpenSSL 搭載の Apache の使用

このトピックでは、Linux ウェブサーバーで で AWS CloudHSM SSL/TLS オフロードを設定する step-by-step 手順について説明します。

トピック

概要

Linux では、NGINX と Apache HTTP サーバーのウェブサーバーソフトウェアは OpenSSL とネイティブに統合して HTTPS をサポートします。OpenSSL 用AWS CloudHSM 動的エンジンには、暗号化オフロードとキーストレージ用に、ウェブサーバーソフトウェアがクラスターの HSM を使用することを許可するインターフェイスが用意されています。OpenSSL エンジンは、ウェブサーバーと AWS CloudHSM クラスターの橋渡しをします。

このチュートリアルを完了するには、まず、Linux 上で NGINX と Apache のどちらのウェブサーバーを使用するかを選択する必要があります。選択したら、チュートリアルに以下の方法が表示されます。

- Amazon EC2 インスタンスに、ウェブサーバーソフトウェアをインストールします。
- AWS CloudHSM クラスターに保存されている秘密キーで、HTTPS をサポートするようにウェブサーバソフトウェアを設定します。
- (オプション) Amazon EC2 を使用して 2 台目のウェブサーバーインスタンスを作成し、Elastic Load Balancing を使用してロードバランサーを作成します。ロードバランサーを使用すると、複数のサーバーに負荷を分散することでパフォーマンスを向上させることができます。また、1 つ以上のサーバーに障害が発生した場合、冗長性と高可用性を提供します。

始める準備ができたら、「ステップ 1: 前提条件の設定」を参照してください。

ステップ 1: 前提条件の設定

プラットフォームごとに、異なる前提条件が必要です。以下の前提条件セクションのうち、お使いのプラットフォームに合ったものをご利用ください。

トピック

- Client SDK 5 の前提条件
- クライアント SDK 3 の前提条件

Client SDK 5 の前提条件

クライアント SDK 5でウェブサーバー SSL/TLS オフロードを設定するには、以下が必要です。

- 少なくとも 2 つのハードウェアセキュリティモジュール (HSM) を持つアクティブな AWS CloudHSM クラスター

> **Note**
>
> HSM クラスターは1つでも使用できますが、まずクライアントキーの耐久性を無効にする必要があります。詳細については、クライアントキーの耐久性設定の管理 そして クライアント SDK 5 設定ツール を参照してください。

- Amazon EC2 インスタンスが Linux オペレーティングシステムを実行します。インスタンスに次のソフトウェアがインストールされていることを確認します。
  - ウェブサーバー (NGINX または Apache)
  - クライアント SDK 5 の OpenSSL 動的エンジン
- HSM でこのウェブサーバーのプライベートキーを所有および管理する Crypto User (CU)。

Linux ウェブサーバーインスタンスをセットアップし、HSM で CU を作成するには

1. 用の OpenSSL Dynamic Engine をインストールして設定します AWS CloudHSM。OpenSSL ダイナミックエンジンのインストールの詳細については、クライアント SDK 5 の OpenSSL 動的エンジン を参照してください。
2. クラスターにアクセスできる EC2 Linux インスタンスで、NGINX または Apache ウェブサーバーをインストールします。

   Amazon Linux

   - NGINX

     ```
     $ sudo yum install nginx
     ```

   - Apache

     ```
     $ sudo yum install httpd24 mod24_ssl
     ```

Amazon Linux 2

- Amazon Linux 2 で最新バージョンの NGINX をダウンロードする方法については、NGINX のウェブサイト を参照してください。

  Amazon Linux 2 で入手可能な NGINX の最新バージョンは、システムバージョンの OpenSSL よりも新しいバージョンの OpenSSL を使用しています。NGINX をインストールしたら、 AWS CloudHSM OpenSSL Dynamic Engine ライブラリから、このバージョンの OpenSSL が想定する場所へのシンボリックリンクを作成する必要があります。

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 7

- CentOS 7 で最新バージョンの NGINX をダウンロードする方法については、NGINX のウェブサイト を参照してください。

  CentOS 7 で入手可能な NGINX の最新バージョンは、システムバージョンの OpenSSL よりも新しいバージョンの OpenSSL を使用しています。NGINX をインストールしたら、AWS CloudHSM OpenSSL Dynamic Engine ライブラリから、このバージョンの OpenSSL が想定する場所へのシンボリックリンクを作成する必要があります。

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 7

- Red Hat 7 で最新バージョンの NGINX をダウンロードする方法については、NGINX のウェブサイト を参照してください。

  RedHat 7 で入手可能な NGINX の最新バージョンは、システムバージョンの OpenSSL よりも新しいバージョンの OpenSSL を使用しています。NGINX をインストールしたら、AWS CloudHSM OpenSSL Dynamic Engine ライブラリから、このバージョンの OpenSSL が想定する場所へのシンボリックリンクを作成する必要があります。

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Ubuntu 18.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 20.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 22.04

OpenSSL Dynamic Engine はまだサポートされていません。

3. CloudHSM CLI を使用して CU を作成します。HSM ユーザーの管理の詳細については、CloudHSM CLI を使用した HSM ユーザー管理について を参照してください。

> **Tip**
>
> CU のユーザー名とパスワードを書き留めます。後に、ウェブサーバーの HTTPS プライベートキーや証明書を生成またはインポートするときに必要になります。

以上のステップが完了したら、「ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする」に進みます。

メモ

- セキュリティ強化 Linux (SELinux) および Web サーバーを使用するには、クライアント SDK 5 が HSM と通信するために使用するポート 2223 でアウトバウンド TCP 接続を許可する必要があります。
- クラスターを作成してアクティブ化し、EC2 インスタンスにクラスターへのアクセス権を付与するには、AWS CloudHSMの使用開始 の手順を実行します。入門ガイドには、1 つの HSM と Amazon EC2 クライアントインスタンスを使用してアクティブなクラスターを作成する step-by-step 手順が記載されています。このクライアントインスタンスをウェブサーバーとして使用することができます。
- クライアントキーの耐久性を無効化しないようにするには、クラスターに複数の HSM を追加します。詳細については、「HSM の追加」を参照してください。
- クライアントインスタンスに接続するには、SSH または PuTTY を使用することができます。詳細については、「Amazon EC2 ドキュメント」の「SSH を使用した Linux インスタンスへの接続」または「PuTTY を使用した Windows から Linux インスタンスへの接続」を参照してください。

クライアント SDK 3 の前提条件

Client SDK 3 でウェブサーバー SSL/TLS オフロードを設定するには、以下が必要です。

- 少なくとも 1 つの HSM を持つアクティブな AWS CloudHSM クラスター。
- Amazon EC2 インスタンスが Linux オペレーティングシステムを実行します。インスタンスに次のソフトウェアがインストールされていることを確認します。
  - AWS CloudHSM クライアントおよびコマンドラインツール。
  - NGINX または Apache ウェブサーバーアプリケーション。
  - OpenSSL の AWS CloudHSM 動的エンジン。
- HSM でこのウェブサーバーのプライベートキーを所有および管理する Crypto User (CU)。

Linux ウェブサーバーインスタンスをセットアップし、HSM で CU を作成するには

1. 「開始」のステップを完了します。1 つの HSM と Amazon EC2 クライアントインスタンスを持つアクティブなクラスターが作成されます。EC2 インスタンスは、コマンドラインツールで構成されます。このクライアントインスタンスをウェブサーバーとして使用します。

2. クライアントインスタンスに接続します。詳細については、「Amazon EC2 ドキュメント」の「SSH を使用した Linux インスタンスへの接続」または「PuTTY を使用した Windows から Linux インスタンスへの接続」を参照してください。
3. クラスターにアクセスできる EC2 Linux インスタンスで、NGINX または Apache ウェブサーバーをインストールします。

   Amazon Linux

   - NGINX

     ```
     $ sudo yum install nginx
     ```

   - Apache

     ```
     $ sudo yum install httpd24 mod24_ssl
     ```

   Amazon Linux 2

   - NGINX バージョン 1.19 は Amazon Linux 2 のClient SDK 3 エンジンと互換性のある NGINX の最新バージョンです。

     詳細と NGINX バージョン 1.19 のダウンロードについては、NGINX のウェブサイト を参照してください。

   - Apache

     ```
     $ sudo yum install httpd mod_ssl
     ```

   CentOS 7

   - NGINX バージョン 1.19 は、CentOS 7 のClient SDK 3 エンジンと互換性のある NGINX の最新バージョンです。

     詳細と NGINX バージョン 1.19 のダウンロードについては、NGINX のウェブサイト を参照してください。

   - Apache

     ```
     $ sudo yum install httpd mod_ssl
     ```

Red Hat 7

- NGINX バージョン 1.19 は Red Hat 7 のClient SDK 3 エンジンと互換性のある NGINX の最新バージョンです。

  詳細と NGINX バージョン 1.19 のダウンロードについては、NGINX のウェブサイト を参照してください。

- Apache

```
$ sudo yum install httpd mod_ssl
```

Ubuntu 16.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 18.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

4. (オプション) 他の HSM をクラスターに追加します。詳細については、「HSM の追加」を参照してください。
5. cloudhsm_mgmt_util を使用して CU を作成します。詳細については、「HSM ユーザーの管理」を参照してください。CU のユーザー名とパスワードを書き留めます。後に、ウェブサーバーの HTTPS プライベートキーや証明書を生成またはインポートするときに必要になります。

以上のステップが完了したら、「ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする」に進みます。

ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする

HTTPS を有効にするには、ウェブサーバーアプリケーション (NGINX または Apache) にプライベートキーおよび対応する SSL/TLS 証明書が必要です。でウェブサーバーの SSL/TLS オフロードを使用するには AWS CloudHSM、プライベートキーを AWS CloudHSM クラスターの HSM に保存する必要があります。これを行うには、以下のいずれかの方法を使用できます。

- プライベートキーとそれに対応する証明書を持っていない場合、HSM でプライベートキーを生成できます。このプライベートキーを使用して証明書署名リクエスト (CSR) を作成し、それを使用してSSL/TLS証明書を作成します。
- プライベートキーおよび対応する証明書が既にある場合は、そのプライベートキーを HSM 内にインポートします。

前述の方法のどれを選択しても、HSM から フェイク PEM プライベートキー をエクスポートします。これは、HSM に保存されているプライベートキーへの参照を含む PEM 形式のプライベートキーファイルで、実際のプライベートキーではありません。ウェブサーバーは、SSL/TLS オフロード中にフェイク PEM プライベートキーファイルを使用して HSM のプライベートキーを識別します。

次のいずれかを行います。

- プライベートキーと証明書の生成
- 既存のプライベートキーと証明書をインポートします

プライベートキーと証明書の生成

プライベートキーの生成

このセクションでは、Client SDK 3 のキー管理ユーティリティ (KMU) を使用してキーペアを生成する方法を示します。HSM 内でキーペアを生成したら、それをフェイク PEM ファイルとしてエクスポートし、対応する証明書を生成できます。

キー管理ユーティリティ (KMU) を使用して生成されるプライベートキーは、Client SDK 3 およびクライアント SDK 5 の両方で使用できます。

キー管理ユーティリティ (KMU) のインストールと設定

1. クライアントインスタンスに接続します。
2. Client SDK 3 の インストールおよび設定。
3. 次のコマンドを実行してクライアントを起動します AWS CloudHSM 。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

4. 次のコマンドを使用して、key_mgmt_util コマンドラインツールを起動します。

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

5. 次のコマンドを実行して HSM にログインします。*<user name>* と *<password>* を、暗号化ユーザー (CU) のユーザー名とパスワードに置き換えます。

```
Command: loginHSM -u CU -s <user name> -p <password>>
```

プライベートキーの生成

ユースケースに応じて、RSA または EC キーペアを生成できます。次のいずれかを行います。

- HSM で RSA プライベートキーを生成するには

  genRSAKeyPair コマンドを使用して RSA キー ペアを生成します。この例では、モジュールが2048、公開指数が65537、ラベルが *tls_rsa_keypair* の RSA キーペアを生成します。

```
Command: genRSAKeyPair -m 2048 -e 65537 -l tls_rsa_keypair
```

  コマンドが成功すると、RSA キーペアが正常に生成されることを示す次のような出力が表示されます。

```
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

                Cfm3GenerateKeyPair:    public key handle: 7    private key handle: 8

                Cluster Status:
                Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

- HSM で EC プライベートキーを生成するには

genECCKeyPair コマンドを使用して EC キーペアを生成します。この例では、カーブ ID が 2 (NID_X9_62_prime256v1 カーブに対応) で、ラベルが *tls_ec_keypair* の EC キーペアを生成します。

```
Command: genECCKeyPair -i 2 -l tls_ec_keypair
```

コマンドが成功すると、EC キーペアが正常に生成されることを示す次のような出力が表示されます。

```
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

                Cfm3GenerateKeyPair:    public key handle: 7    private key handle: 8

                Cluster Status:
                Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

フェイク PEM プライベートキーファイルをエクスポート

HSM にプライベートキーを作成したら、フェイク PEM プライベートキーファイルをエクスポートする必要があります。このファイルには実際のキーデータは含まれていませんが、OpenSSL Dynamic Engine が HSM 上のプライベートキーを識別できるようにします。その後、プライベートキーを使用して証明書署名リクエスト (CSR) を作成し、CSR に署名して証明書を作成できます。

> **Note**
>
> キー管理ユーティリティ (KMU) を使用して生成されるフェイク PEM ファイルは、Client SDK 3 およびクライアント SDK 5 の両方で使用できます。

フェイク PEM としてエクスポートするキーに対応するキーハンドルを特定し、次のコマンドを実行してプライベートキーを偽の PEM 形式でエクスポートし、ファイルに保存します。以下の値は独自の値に置き換えてください。

- *<private_key_handle>* - 生成したプライベートキーのハンドルです。このハンドルは、前の手順のキー生成コマンドの 1 つによって生成されました。前の例で生成したプライベートキーのハンドルは 8 です。
- *<web_server_fake_PEM.key>* – フェイク PEM キーが書き込まれるファイルの名前。

```
Command: getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
```

Exit

次のコマンドを実行して key_mgmt_util を停止します。

```
Command: exit
```

これで、*<web_server_fake_PEM.key>* 前のコマンドで指定したパスに新しいファイルがシステム上に作成されることになります。このファイルはフェイク PEM プライベートキーファイルです。

自己署名証明書を生成します

フェイク PEM プライベートキーを生成したら、このファイルを使用して証明書署名リクエスト (CSR) と証明書を生成できます。

本稼働環境では、通常、認証機関 (CA) を使用して CSR から証明書を作成します。CA は、テスト環境では必要ありません。CA を使用する場合は、CA に CSR ファイルを送信し、HTTPS 用のウェブサーバーで提供される署名付き SSL/TLS 証明書を使用してください。

CA を使用する代わりに、 AWS CloudHSM OpenSSL Dynamic Engine を使用して自己署名証明書を作成できます。自己署名証明書はブラウザによって信頼されないため、本稼働環境では使用しないでください。これらは、テスト環境で使用することができます。

> **⚠ Warning**
>
> 自己署名証明書はテスト環境でのみ使用する必要があります。本稼働環境では、証明機関を使用して証明書を作成するなど、より安全な方法を使用してください。

OpenSSL Dynamic Engine をインストールして設定します

1. クライアントインスタンスに接続します。
2. インストールして設定を行うには、以下のいずれかを実行します。
   - the section called “OpenSSL Dynamic Engine のインストール”
   - the section called “OpenSSL Dynamic Engine”

証明書を生成する

1. 以前のステップで生成したフェイク PEM ファイルのコピーを入手します。
2. CSR を作成する

   次のコマンドを実行して、 AWS CloudHSM OpenSSL Dynamic Engine を使用して証明書署名リクエスト (CSR) を作成します。*<web_server_fake_PEM.key>* をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。*<web_server.csr>* を CSR が含まれるファイルの名前に置き換えます。

   `req` コマンドは対話的です。各フィールドに対応します。このフィールド情報は、SSL/TLS 証明書にコピーされます。

   ```
   $ openssl req -engine cloudhsm -new -key <web_server_fake_PEM.key> -
   out <web_server.csr>
   ```
3. 自己署名の証明書を作成する

   次のコマンドを実行して、 AWS CloudHSM OpenSSL Dynamic Engine を使用して、HSM のプライベートキーで CSR に署名します。これにより、自己署名証明書が作成されます。コマンドの以下の値を独自の値に置き換えます。

   - *<web_server.csr>* - CSR を含むファイルの名前です。
   - *<web_server_fake_PEM.key>* - フェイク PEM プライベートキーが含まれるファイルの名前です。
   - *<web_server.crt>* - ウェブサーバー証明書が含まれるファイルの名前です。

   ```
   $ openssl x509 -engine cloudhsm -req -days 365 -in <web_server.csr> -
   signkey <web_server_fake_PEM.key> -out <web_server.crt>
   ```

以上のステップが完了したら、「ステップ 3: ウェブサーバーを設定する」に進みます。

既存のプライベートキーと証明書をインポートします

ウェブサーバーの HTTPS 用のプライベートキーおよび対応する SSL/TLS 証明書がすでにある場合があります。その場合、このセクションの手順に従って、そのキーを HSM にインポートすることができます。

> **Note**
>
> プライベートキーのインポートとクライアント SDKの互換性についてのいくつかの注意事項:
>
> - 既存のプライベートキーをインポートするには、クライアント SDK 3 が必要です。
> - クライアント SDK 5 では、クライアント SDK 3 のプライベートキーを使用できます。
> - クライアント SDK 3 の OpenSSL 動的エンジンは、最新の Linux プラットフォームをサポートしていませんが、クライアント SDK 5 の OpenSSL 動的エンジンの実装はサポートしています。Client SDK 3 で提供されるキー管理ユーティリティ (KMU) を使用して既存のプライベートキーをインポートしてから、そのプライベートキーとクライアント SDK 5 での OpenSSL Dynamic Engine の実装を使用して、最新の Linux プラットフォームで SSL/TLS オフロードをサポートできます。

Client SDK 3 で既存のプライベートキーを HSM にインポートするには

1. Amazon EC2 クライアントインスタンスに接続します。必要に応じて、既存のプライベートキーと証明書をインスタンスにコピーします。
2. Client SDK 3 の インストールおよび設定
3. 次のコマンドを実行してクライアントを起動します AWS CloudHSM 。

   Amazon Linux

   ```
   $ sudo start cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 8

   ```
   $ sudo service cloudhsm-client start
   ```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

4. 次のコマンドを使用して、key_mgmt_util コマンドラインツールを起動します。

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

5. 次のコマンドを実行して HSM にログインします。*<user name>* と *<password>* を、暗号化ユーザー (CU) のユーザー名とパスワードに置き換えます。

```
Command: loginHSM -u CU -s <user name> -p <password>
```

6. 次のコマンドを実行して、プライベートキーを HSM 内にインポートします。

   a. 次のコマンドを実行して、現在のセッションでのみ有効な対称ラップキーを作成します。コマンドと出力が表示されます。

```
Command: genSymKey -t 31 -s 16 -sess -l wrapping_key_for_import
```

```
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Created.  Key Handle: 6
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

b. 次のコマンドを実行して、既存のプライベートキーを HSM 内にインポートします。コマンドと出力が表示されます。以下の値は独自の値に置き換えてください。

- *<web_server_existing.key>* - プライベートキーが含まれるファイルの名前です。
- *<web_server_imported_key>* - インポートしたプライベートキーのラベルです。
- *<wrapping_key_handle>* - 前述のコマンドで生成したラップキーのハンドルです。前の例で生成したラップキーのハンドルは 6 です。

```
Command: importPrivateKey -f <web_server_existing.key> -
l <web_server_imported_key> -w <wrapping_key_handle>

BER encoded key length is 1219
Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS
Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS
Private Key Unwrapped.  Key Handle: 8
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

7. 次のコマンドを実行して、プライベートキーをフェイク PEM 形式でエクスポートしてファイルに保存します。以下の値は独自の値に置き換えてください。

- *<private_key_handle>* - インポートしたプライベートキーのハンドルです。このハンドルは、前のステップで 2 番目のコマンドで生成したものです。前の例で生成したプライベートキーのハンドルは 8 です。
- *<web_server_fake_PEM.key>* - エクスポートしたフェイク PEM プライベートキーが含まれるファイルの名前です。

```
Command: getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
```

8. 次のコマンドを実行して key_mgmt_util を停止します。

```
Command: exit
```

以上のステップが完了したら、「ステップ 3: ウェブサーバーを設定する」に進みます。

ステップ 3: ウェブサーバーを設定する

前のステップで作成した HTTPS 証明書とフェイク PEM プライベートキーを使用するようにウェブサーバーソフトウェアの設定を更新します。開始する前に、既存の証明書とキーを必ずバックアップしてください。これで、 AWS CloudHSMを使用して、Linux ウェブサーバーソフトウェアに SSL/TLS オフロードを設定できます。

次のいずれかのセクションの手順を完了します。

トピック

- NGINX ウェブサーバーを設定する
- Apache ウェブサーバーの設定をします。

NGINX ウェブサーバーを設定する

このセクションは、サポートされているプラットフォームで NGINX を設定するために使用します。

NGINX のウェブサーバー設定を更新するには

1. クライアントインスタンスに接続します。
2. 次のコマンドを実行して、ウェブサーバー証明書とフェイク PEM プライベートキーに必要なディレクトリを作成します。

```
$ sudo mkdir -p /etc/pki/nginx/private
```

3. 次のコマンドを実行して、ウェブサーバーの証明書を所定場所にコピーします。*<web_server.crt>* を、ウェブサーバー証明書の名前に置き換えます。

```
$ sudo cp <web_server.crt> /etc/pki/nginx/server.crt
```

4. 次のコマンドを実行して、フェイク PEM プライベートキーを所定場所にコピーします。*<web_server_fake_PEM.key>* をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/nginx/private/server.key
```

5. 次のコマンドを実行してファイルの所有権を変更し、nginx という名前のユーザーがそれらのファイルを読み取れるようにします。

```
$ sudo chown nginx /etc/pki/nginx/server.crt /etc/pki/nginx/private/server.key
```

6. 次のコマンドを実行して、/etc/nginx/nginx.conf ファイルをバックアップします。

```
$ sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

7. NGINX の設定を更新します。

> **Note**
>
> 各クラスターは、すべての NGINX ウェブサーバーで最大 1000 の NGINX ワーカープロセスをサポートできます。

Amazon Linux

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

- Client SDK 3 を使用している場合

```
ssl_engine cloudhsm;
env n3fips_password;
```

- Client SDK 5 を使用している場合

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
```

```
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Amazon Linux 2

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

- Client SDK 3 を使用している場合

```
ssl_engine cloudhsm;
env n3fips_password;
```

- Client SDK 5 を使用している場合

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
```

```
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

CentOS 7

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

- Client SDK 3 を使用している場合

```
ssl_engine cloudhsm;
env n3fips_password;
```

- Client SDK 5 を使用している場合

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
```

```
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

CentOS 8

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
```

```
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Red Hat 7

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

- Client SDK 3 を使用している場合

```
ssl_engine cloudhsm;
env n3fips_password;
```

- Client SDK 5 を使用している場合

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
```

```
}
```

Red Hat 8

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }
```

```
    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Ubuntu 16.04 LTS

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

```
ssl_engine cloudhsm;
    env n3fips_password;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
```

```
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 18.04 LTS

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

```
ssl_engine cloudhsm;
    env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
```

```
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 20.04 LTS

テキストエディタを使用して、/etc/nginx/nginx.conf ファイルを編集します。これには Linux の root 権限が必要です。ファイルの先頭に、次の行を追加します。

```
ssl_engine cloudhsm;
    env CLOUDHSM_PIN;
```

次に、ファイルの TLS セクションに次の内容を追加します。

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;
```

```
        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

ファイルを保存します。

8. systemd 設定ファイルをバックアップしてから、EnvironmentFile パスを設定します。

Amazon Linux

対処は必要ありません。

Amazon Linux 2

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

CentOS 7

対処は必要ありません。

CentOS 8

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

Red Hat 7

対処は必要ありません。

Red Hat 8

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 16.04

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 18.04

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 20.04 LTS

1. nginx.service ファイルをバックアップします。

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. /lib/systemd/system/nginx.service ファイルをテキストエディタで開き、[Service] セクションに次のパスを追加します。

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

9. /etc/sysconfig/nginx ファイルの存在を確認してから、次のいずれかを実行します。

- ファイルが存在する場合は、次のコマンドを実行してファイルをバックアップします。

```
$ sudo cp /etc/sysconfig/nginx /etc/sysconfig/nginx.backup
```

- ファイルが存在しない場合は、テキストエディタを開き、/etc/sysconfig/ フォルダ内に nginx という名前のファイルを作成します。

10. NGINX 環境を設定します。

> **Note**
>
> クライアント SDK 5 では CU の認証情報を保存するための CLOUDHSM_PIN 環境変数が導入されています。

Amazon Linux

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Amazon Linux 2

テキストエディタで `/etc/sysconfig/nginx` ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

CentOS 7

テキストエディタで `/etc/sysconfig/nginx` ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

CentOS 8

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Red Hat 7

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Red Hat 8

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Ubuntu 16.04 LTS

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
n3fips_password=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Ubuntu 18.04 LTS

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Ubuntu 20.04 LTS

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

ファイルを保存します。

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

11. NGINX ウェブサーバーを起動します。

Amazon Linux

テキストエディタで /etc/sysconfig/nginx ファイルを開きます。これには Linux の root 権限が必要です。Cryptography User (CU) 認証情報を追加:

```
$ sudo service nginx start
```

Amazon Linux 2

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

CentOS 7

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

CentOS 8

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Red Hat 7

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Red Hat 8

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Ubuntu 16.04 LTS

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Ubuntu 18.04 LTS

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Ubuntu 20.04 LTS

実行中の NGINX プロセスをすべて停止する

```
$ sudo systemctl stop nginx
```

systemd 設定をリロードして最新の変更を取得する

```
$ sudo systemctl daemon-reload
```

NGINX プロセスを開始する

```
$ sudo systemctl start nginx
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

12. （オプション）スタートアップ時に NGINX を起動するようにプラットフォームを設定します。

Amazon Linux

```
$ sudo chkconfig nginx on
```

Amazon Linux 2

```
$ sudo systemctl enable nginx
```

CentOS 7

対処は必要ありません。

CentOS 8

```
$ sudo systemctl enable nginx
```

Red Hat 7

対処は必要ありません。

Red Hat 8

```
$ sudo systemctl enable nginx
```

Ubuntu 16.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 18.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 20.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

ウェブサーバー設定を更新したら、「ステップ 4: HTTPS トラフィックを有効にして証明書を検証する」に移動します。

Apache ウェブサーバーの設定をします。

このセクションでは、サポートされているプラットフォームで Apache を設定します。

Apache のウェブサーバー設定を更新するには

1. Amazon EC2 クライアントインスタンスに接続します。
2. プラットフォーム用の証明書とプライベートキーのデフォルトの場所を定義します。

Amazon Linux

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Amazon Linux 2

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

CentOS 7

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

CentOS 8

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Red Hat 7

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Red Hat 8

/etc/httpd/conf.d/ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Ubuntu 16.04 LTS

/etc/apache2/sites-available/default-ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 18.04 LTS

/etc/apache2/sites-available/default-ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 20.04 LTS

/etc/apache2/sites-available/default-ssl.conf ファイルに、次の値が存在することを確認します。

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

3. ウェブサーバーの証明書を、プラットフォームで必要な場所にコピーします。

Amazon Linux

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Amazon Linux 2

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

CentOS 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

CentOS 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Red Hat 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Red Hat 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Ubuntu 16.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Ubuntu 18.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Ubuntu 20.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

<web_server.crt> を、ウェブサーバー証明書の名前に置き換えます。

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

4. 偽の PEM プライベートキーをプラットフォームの所定場所にコピーします。

Amazon Linux

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Amazon Linux 2

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

CentOS 7

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

CentOS 8

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Red Hat 7

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Red Hat 8

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Ubuntu 16.04 LTS

```
$ sudo cp <web_server_fake_PEM.key> /etc/ssl/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Ubuntu 18.04 LTS

```
$ sudo cp <web_server_fake_PEM.key> /etc/ssl/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Ubuntu 20.04 LTS

```
$ sudo cp <web_server_fake_PEM.key> /etc/ssl/private/localhost.key
```

<web_server_fake_PEM.key> をフェイク PEM プライベートキーが含まれるファイルの名前に置き換えます。

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

5. プラットフォームで必要な場合は、これらのファイルの所有権を変更します。

Amazon Linux

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

Amazon Linux 2

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

CentOS 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

CentOS 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

Red Hat 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

Red Hat 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

apache という名前のユーザーに読み取り権限を与えます。

Ubuntu 16.04 LTS

対処は必要ありません。

Ubuntu 18.04 LTS

対処は必要ありません。

Ubuntu 20.04 LTS

対処は必要ありません。

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

6. プラットフォームに合わせて、Apache のディレクティブを設定します。

Amazon Linux

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

ファイルを保存します。

Amazon Linux 2

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
```

```
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

ファイルを保存します。

CentOS 7

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

ファイルを保存します。

CentOS 8

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

ファイルを保存します。

Red Hat 7

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

ファイルを保存します。

Red Hat 8

このプラットフォームの SSL ファイルを探します。

```
/etc/httpd/conf.d/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

ファイルを保存します。

Ubuntu 16.04 LTS

このプラットフォームの SSL ファイルを探します。

```
/etc/apache2/mods-available/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

ファイルを保存します。

SSL モジュールとデフォルトの SSL サイト設定を有効にします。

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 18.04 LTS

このプラットフォームの SSL ファイルを探します。

```
/etc/apache2/mods-available/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProtocol TLSv1.2 TLSv1.3
```

ファイルを保存します。

SSL モジュールとデフォルトの SSL サイト設定を有効にします。

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 20.04 LTS

このプラットフォームの SSL ファイルを探します。

```
/etc/apache2/mods-available/ssl.conf
```

このファイルには、サーバーの実行方法を定義する Apache ディレクティブが含まれています。ディレクティブは左側に表示され、その後に値が続きます。テキストエディタを使用して、このファイルを編集します。これには Linux の root 権限が必要です。

これらの値を使用して、次のディレクティブを更新または入力します。

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProtocol TLSv1.2 TLSv1.3
```

ファイルを保存します。

SSL モジュールとデフォルトの SSL サイト設定を有効にします。

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

7. プラットフォーム用の環境値ファイルを設定します。

Amazon Linux

対処は必要ありません。/etc/sysconfig/httpd に環境値が入ります

Amazon Linux 2

httpd サービスファイルを開きます。

```
/lib/systemd/system/httpd.service
```

[Service] セクションの下に、以下を追加します。

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 7

httpd サービスファイルを開きます。

```
/lib/systemd/system/httpd.service
```

[Service] セクションの下に、以下を追加します。

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 8

httpd サービスファイルを開きます。

```
/lib/systemd/system/httpd.service
```

[Service] セクションの下に、以下を追加します。

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 7

httpd サービスファイルを開きます。

```
/lib/systemd/system/httpd.service
```

[Service] セクションの下に、以下を追加します。

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 8

httpd サービスファイルを開きます。

```
/lib/systemd/system/httpd.service
```

[Service] セクションの下に、以下を追加します。

```
EnvironmentFile=/etc/sysconfig/httpd
```

Ubuntu 16.04 LTS

対処は必要ありません。/etc/sysconfig/httpd に環境値が入ります

Ubuntu 18.04 LTS

対処は必要ありません。/etc/sysconfig/httpd に環境値が入ります

Ubuntu 20.04 LTS

対処は必要ありません。/etc/sysconfig/httpd に環境値が入ります

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

8. プラットフォーム用の環境変数を格納するファイルで、暗号化ユーザー (CU) の認証情報を含む環境変数を設定します。

Amazon Linux

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

Amazon Linux 2

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

CentOS 7

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

CentOS 8

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

Red Hat 7

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

- Client SDK 3 を使用している場合

```
n3fips_password=<CU user name>:<password>
```

- Client SDK 5 を使用している場合

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

Red Hat 8

テキストエディタを使用して、/etc/sysconfig/httpd を編集します。

```
CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

> **Note**
>
> クライアント SDK 5 では CU の認証情報を保存するための CLOUDHSM_PIN 環境変数が導入されています。

Ubuntu 16.04 LTS

テキストエディタを使用して、/etc/apache2/envvars を編集します。

```
export n3fips_password=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

Ubuntu 18.04 LTS

テキストエディタを使用して、/etc/apache2/envvars を編集します。

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

> **Note**
>
> クライアント SDK 5 では CU の認証情報を保存するための CLOUDHSM_PIN 環境変数が導入されています。クライアント SDK 3 では、CU の認証情報を n3fips_password 環境変数に保存していました。クライアント SDK 5 は両方の環境変数をサポートしますが、CLOUDHSM_PIN を使用することを推奨します。

Ubuntu 20.04 LTS

テキストエディタを使用して、/etc/apache2/envvars を編集します。

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

<CU #####> と *<#####>* を CU の認証情報に置き換えます。

> **Note**
>
> クライアント SDK 5 では CU の認証情報を保存するための CLOUDHSM_PIN 環境変数が導入されています。クライアント SDK 3 では、CU の認証情報を n3fips_password 環境変数に保存していました。クライアント SDK 5 は両方の環境変数をサポートしますが、CLOUDHSM_PIN を使用することを推奨します。

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

9. Apache ウェブサーバーを起動します。

Amazon Linux

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Amazon Linux 2

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Ubuntu 16.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 18.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 20.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

10. (オプション) スタートアップに Apache を起動するようにプラットフォームを設定します。

Amazon Linux

```
$ sudo chkconfig httpd on
```

Amazon Linux 2

```
$ sudo chkconfig httpd on
```

CentOS 7

```
$ sudo chkconfig httpd on
```

CentOS 8

```
$ systemctl enable httpd
```

Red Hat 7

```
$ sudo chkconfig httpd on
```

Red Hat 8

```
$ systemctl enable httpd
```

Ubuntu 16.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 18.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 20.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 22.04 LTS

OpenSSL Dynamic Engine はまだサポートされていません。

ウェブサーバー設定を更新したら、「ステップ 4: HTTPS トラフィックを有効にして証明書を検証する」に移動します。

ステップ 4: HTTPS トラフィックを有効にして証明書を検証する

で SSL/TLS オフロード用にウェブサーバーを設定したら AWS CloudHSM、インバウンド HTTPS トラフィックを許可するセキュリティグループにウェブサーバーインスタンスを追加します。これにより、ウェブブラウザなどのクライアントがウェブサーバーと HTTPS 接続を確立できるようになります。次に、ウェブサーバーに HTTPS 接続を行い、 で SSL/TLS オフロード用に設定した証明書を使用していることを確認します AWS CloudHSM。

トピック

- インバウンド HTTPS 接続の有効化
- 設定した証明書が HTTPS で使用されていることを検証する

インバウンド HTTPS 接続の有効化

クライアント (ウェブブラウザなど) からウェブサーバーに接続するには、インバウンド HTTPS 接続を許可するセキュリティグループを作成します。具体的には、ポート 443 でインバウンドの TCP 接続を許可する必要があります。このセキュリティグループをウェブサーバーに割り当てます。

HTTPS のセキュリティグループを作成してウェブサーバーに割り当てるには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[セキュリティグループ] を選択します。
3. [Create Security Group] を選択します。
4. [Create Security Group] で、以下の操作を行います。
    a. [Security group name] に、作成するセキュリティグループの名前を入力します
    b. (オプション) 作成するセキュリティグループの説明を入力します。
    c. [VPC] で、ウェブサーバーのAmazon EC2インスタンスが含まれている VPC を選択します。
    d. [Add rule (ルールの追加)] を選択します。
    e. [タイプ] で、ドロップダウンウィンドウから [HTTPS] を選択します。
    f. [ソース] には、ソースの場所を入力します。
    g. [Create Security Group] を選択します。

5. ナビゲーションペインで、[インスタンス] を選択します。
6. ウェブサーバーインスタンスの横にあるチェックボックスを選択します。
7. ページの上部で [アクション] ドロップダウンメニューを選択します。[セキュリティ] を選択し、[セキュリティグループの変更] を選択します。
8. [関連付けられたセキュリティグループ] で、検索ボックスを選択して HTTPS 用に作成したセキュリティグループを選択します。次に、[セキュリティグループの追加] を選択します。
9. [Save] を選択します。

設定した証明書が HTTPS で使用されていることを検証する

ウェブサーバーをセキュリティ グループに追加した後、SSL/TLS オフロードが自己署名証明書を使用していることを確認できます。この検証には、ウェブブラウザ、または OpenSSL s_client などのツールを使用できます。

ウェブブラウザで SSL/TLS オフロードを確認するには

1. ウェブブラウザを使用し、サーバーの公開 DNS 名または IP アドレスを使用してウェブサーバーに接続します。アドレスバーの URL が https:// で始まっていることを確認します。例えば **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`** です。

   > **Tip**
   >
   > Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. ウェブブラウザを使用して、ウェブサーバー証明書を表示します。詳細については、次を参照してください。

   - Mozilla Firefox の場合は、Mozilla サポートウェブサイトの「証明書を見る」を参照してください。
   - Google Chrome の場合は、ウェブ開発者向け Google ツールのウェブサイトで「セキュリティの問題を理解する」を参照してください。

   他のウェブブラウザでも、同様の機能を使用してウェブサーバー証明書を表示できる場合があります。

3. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

OpenSSL s_client で SSL/TLS オフロードを確認するには

1. HTTPS を使用してウェブサーバーに接続するには、次の OpenSSL コマンドを実行します。<#####> は、ウェブサーバーの公開 DNS 名または IP アドレスに置き換えます。

```
openssl s_client -connect <server name>:443
```

> **Tip**
>
> Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

これで、ウェブサイトが HTTPS で保護されるようになりました。ウェブサーバーのプライベートキーは、 AWS CloudHSM クラスターの HSM に保存されます。

ロードバランサーを追加するには、「Elastic Load Balancing を使用してロードバランサーを追加する (オプション)」を参照してください。

Linux での SSL/TLS オフロードに JSSE で Tomcat を使う

このトピックでは、 AWS CloudHSM JCE SDK で Java Secure Socket Extension (JSSE) を使用して SSL/TLS オフロードを設定する step-by-step 手順について説明します。

トピック

- 概要
- ステップ 1: 前提条件の設定
- ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする
- ステップ 3: Tomcat ウェブサーバーを設定する
- ステップ 4: HTTPS トラフィックを有効にして証明書を検証する

概要

では AWS CloudHSM、Tomcat ウェブサーバーは Linux で HTTPS をサポートしています。 AWS CloudHSM JCE SDK は、JSSE (Java Secure Socket Extension) で使用できるインターフェイスを提供し、そのようなウェブサーバーに HSMsを使用できるようにします。 AWS CloudHSM JCE は、JSSE を AWS CloudHSM クラスターに接続するブリッジです。JSSE は、Secure Socket Layer (SSL) と Transport Layer Security (TLS) プロトコル用の Java API です。

ステップ 1: 前提条件の設定

Linux で SSL/TLS オフロード AWS CloudHSM に で Tomcat ウェブサーバーを使用するには、次の前提条件に従います。Client SDK 5 と Tomcat ウェブサーバーでウェブサーバー SSL/TLS オフロードを設定するには、これらの前提条件を満たす必要があります。

> **Note**
>
> プラットフォームごとに、異なる前提条件が必要です。使用しているプラットフォームに適したインストール手順を必ず実行してください。

前提条件

- Tomcat ウェブサーバーがインストールされた Linux オペレーティングシステムを実行する Amazon EC2 インスタンス。
- HSM でこのウェブサーバーのプライベートキーを所有および管理する Crypto User (CU)。
- JCE for Client SDK 5 がインストールされ、設定された 2 HSMs) を持つアクティブな AWS CloudHSM クラスター。 ???

> **Note**
>
> HSM クラスターは1つでも使用できますが、まずクライアントキーの耐久性を無効にする必要があります。詳細については、クライアントキーの耐久性設定の管理 そして クライアント SDK 5 設定ツール を参照してください。

前提条件を満たすには

1. 少なくとも 2 つのハードウェアセキュリティモジュール (HSM) を持つアクティブな AWS CloudHSM クラスター AWS CloudHSM に の JCE をインストールして設定します。 HSMs インストールの詳細については、「Client SDK 5 向け JCE」を参照してください。
2. AWS CloudHSM クラスターにアクセスできる EC2 Linux インスタンスで、Apache Tomcat の指示に従って Tomcat ウェブサーバーをダウンロードしてインストールします。
3. CloudHSM CLI を使用して Crypto User (CU) を作成します。HSM ユーザーの管理の詳細については、CloudHSM CLI を使用した HSM ユーザー管理について を参照してください。

   > **Tip**
   >
   > CU のユーザー名とパスワードを書き留めます。後に、ウェブサーバーの HTTPS プライベートキーや証明書を生成またはインポートするときに必要になります。

4. Java キーツールを使用して JCE をセットアップするには、Client SDK 5 を使用して Java Keytool および Jarsigner と統合する に記載されている手順に従ってください。

以上のステップが完了したら、「ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする」に進みます。

メモ

- セキュリティ強化 Linux (SELinux) および Web サーバーを使用するには、クライアント SDK 5 が HSM と通信するために使用するポート 2223 でアウトバウンド TCP 接続を許可する必要があります。
- クラスターを作成してアクティブ化し、EC2 インスタンスにクラスターへのアクセス権を付与するには、AWS CloudHSMの使用開始 の手順を実行します。このセクションでは、1 つの HSM と Amazon EC2 クライアントインスタンスを使用してアクティブなクラスターを作成する step-by-step 手順について説明します。このクライアントインスタンスをウェブサーバーとして使用することができます。
- クライアントキーの耐久性を無効化しないようにするには、クラスターに複数の HSM を追加します。詳細については、「HSM の追加」を参照してください。
- クライアントインスタンスに接続するには、SSH または PuTTY を使用することができます。詳細については、「Amazon EC2 ドキュメント」の「SSH を使用した Linux インスタンスへの接続」または「PuTTY を使用した Windows から Linux インスタンスへの接続」を参照してください。

ステップ 2: プライベートキーと SSL/TLS 証明書を生成またはインポートする

HTTPS を有効にするには、Tomcat ウェブサーバーアプリケーションにプライベートキーと、それに対応する SSL/TLS 証明書が必要です。でウェブサーバーの SSL/TLS オフロードを使用するには AWS CloudHSM、プライベートキーを AWS CloudHSM クラスターの HSM に保存する必要があります。

> **Note**
>
> プライベートキーとそれに対応する証明書を持っていない場合、HSM でプライベートキーを生成できます。このプライベートキーを使用して証明書署名リクエスト (CSR) を作成し、それを使用してSSL/TLS証明書を作成します。

HSM のプライベートキーへの参照と関連する証明書を含むローカル AWS CloudHSM KeyStore ファイルを作成します。ウェブサーバーは AWS CloudHSM KeyStore ファイルを使用して、SSL/TLS オフロード中に HSM のプライベートキーを識別します。

トピック

- プライベートキーの生成
- 自己署名証明書を生成します

プライベートキーの生成

このセクションでは、JDK KeyTool から を使用してキーペアを生成する方法を示します。HSM 内でキーペアを生成したら、ファイルとしてエクスポートし KeyStore 、対応する証明書を生成できます。

ユースケースに応じて、RSA または EC キーペアを生成できます。以下の手順では、RSA キーペアを生成する方法を示します。

で **genkeypair** コマンド KeyTool を使用して RSA キーペアを生成する

1. 下の *<VARIABLES>* を特定のデータに置き換えたら、次のコマンドを使用して `jsse_keystore.keystore` という名前のキーストアファイルを生成します。このファイルには HSM 上のプライベートキーへの参照が含まれます。

```
$ keytool -genkeypair -alias <UNIQUE ALIAS FOR KEYS> -keyalg <KEY ALGORITHM> -
keysize <KEY SIZE> -sigalg <SIGN ALGORITHM> \
```

```
        -keystore <PATH>/<JSSE KEYSTORE NAME>.keystore -storetype CLOUDHSM \
        -dname CERT_DOMAIN_NAME \
        -J-classpath '-J'$JAVA_LIB'/*:/opt/cloudhsm/java/*:./*' \
        -provider "com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider" \
        -providerpath "$CLOUDHSM_JCE_LOCATION" \
        -keypass <KEY PASSWORD> -storepass <KEYSTORE PASSWORD>
```

- ***<PATH>***: キーストアファイルを生成するパス。
- ***<UNIQUE ALIAS FOR KEYS>***: これは HSM 上でキーを一意に識別するために使用されます。このエイリアスは、キーの LABEL 属性として設定されます。
- ***<KEY PASSWORD>***: キーへの参照はローカルキーストアファイルに保存され、このパスワードによってローカル参照が保護されます。
- ***<KEYSTORE PASSWORD>***: これはローカルキーストアファイルのパスワードです。
- ***<JSSE KEYSTORE NAME>***: キーストアファイルの名前。
- ***<CERT DOMAIN NAME>***: X.500 識別名。
- ***<KEY ALGORITHM>***: キーペアを生成するためのキーアルゴリズム (RSA と EC など)。
- ***<KEY SIZE>***: キーペアを生成するためのキーサイズ (たとえば、2048、3072、4096)。
- ***<SIGN ALGORITHM>***: キーペアを生成するためのキーサイズ (たとえば、SHA1withRSA、SHA224withRSA、SHA256withRSA、SHA384withRSA、SHA512withRSA) 。

2. コマンドが成功したことを確認するには、次のコマンドを入力し、RSA キーペアが正常に生成されたことを確認します。

```
$ ls <PATH>/<JSSE KEYSTORE NAME>.keystore
```

自己署名証明書を生成します

キーストアファイルとともにプライベートキーを生成したら、このファイルを使用して証明書署名リクエスト (CSR) と証明書を生成できます。

本稼働環境では、通常、認証機関 (CA) を使用して CSR から証明書を作成します。CA は、テスト環境では必要ありません。CA を使用する場合は、CA に CSR ファイルを送信し、HTTPS 用のウェブサーバーで提供される署名付き SSL/TLS 証明書を使用してください。

CA を使用する代わりに、 を使用して自己署名証明書 KeyTool を作成できます。自己署名証明書はブラウザによって信頼されないため、本稼働環境では使用しないでください。これらは、テスト環境で使用することができます。

> **Warning**
>
> 自己署名証明書はテスト環境でのみ使用する必要があります。本稼働環境では、証明機関を使用して証明書を作成するなど、より安全な方法を使用してください。

証明書を生成する

1. 前のステップで生成したキーストアファイルのコピーを入手します。
2. 次のコマンドを実行して、 を使用して証明書署名リクエスト (CSR) KeyTool を作成します。

```
$ keytool -certreq -keyalg RSA -alias unique_alias_for_key -file certreq.csr \
        -keystore <JSSE KEYSTORE NAME>.keystore -storetype CLOUDHSM \
        -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
        -keypass <KEY PASSWORD> -storepass <KEYSTORE PASSWORD>
```

> **Note**
>
> 証明書署名リクエストの出力ファイルは `certreq.csr` です。

証明書に署名する

- 下の *<VARIABLES>* を特定のデータに置き換えた後、次のコマンドを実行して、HSM 上のプライベートキーを使用して CSR に署名します。これにより、自己署名証明書が作成されます。

```
$ keytool -gencert -infile certreq.csr -outfile certificate.crt \
    -alias <UNIQUE ALIAS FOR KEYS> -keypass <KEY_PASSWORD> -
storepass <KEYSTORE_PASSWORD> -sigalg SIG_ALG \
    -storetype CLOUDHSM -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
    -keystore jsse_keystore.keystore
```

> **Note**
>
> `certificate.crt` は、エイリアスのプライベートキーを使用する署名付き証明書です。

Keystore に証明書をインポートする

- 下の *<VARIABLES>* を特定のデータに置き換えた後、次のコマンドを実行して、署名付き証明書を信頼できる証明書としてインポートします。このステップでは、エイリアスで識別されるキーストアエントリに証明書を保存します。

```
$ keytool -import -alias <UNIQUE ALIAS FOR KEYS> -keystore jsse_keystore.keystore \
    -file certificate.crt -storetype CLOUDHSM \
    -v -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
    -keypass <KEY PASSWORD> -storepass <KEYSTORE_PASSWORD>
```

証明書を PEM に変換する

- 次のコマンドを実行して、署名付き証明書ファイル (.crt) を PEM に変換します。PEM ファイルは http クライアントからのリクエストの送信に使用されます。

```
$ openssl x509 -inform der -in certificate.crt -out certificate.pem
```

これらの手順を完了したら、「ステップ 3: ウェブサーバーの設定」に進みます。

ステップ 3: Tomcat ウェブサーバーを設定する

前のステップで作成した HTTPS 証明書と PEM ファイルを使用するようにウェブサーバーソフトウェアの設定を更新します。開始する前に、既存の証明書とキーを必ずバックアップしてください。これで、 AWS CloudHSMを使用して、Linux ウェブサーバーソフトウェアに SSL/TLS オフロードを設定できます。詳細については、「Apache Tomcat 9 設定リファレンス」を参照してください。

サーバーを停止します

- 下の *<VARIABLES>* を特定のデータに置き換えたら、設定を更新する前に、次のコマンドを実行して Tomcat Server を停止します

```
$ /<TOMCAT DIRECTORY>/bin/shutdown.sh
```

  - ***<TOMCAT DIRECTORY>***: Tomcat のインストールディレクトリ。

Tomcat のクラスパスを更新してください

1. クライアントインスタンスに接続します。
2. Tomcat インストールフォルダを探します。
3. 下の *<VARIABLES>* を特定のデータに置き換えたら、次のコマンドを使用して、Tomcat/bin/catalina.sh ファイルにある Tomcat クラスパスに Java ライブラリと Cloudhsm Java パスを追加します。

```
$ sed -i 's@CLASSPATH="$CLASSPATH""$CATALINA_HOME"\/bin\/
bootstrap.jar@CLASSPATH="$CLASSPATH""$CATALINA_HOME"\/bin\/bootstrap.jar:'"
        <JAVA LIBRARY>"'\/*:\/opt\/cloudhsm\/java\/*:.\/*@' <TOMCAT PATH> /bin/
catalina.sh
```

- ***<JAVA LIBRARY>***: Java JRE ライブラリの場所。
- ***<TOMCAT PATH>***: Tomcat のインストールフォルダー。

HTTPS コネクタをサーバー設定に追加します。

1. Tomcat のインストールフォルダーに移動します。
2. 下の *<VARIABLES>* を特定のデータに置き換えたら、以下のコマンドを使用して HTTPS コネクタを追加し、前提条件で生成された証明書を使用します。

```
$ sed -i '/<Connector port="8080"/i <Connector port=\"443\" maxThreads=\"200\"
 scheme=\"https\" secure=\"true\" SSLEnabled=\"true\" keystoreType=\"CLOUDHSM\"
 keystoreFile=\"
        <CUSTOM DIRECTORY>/<JSSE KEYSTORE NAME>.keystore\" keystorePass=\"<KEYSTORE
 PASSWORD>\" keyPass=\"<KEY PASSWORD>
        \" keyAlias=\"<UNIQUE ALIAS FOR KEYS>" clientAuth=\"false\" sslProtocol=
\"TLS\"/>' <TOMCAT PATH>/conf/server.xml
```

- ***<CUSTOM DIRECTORY>***: キーストアファイルが置かれているディレクトリ。
- ***<JSSE KEYSTORE NAME>***: キーストアファイルの名前。
- ***<KEYSTORE PASSWORD>***: これはローカルキーストアファイルのパスワードです。
- ***<KEY PASSWORD>***: キーへの参照はローカルキーストアファイルに保存され、このパスワードによってローカル参照が保護されます。
- ***<UNIQUE ALIAS FOR KEYS>***: これは HSM 上でキーを一意に識別するために使用されます。このエイリアスは、キーの LABEL 属性として設定されます。

- ***<TOMCAT PATH>***: Tomcat フォルダへのパス。

サーバーの起動

- 下の *<VARIABLES>* を特定のデータに置き換えたら、以下のコマンドを使用して Tomcat サーバーを起動します。

```
$ /<TOMCAT DIRECTORY>/bin/startup.sh
```

> **Note**
>
> ***<TOMCAT DIRECTORY>*** は、Tomcat のインストールディレクトリの名前です。

ウェブサーバー設定を更新したら、「ステップ 4: HTTPS トラフィックを有効にして証明書を検証する」に移動します。

ステップ 4: HTTPS トラフィックを有効にして証明書を検証する

で SSL/TLS オフロード用にウェブサーバーを設定したら AWS CloudHSM、インバウンド HTTPS トラフィックを許可するセキュリティグループにウェブサーバーインスタンスを追加します。これにより、ウェブブラウザなどのクライアントがウェブサーバーと HTTPS 接続を確立できるようになります。次に、ウェブサーバーに HTTPS 接続を行い、 で SSL/TLS オフロード用に設定した証明書を使用していることを確認します AWS CloudHSM。

トピック

- インバウンド HTTPS 接続の有効化
- 設定した証明書が HTTPS で使用されていることを検証する

インバウンド HTTPS 接続の有効化

クライアント (ウェブブラウザなど) からウェブサーバーに接続するには、インバウンド HTTPS 接続を許可するセキュリティグループを作成します。具体的には、ポート 443 でインバウンドの TCP 接続を許可する必要があります。このセキュリティグループをウェブサーバーに割り当てます。

HTTPS のセキュリティグループを作成してウェブサーバーに割り当てるには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。

2. ナビゲーションペインで、[セキュリティグループ] を選択します。
3. [Create Security Group] を選択します。
4. [Create Security Group] で、以下の操作を行います。
   a. [Security group name] に、作成するセキュリティグループの名前を入力します
   b. (オプション) 作成するセキュリティグループの説明を入力します。
   c. [VPC] で、ウェブサーバーのAmazon EC2インスタンスが含まれている VPC を選択します。
   d. [Add rule (ルールの追加)] を選択します。
   e. [タイプ] で、ドロップダウンウィンドウから [HTTPS] を選択します。
   f. [ソース] には、ソースの場所を入力します。
   g. [Create Security Group] を選択します。
5. ナビゲーションペインで、[インスタンス] を選択します。
6. ウェブサーバーインスタンスの横にあるチェックボックスを選択します。
7. ページの上部で [アクション] ドロップダウンメニューを選択します。[セキュリティ] を選択し、[セキュリティグループの変更] を選択します。
8. [関連付けられたセキュリティグループ] で、検索ボックスを選択して HTTPS 用に作成したセキュリティグループを選択します。次に、[セキュリティグループの追加] を選択します。
9. [Save] を選択します。

設定した証明書が HTTPS で使用されていることを検証する

ウェブサーバーをセキュリティ グループに追加した後、SSL/TLS オフロードが自己署名証明書を使用していることを確認できます。この検証には、ウェブブラウザ、または OpenSSL s_client などのツールを使用できます。

ウェブブラウザで SSL/TLS オフロードを確認するには

1. ウェブブラウザを使用し、サーバーの公開 DNS 名または IP アドレスを使用してウェブサーバーに接続します。アドレスバーの URL が https:// で始まっていることを確認します。例えば **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`** です。

> **Tip**
>
> Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. ウェブブラウザを使用して、ウェブサーバー証明書を表示します。詳細については、次を参照してください。

   - Mozilla Firefox の場合は、Mozilla サポートウェブサイトの「証明書を見る」を参照してください。
   - Google Chrome の場合は、ウェブ開発者向け Google ツールのウェブサイトで「セキュリティの問題を理解する」を参照してください。

   他のウェブブラウザでも、同様の機能を使用してウェブサーバー証明書を表示できる場合があります。

3. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

OpenSSL s_client で SSL/TLS オフロードを確認するには

1. HTTPS を使用してウェブサーバーに接続するには、次の OpenSSL コマンドを実行します。*<#####>* は、ウェブサーバーの公開 DNS 名または IP アドレスに置き換えます。

```
openssl s_client -connect <server name>:443
```

> **Tip**
>
> Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

これで、ウェブサイトが HTTPS で保護されるようになりました。ウェブサーバーのプライベートキーは、 AWS CloudHSM クラスターの HSM に保存されます。

ロードバランサーを追加するには、「Elastic Load Balancing を使用してロードバランサーを追加する (オプション)」を参照してください。

Windows での SSL/TLS オフロードに IIS と CNG の使用

このチュートリアルでは、Windows ウェブサーバーで で AWS CloudHSM SSL/TLS オフロードを設定する step-by-step 手順について説明します。

トピック

概要

Windows では、Windows Server 用インターネットインフォメーションサービス (IIS) ウェブサーバーアプリケーションは HTTPS をネイティブにサポートしています。Microsoft の Cryptography API:Next Generation (CNG) のAWS CloudHSM キーストレージプロバイダー (KSP) には、暗号化オフロードとキーストレージ用に、IIS がクラスターの HSM を使用することを許可するインターフェイスが用意されています。 AWS CloudHSM KSP は IIS を AWS CloudHSM クラスターに接続するブリッジです。

このチュートリアルでは、以下のことを実行する方法を示します。

- Amazon EC2 インスタンスに、ウェブサーバーソフトウェアをインストールします。
- AWS CloudHSM クラスターに保存されている秘密キーで、HTTPS をサポートするようにウェブサーバソフトウェアを設定します。
- (オプション) Amazon EC2 を使用して 2 台目のウェブサーバーインスタンスを作成し、Elastic Load Balancing を使用してロードバランサーを作成します。ロードバランサーを使用すると、複数のサーバーに負荷を分散することでパフォーマンスを向上させることができます。また、1 つ以上のサーバーに障害が発生した場合、冗長性と高可用性を提供します。

始める準備ができたら、「ステップ 1: 前提条件の設定」を参照してください。

ステップ 1: 前提条件の設定

でウェブサーバーの SSL/TLS オフロードを設定するには AWS CloudHSM、以下が必要です。

- 少なくとも 1 つの HSM を持つアクティブな AWS CloudHSM クラスター。
- Windows OS が動作する Amazon EC2 インスタンスで、以下のソフトウェアがインストールされていることを確認します。
  - Windows 用の AWS CloudHSM クライアントソフトウェア。
  - Windows Server 用インターネットインフォメーションサービス (IIS)。
- HSM でこのウェブサーバーのプライベートキーを所有および管理する Crypto User (CU)。

> **Note**
>
> このチュートリアルでは Microsoft Windows Server 2016 を使用します。また、Microsoft Windows Server 2012 もサポートされていますが、Microsoft Windows Server 2012 R2 はサポート対象外です。

Windows Server インスタンスをセットアップし、HSM で CU を作成するには

1. 「開始」のステップを完了します。Amazon EC2 クライアントを起動する場合は、Windows Server 2016 または Windows Server 2012 AMI を選択します。これらのステップを完了すると、少なくとも 1 つの HSM を含むアクティブなクラスターが提供されます。Windows 用のクライアントソフトウェアがインストールされた Windows Server を実行している Amazon EC2 AWS CloudHSM クライアントインスタンスもあります。
2. (オプション) 他の HSM をクラスターに追加します。詳細については、「HSM の追加」を参照してください。
3. Windows Server に接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
4. CloudHSM CLI を使用して Crypto User (CU) を作成します。CU のユーザー名とパスワードを書き留めます。次のステップを完了するために必要になります。

> **Note**
>
> ユーザーの作成については、「CloudHSM CLI による HSM ユーザーの管理」を参照してください。

5. 前のステップで作成した CU ユーザー名とパスワードを使用して、HSM のログイン認証情報を設定します。
6. ステップ 5 で Windows 認証情報マネージャーを使用して HSM 認証情報を設定した場合は、psexec.exeからダウンロード SysInternals して、NT Authority\SYSTEM として次のコマンドを実行します。

```
psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe"
 --username <USERNAME> --password <PASSWORD>
```

<USERNAME> と *<PASSWORD>* を HSM 認証情報に置き換えてください。

IIS を Windows Server にインストールするには

1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. Windows Server で [サーバーマネージャー] を起動します。
3. [サーバー マネージャー] ダッシュボードで、[役割と機能の追加] を選択します。
4. [開始する前に] 情報を読み、[次へ] を選択します。
5. [インストールのタイプ] ページで、[ロールベースまたは機能ベースのインストール] を選択します。次いで、[次へ] を選択します。
6. [サーバーの選択] で、[Select a server from the server pool (サーバープールからサーバーを選択する)] を選択します。次いで、[次へ] を選択します。
7. [Server Roles (サーバーの役割位)] で、以下を実行します。
   a. [Web Server (IIS)] を選択します。
   b. [Add features that are required for Web Server (IIS) (Web Server (IIS) に必要な機能を追加する)] で、[機能の追加] を選択します。
   c. [次へ] を選択してサーバーロールの選択を完了します。
8. [Features (機能)] を選択し, 、デフォルト設定を使用します。次いで、[次へ] を選択します。

9. [Web Server Role (IIS)] の内容をお読みください。次いで、[次へ] を選択します。
10. [Select role services (ロールサービスの追加)] でデフォルトを受け入れるか、必要に応じて設定を変更します。次いで、[次へ] を選択します。
11. [確認] で確認情報を通読します。次に、[インストール] を選択します。
12. インストールが完了したら、[Close] をクリックします。

以上のステップが完了したら、「ステップ 2: 証明書署名リクエスト (CSR) および証明書を作成する」に進みます。

ステップ 2: 証明書署名リクエスト (CSR) および証明書を作成する

HTTPS を有効にするには、SSL/TLS 証明書とそれに対応するプライベートキーがウェブサーバーに必要です。で SSL/TLS オフロードを使用するには AWS CloudHSM、プライベートキーをクラスターの HSM に AWS CloudHSM 保存します。そのためには、Microsoft の Cryptography API:Next Generation (CNG) のAWS CloudHSM キーストレージプロバイダー (KSP) を使用して、証明書署名リクエスト (CSR) を作成します。作成したら、その CSR を 証明機関 (CA) に送信します。これで、証明書を生成する CSR に署名されます。

トピック

- CSR を作成する
- 署名証明書を取得してインポートする

CSR を作成する

Windows Server で AWS CloudHSM KSP を使用して CSR を作成します。

CSR を作成するには

1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. クライアント AWS CloudHSM デーモンを起動するには、次のコマンドを使用します。

   Amazon Linux

   ```
   $ sudo start cloudhsm-client
   ```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

3. Windows Server では、テキストエディタを使用して証明書リクエストファイルを作成し、名前を IISCertRequest.inf とします。IISCertRequest.inf ファイルの内容の例を以下に示します。ファイルで指定可能なセクション、キー、値の詳細については、「Microsoft のドキュメント」を参照してください。値 (ProviderName) は変更しないでください。

```
[Version]
Signature = "$Windows NT$"
[NewRequest]
Subject = "CN=example.com,C=US,ST=Washington,L=Seattle,O=ExampleOrg,OU=WebServer"
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
ProviderName = "Cavium Key Storage Provider"
KeyUsage = 0xf0
MachineKeySet = True
[EnhancedKeyUsageExtension]
OID=1.3.6.1.5.5.7.3.1
```

4. Windows の certreq コマンドを使用して、前のステップで作成した IISCertRequest.inf ファイルから CSR を作成します。以下の例では、CSR を IISCertRequest.csr という名前のファイルに保存します。証明書リクエストファイルに別のファイル名を使用した場合は、*IIS CertRequest.inf* を適切なファイル名に置き換えます。必要に応じて、*IIS CertRequest.csr* を CSR ファイルの別のファイル名に置き換えることができます。

```
C:\>certreq -new IISCertRequest.inf IISCertRequest.csr
        SDK Version: 2.03

CertReq: Request Created
```

   IISCertRequest.csr ファイルには、CSR が含まれます。署名証明書を取得するには、この CSR が必要です。

署名証明書を取得してインポートする

本稼働環境では、通常、認証機関 (CA) を使用して CSR から証明書を作成します。CA は、テスト環境では必要ありません。CA を使用する場合は、CSR ファイル (IISCertRequest.csr) を送信後、その CA を使用して署名済み SSL/TLS 証明書を作成します。

CA を使用する代わりに、OpenSSL のようなツールを使用して、自己署名証明書を作成することもできます。

> **Warning**
>
> 自己署名証明書はブラウザによって信頼されないため、本稼働環境では使用しないでください。これらは、テスト環境で使用することができます。

次の手順では、自己署名証明書を作成してウェブサーバーの CSR に署名する方法を示します。

自己署名証明書を作成するには

1. プライベートキーを作成するには、次の OpenSSL コマンドを使用します。オプションで *SelfSignedCA.key* をファイル名に置き換えて、プライベートキーを含めることができます。

```
openssl genrsa -aes256 -out SelfSignedCA.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.......................................+++
e is 65537 (0x10001)
Enter pass phrase for SelfSignedCA.key:
Verifying - Enter pass phrase for SelfSignedCA.key:
```

2. OpenSSL コマンドを使用して、前のステップで作成したプライベートキーで自己署名発行証明書を作成します。これは対話型コマンドです。画面の指示を読み、プロンプトに従います。*SelfSignedCA.key* をプライベートキーを含むファイルの名前 (異なる場合) に置き換えます。オプションで *SelfSignedCA.crt* をファイル名に置き換えて、自己署名証明書を含めることができます。

```
openssl req -new -x509 -days 365 -key SelfSignedCA.key -out SelfSignedCA.crt
Enter pass phrase for SelfSignedCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
```

```
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

自己署名証明書を使用してウェブサーバーの CSR に署名するには

- プライベートキーおよび自己署名証明書を使用して CSR に署名するには、次の OpenSSL コマンドを使用します。以下について、対応データを含むファイルの名前に置き換えます (異なる場合)。
  - *IIS CertRequest.csr* – ウェブサーバーの CSR を含むファイルの名前
  - *SelfSignedCA.crt* – 自己署名証明書を含むファイルの名前
  - *SelfSignedCA.key* – プライベートキーを含むファイルの名前
  - *IISCert.crt* - ウェブサーバーの署名証明書を含むファイルの名前です。

```
openssl x509 -req -days 365 -in IISCertRequest.csr \
                            -CA SelfSignedCA.crt \
                            -CAkey SelfSignedCA.key \
                            -CAcreateserial \
                            -out IISCert.crt
Signature ok
subject=/ST=IIS-HSM/L=IIS-HSM/OU=IIS-HSM/O=IIS-HSM/CN=IIS-HSM/C=IIS-HSM
Getting CA Private Key
Enter pass phrase for SelfSignedCA.key:
```

上記のステップが完了したら、ウェブサーバーの証明書 (IISCert.crt) と署名証明書 (SelfSignedCA.crt) への署名が完了です。これらのファイルがある場合は、「ステップ 3: ウェブサーバーを設定する」を参照してください。

ステップ 3: ウェブサーバーを設定する

「前のステップ」の最後に作成した HTTPS 証明書を使用するには、IIS ウェブサイトの設定を更新します。これで、 AWS CloudHSMを使用して、SSL/TLS オフロード用に Windows ウェブサーバーソフトウェア (IIS) を設定できます。

自己署名証明書を使用して、CSR に署名した場合は、まずその自己署名証明書を Windows Trusted Root Certification Authorities にインポートする必要があります。

自己署名証明書を Windows Trusted Root Certification Authorities にインポートするには

1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. 自己署名証明書を Windows server にコピーします。
3. Windows Server で、[コントロールパネル] を開きます。
4. [Search Control Panel (コントロールパネルを検索)] に **certificates** と入力します。続いて、[Manage computer certificates (コンピュータ証明書の管理)] を選択します。
5. [Certificates - Local Computer (証明書 – ローカルコンピュータ)] の [Trusted Root Certification Authorities] を展開します。
6. [証明書] を右クリックし、[All Tasks (すべてのタスク)]、[インポート] の順に選択します。
7. [Certificate Import Wizard (証明書インポートウィザード)] で [次へ] を選択します。
8. [Browse (参照)] を選択後、自己署名証明書を検索して選択します。「このチュートリアルの前のステップ」の手順に従って自己署名証明書を作成した場合、自己署名証明書の名前は、`SelfSignedCA.crt` です。開く をクリックします。
9. [次へ] をクリックします。
10. [証明書ストア] で、[Place all certificates in the following store (すべての証明書を以下のストアに配置)] を選択します。次に、[Trusted Root Certification Authorities] が [証明書ストア] で選択されていることを確認します。
11. [Next] を選択し、[Finish] を選択します。

IIS ウェブサイトの設定を更新するには

1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. AWS CloudHSM クライアントデーモンを起動します。
3. このチュートリアルの前のステップ の最後に作成したウェブサーバーの署名付き証明書を、Windowsサーバーにコピーします。
4. Windows Server では、Windows の certreq コマンドを使用して、以下の例のように署名証明書を受け入れます。*IISCert.crt* をウェブサーバーの署名証明書を含むファイルの名前に置き換えます。

```
C:\>certreq -accept IISCert.crt
        SDK Version: 2.03
```

5. Windows Server で [サーバーマネージャー] を起動します。
6. [Server Manager] ダッシュボードの右上隅で、[ツール]、[Internet Information Services (IIS) Manager] の順に選択します。
7. [Internet Information Services (IIS) Manager] ウィンドウで、サーバー名をダブルクリックします。次に、[Sites (サイト)] をダブルクリックします。ウェブサイトを選択します。
8. [SSL Settings (SSL 設定)] を選択します。ウィンドウの右側の [Bindings (バインディング)] を選択します。
9. [Site Bindings] ウィンドウで、[追加] を選択します。
10. [Type (タイプ)] で、[https] を選択します。[SSL 証明書] で、「このチュートリアルの前のステップ」の最後に作成した HTTPS 証明書を選択します。

> **Note**
>
> 証明書のバインディング中にエラーが発生した場合は、サーバーを再起動し、このステップを再試行します。

11. [OK] をクリックします。

ウェブサイトの設定を更新したら、「ステップ 4: HTTPS トラフィックを有効にして証明書を検証する」に移動します。

ステップ 4: HTTPS トラフィックを有効にして証明書を検証する

で SSL/TLS オフロード用にウェブサーバーを設定したら AWS CloudHSM、インバウンド HTTPS トラフィックを許可するセキュリティグループにウェブサーバーインスタンスを追加します。これにより、ウェブブラウザなどのクライアントがウェブサーバーと HTTPS 接続を確立できるようになります。次に、ウェブサーバーに HTTPS 接続を行い、 で SSL/TLS オフロード用に設定した証明書を使用していることを確認します AWS CloudHSM。

トピック

- インバウンド HTTPS 接続の有効化
- 設定した証明書が HTTPS で使用されていることを検証する

インバウンド HTTPS 接続の有効化

クライアント (ウェブブラウザなど) からウェブサーバーに接続するには、インバウンド HTTPS 接続を許可するセキュリティグループを作成します。具体的には、ポート 443 でインバウンドの TCP 接続を許可する必要があります。このセキュリティグループをウェブサーバーに割り当てます。

HTTPS のセキュリティグループを作成してウェブサーバーに割り当てるには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[セキュリティグループ] を選択します。
3. [Create Security Group] を選択します。
4. [Create Security Group] で、以下の操作を行います。
    a. [Security group name] に、作成するセキュリティグループの名前を入力します
    b. (オプション) 作成するセキュリティグループの説明を入力します。
    c. [VPC] で、ウェブサーバーのAmazon EC2インスタンスが含まれている VPC を選択します。
    d. [Add rule (ルールの追加)] を選択します。
    e. [タイプ] で、ドロップダウンウィンドウから [HTTPS] を選択します。
    f. [ソース] には、ソースの場所を入力します。
    g. [Create Security Group] を選択します。
5. ナビゲーションペインで、[インスタンス] を選択します。
6. ウェブサーバーインスタンスの横にあるチェックボックスを選択します。
7. ページの上部で [アクション] ドロップダウンメニューを選択します。[セキュリティ] を選択し、[セキュリティグループの変更] を選択します。
8. [関連付けられたセキュリティグループ] で、検索ボックスを選択して HTTPS 用に作成したセキュリティグループを選択します。次に、[セキュリティグループの追加] を選択します。
9. [Save] を選択します。

設定した証明書が HTTPS で使用されていることを検証する

ウェブサーバーをセキュリティ グループに追加した後、SSL/TLS オフロードが自己署名証明書を使用していることを確認できます。この検証には、ウェブブラウザ、または OpenSSL s_client などのツールを使用できます。

ウェブブラウザで SSL/TLS オフロードを確認するには

1. ウェブブラウザを使用し、サーバーの公開 DNS 名または IP アドレスを使用してウェブサーバーに接続します。アドレスバーの URL が https:// で始まっていることを確認します。例えば **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`** です。

   > **Tip**
   >
   > Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. ウェブブラウザを使用して、ウェブサーバー証明書を表示します。詳細については、次を参照してください。

   - Mozilla Firefox の場合は、Mozilla サポートウェブサイトの「証明書を見る」を参照してください。
   - Google Chrome の場合は、ウェブ開発者向け Google ツールのウェブサイトで「セキュリティの問題を理解する」を参照してください。

   他のウェブブラウザでも、同様の機能を使用してウェブサーバー証明書を表示できる場合があります。

3. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

OpenSSL s_client で SSL/TLS オフロードを確認するには

1. HTTPS を使用してウェブサーバーに接続するには、次の OpenSSL コマンドを実行します。*<#####>* は、ウェブサーバーの公開 DNS 名または IP アドレスに置き換えます。

```
openssl s_client -connect <server name>:443
```

   > **Tip**
   >
   > Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細につ

いては、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. SSL/TLS 証明書が、ウェブサーバーに設定したものであることを確認してください。

これで、ウェブサイトが HTTPS で保護されるようになりました。ウェブサーバーのプライベートキーは、 AWS CloudHSM クラスターの HSM に保存されます。

ロードバランサーを追加するには、「Elastic Load Balancing を使用してロードバランサーを追加する (オプション)」を参照してください。

Elastic Load Balancing を使用してロードバランサーを追加する (オプション)

1 つのウェブサーバーで SSL/TLS オフロードを設定した後で、さらにウェブサーバーを作成し、HTTPS トラフィックをウェブサーバーにルーティングする Elastic Load Balancing ロードバランサーを作成することができます。ロードバランサーは、2 つ以上のウェブサーバーにトラフィックを分散することで、サーバーに対する負荷を軽減できます。また、ロードバランサーはウェブサーバーのヘルス状態をモニタリングして、正常なサーバーにのみトラフィックをルーティングするため、ウェブサイトの可用性も改善できます。ウェブサーバーに障害が発生すると、ロードバランサーはそのウェブサーバーに対するトラフィックのルーティングを自動的に停止します。

トピック

- 2 番目のウェブサーバーのサブネットを作成する
- 2 番目のウェブサーバーを作成する
- ロードバランサーを作成する

2 番目のウェブサーバーのサブネットを作成する

別のウェブサーバーを作成する前に、既存のウェブサーバーと AWS CloudHSM クラスターを含む同じ VPC に新しいサブネットを作成する必要があります。

新しいサブネットを作成するには

1. Amazon VPC コンソールの [サブネット] セクションを開きます。
2. [Create Subnet (サブネットの作成)] を選択します。
3. [Create Subnet] ダイアログボックスで、次の操作を行います。

a. [Name tag] に、サブネットの名前を入力します。

b. VPC では、既存のウェブサーバーと AWS CloudHSM クラスターを含む AWS CloudHSM VPC を選択します。

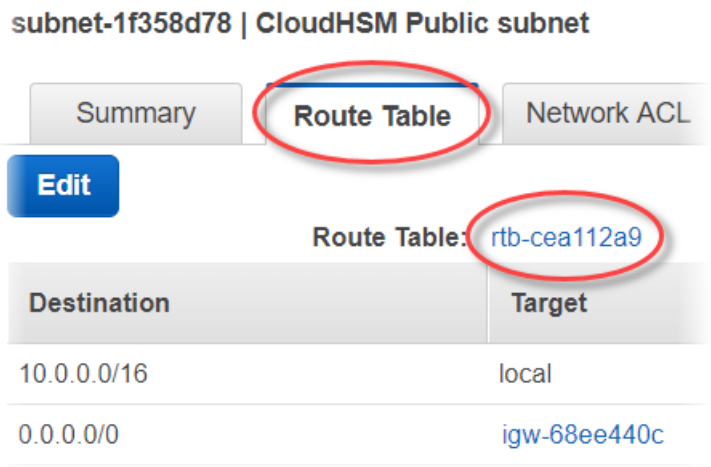
c. [Availability Zone] で、既存のウェブサーバーが含まれているのとは異なるアベイラビリティーゾーンを選択します。

d. [IPv4 CIDR block] に、サブネットで使用する CIDR ブロックを入力します。たとえば、**`10.0.10.0/24`** と入力します。

e. [はい、作成する] を選択します。

4. 既存のウェブサーバーが含まれているパブリックサブネットの横にあるチェックボックスを選択します。これは、前のステップで作成したパブリックサブネットとは異なります。

5. コンテンツペインで、[ルートテーブル] タブを選択します。次に、ルートテーブルのリンクを選択します。



6. ルートテーブルの横にあるチェックボックスをオンにします。

7. [Subnet Associations] タブを選択します。次に、[編集] を選択します。

8. この手順で前に作成したパブリックサブネットの横にあるチェックボックスを選択します。次に、[Save] (保存) を選択します。

2 番目のウェブサーバーを作成する

次の手順を実行し、既存のウェブサーバーと同じ設定で 2 番目のウェブサーバーを作成します。

2 番目のウェブサーバーを作成するには

1. Amazon EC2 コンソールの [インスタンス] セクションを開きます。

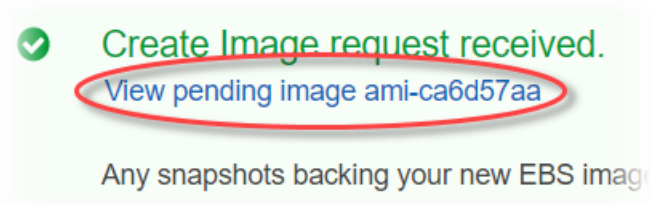
2. 既存のウェブサーバーインスタンスの横にあるチェックボックスをオンにします。
3. [Actions]、[Image]、[Create Image] の順に選択します。
4. [Create Image] ダイアログボックスで、次の操作を行います。

   a. [Image name] には、イメージの名前を入力します。

   b. [Image description] としてイメージの説明を入力します。

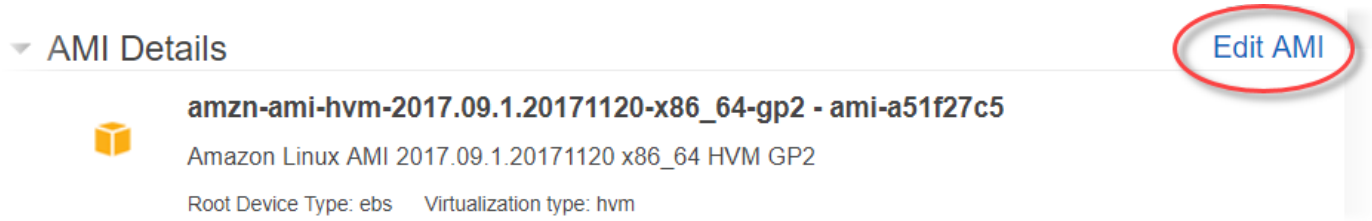
   c. [Create Image] を選択します。このアクションにより、既存のウェブサーバーが再起動されます。

   d. [View pending image ami-*<AMI ID>*] リンクを選択します。



   [Status] 列で、イメージのステータスを確認します。イメージのステータスが [available] になったら (これには数分かかることがあります)、次のステップに進みます。

5. ナビゲーションペインで、[インスタンス] を選択します。
6. 既存のウェブサーバーの横にあるチェックボックスをオンにします。
7. [Actions] を選択し、[Launch More Like This] を選択します。
8. [Edit AMI] を選択します。



9. 左側のナビゲーションペインで、[My AMIs] を選択します。次に、検索ボックスのテキストを消去します。
10. ウェブサーバーイメージの横にある [Select] を選択します。
11. [Yes, I want to continue with this AMI (*<image name>* - ami-*<AMI ID>*)] を選択します。
12. [次へ] をクリックします。
13. インスタンスタイプを選択し、[次: インスタンスの詳細の設定] を選択します。
14. [ステップ 3: インスタンスの詳細の設定] で、以下の操作を行います。

a. [Network] で、既存のウェブサーバーが含まれている VPC を選択します。

b. [Subnet] で、2 番目のウェブサーバー用に作成したパブリックサブネットを選択します。

c. [Auto-assign Public IP] で、[Enable] を選択します。

d. 必要に応じて、残りのインスタンスの詳細を変更します。続いて、[次の手順: ストレージの追加] を選択します。

15. 必要に応じて、ストレージの設定を変更します。次に、[次の手順: タグの追加] を選択します。

16. 必要に応じて、タグを追加または編集します。次に、[次の手順: セキュリティグループの設定] を選択します。

17. [Step 6: Configure Security Group] で、以下の操作を行います。

a. [セキュリティグループの割り当て] で、[既存のセキュリティグループを選択する] を選択します。

b. cloudhsm-***<cluster ID>***-SG というという名前のセキュリティグループの横にあるチェックボックスを選択します。 AWS CloudHSM は、クラスターを作成する 際に、代理でこのセキュリティグループを作成しました。ウェブサーバーインスタンスからクラスターの HSM への接続を許可するために、このセキュリティグループを選択する必要があります。

c. インバウンド HTTPS トラフィックを許可するセキュリティグループの横にあるチェックボックスをオンにします。このセキュリティグループは前に作成済みです。

d. (オプション) ネットワークからの SSH (Linux) または RDP (Windows) の受信トラフィックを許可するセキュリティ グループの横にあるチェックボックスを選択します。つまり、セキュリティグループは、ポート 22 (Linux の SSH 用) またはポート 3389 (Windows の RDP 用) のインバウンド TCP トラフィックを許可する必要があります。さもないと、クライアントインスタンスに接続することはできません。このようなセキュリティグループがない場合は、作成する必要があります。その後でクライアントインスタンスに割り当てます。

[Review and Launch] (確認と起動) を選択します。

18. インスタンスの詳細を確認し、[Launch] を選択します。

19. インスタンスを起動するために、既存のキーペアを使用するか、新しいキーペアを作成するか、キーペアを使用しないかを選択します。

- 既存のキーペアを使用するには、以下の操作を行います。

  1. [Choose an existing key pair] (既存のキーペアの選択) をクリックします。

  2. [Select a key pair] で、使用するキーペアを選択します。

3. [I acknowledge that I have access to the selected private key file (*<private key file name>*.pem), and that I will not be able to log in my instance.] の横のチェックボックスを選択します。

- 新しいキーペアを作成するには、以下の操作を行います。
  1. 新規キーペア作成を選択します。
  2. [Key pair name] にキーペアの名前を入力します。
  3. [Download Key Pair] を選択して、プライベートキーファイルを安全でアクセス可能な場所に保存します。

  > ⚠ Warning
  >
  > この時点以降、プライベートキーファイルをダウンロードすることはできません。この時点でプライベートキーファイルをダウンロードしないと、以後はクライアントインスタンスにアクセスできなくなります。

- キーペアを使用しないでインスタンスを起動するには、次の操作を行います。
  1. [Proceed without a key pair] を選択します。
  2. [I acknowledge that I will not be able to connect to this instance unless I already know the password built into this AMI] の横にあるチェックボックスをオンにします。

[Launch Instances] (インスタンスを起動) をクリックします。

ロードバランサーを作成する

以下の手順を完了し、HTTPS トラフィックをウェブサーバーにルーティングする Elastic Load Balancing ロードバランサを作成します。

ロードバランサーを作成するには

1. Amazon EC2 コンソールで [ロードバランサー] セクションを開きます。
2. [Create Load Balancer] を選択します。
3. [Network Load Balancer] セクションで、[Create] を選択します。
4. [Step 1: Configure Load Balancer] で、以下の操作を行います。

   a. [Name] に、作成するロードバランサーの名前を入力します。

b. [Listeners] セクションの [Load Balancer Port] で、値を **443** に変更します。

c. [Availability Zones] セクションの [VPC] で、ウェブサーバーが含まれている VPC を選択します。

d. [Availability Zones] セクションで、ウェブサーバーが含まれているサブネットを選択します。

e. [Next: Configure Routing] を選択します。

5. [Step 2: Configure Routing] で、以下の操作を行います。

a. [Name] に、作成するターゲットグループの名前を入力します。

b. [Port] で、値を **443** に変更します。

c. [Next: Register Targets] を選択します。

6. [Step 3: Register Targets] で、次の操作を行ないます。

a. [Instances] セクションで、ウェブサーバーインスタンスの横にあるチェックボックスを選択します。次に、[Add to registered] を選択します。

b. [次へ: レビュー] を選択します。

7. ロードバランサーの詳細を確認し、[Create] を選択します。

8. ロードバランサーが正常に作成されたら、[Close] を選択します。

上記の手順を完了すると、Amazon EC2 コンソールに Elastic Load Balancing ロードバランサが表示されます。

ロードバランサーの状態がアクティブになると、ロードバランサーが動作していることを確認できます。つまり、 AWS CloudHSMでの SSL/TLS オフロードを使用して HTTPS トラフィックがウェブサーバーに送信されていることを検証できます。これは、ウェブブラウザや OpenSSL s_client などのツールを使用して行うことができます。

ロードバランサーが動作していることをウェブブラウザで確認するには

1. Amazon EC2 コンソールで、先ほど作成したロードバランサーの DNS 名 を見つけてください。次に、この DNS 名を選択してコピーします。

2. Mozilla Firefox や Google Chrome などのウェブブラウザで、ロードバランサーの DNS 名を使用してロードバランサーに接続します。アドレスバーの URL が https:// で始まっていることを確認します。

> **Tip**
>
> Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

3. ウェブブラウザを使用して、ウェブサーバー証明書を表示します。詳細については、次を参照してください。

   - Mozilla Firefox の場合は、Mozilla サポートウェブサイトの「証明書を見る」を参照してください。
   - Google Chrome の場合は、ウェブ開発者向け Google ツールのウェブサイトで「セキュリティの問題を理解する」を参照してください。

   他のウェブブラウザでも、同様の機能を使用してウェブサーバー証明書を表示できる場合があります。

4. 証明書が、ウェブサーバーで使用するように設定したものであることを確認します。

ロードバランサーが動作していることを OpenSSL s_client で確認するには

1. 以下の OpenSSL コマンドにより、HTTPS を使用してロードバランサーに接続します。*<DNS name>* を、使用しているロードバランサーの DNS 名に置き換えます。

```
openssl s_client -connect <DNS name>:443
```

> **Tip**
>
> Amazon Route 53 などの DNS サービスを使用して、ウェブサイトのドメイン名 (https://www.example.com/ など) をウェブサーバーにルーティングできます。詳細については、Amazon Route 53 開発者ガイド の Amazon EC2 インスタンスへのトラフィックのルーティング または DNS サービスのドキュメントを参照してください。

2. 証明書が、ウェブサーバーで使用するように設定したものであることを確認します。

これで、ウェブサーバーのプライベートキーが AWS CloudHSM クラスターの HSM に保存され、HTTPS で保護されているウェブサイトができました。ウェブサイトは、2 つのウェブサーバーとロードバランサーにより、効率と可用性が向上します。

AWS CloudHSMを使用した認証機関 (CA) として Windows Server を設定する

公開鍵基盤 (PKI) において、認証機関 (CA) は、デジタル証明書を発行する信頼されたエンティティです。これらのデジタル証明書は、公開鍵暗号方式およびデジタル署名を使用して、パブリックキーを ID (個人または組織) にバインドします。CA を操作するには、CA によって発行された証明書に署名するプライベートキーを保護して、信頼関係を維持する必要があります。プライベートキーを AWS CloudHSM クラスター内の HSM に保存し、HSM を使用して暗号化署名オペレーションを実行できます。

このチュートリアルでは、Windows Server と を使用して CA AWS CloudHSM を設定します。Windows 用 AWS CloudHSM クライアントソフトウェアを Windows Server にインストールしたら、Windows Server に Active Directory Certificate Services (AD CS) のロールを追加します。このロールを設定するときは、 AWS CloudHSM キーストレージプロバイダー (KSP) を使用して CA のプライベートキーを作成し、 AWS CloudHSM クラスターに保存します。KSP は、Windows サーバーを AWS CloudHSM クラスターに接続するブリッジです。最後のステップで、Windows Server CA を使用して、証明書署名リクエスト (CSR) に署名します。

詳細については、次のトピックを参照してください。

トピック

- Windows Server CA ステップ 1: 前提条件の設定
- Windows Server CA ステップ 2: AWS CloudHSM を使用する Windows Server CA を作成する
- Windows Server CA ステップ 3: を使用して Windows Server CA で証明書署名リクエスト (CSR) に署名する AWS CloudHSM

Windows Server CA ステップ 1: 前提条件の設定

で Windows Server を認証局 (CA) としてセットアップするには AWS CloudHSM、以下が必要です。

- 少なくとも 1 つの HSM を持つアクティブな AWS CloudHSM クラスター。

- Windows 用の AWS CloudHSM クライアントソフトウェアがインストールされた Windows Server オペレーティングシステムを実行する Amazon EC2 インスタンス。このチュートリアルでは Microsoft Windows Server 2016 を使用します。
- HSM で CA のプライベートキーを所有および管理するための暗号化ユーザー (CU)。

を使用して Windows Server CA の前提条件を設定するには AWS CloudHSM

1. 「開始」のステップを完了します。Amazon EC2 クライアントを起動するときは、Windows Server AMIを選択します。このチュートリアルでは Microsoft Windows Server 2016 を使用します。これらのステップを完了すると、少なくとも 1 つの HSM を含むアクティブなクラスターが提供されます。Windows 用のクライアントソフトウェアがインストールされた Windows Server を実行している Amazon EC2 AWS CloudHSM クライアントインスタンスもあります。
2. (オプション) 他の HSM をクラスターに追加します。詳細については、「HSM の追加」を参照してください。
3. クライアントインスタンスに接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
4. 「Managing HSM users with CloudHSM CLI」または「Managing HSM users with CloudHSM Management Utility (CMU)」を使用して、Crypto User (CU) を作成します。CU のユーザー名とパスワードを書き留めます。次のステップを完了するために必要になります。
5. 前のステップで作成した CU ユーザー名とパスワードを使用して、HSM のログイン認証情報を設定します。
6. ステップ 5 で Windows 認証情報マネージャーを使用して HSM 認証情報を設定した場合は、psexec.exeからダウンロード SysInternals して、NT Authority\SYSTEM として次のコマンドを実行します。

```
psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe"
 --username <USERNAME> --password <PASSWORD>
```

   <USERNAME> と *<PASSWORD>* を HSM 認証情報に置き換えてください。

を使用して Windows Server CA を作成するには AWS CloudHSM、「」に移動しますWindows Server CA の作成。

Windows Server CA ステップ 2: AWS CloudHSM を使用する Windows Server CA を作成する

Windows Server CA を作成するには、Active Directory 証明書サービス (AD CS) ロールを Windows Server に追加します。このロールを追加するときは、 AWS CloudHSM キーストレージプロバイダー (KSP) を使用して CA のプライベートキーを作成し、 AWS CloudHSM クラスターに保存します。

> **Note**
>
> Windows Server CA を作成すると、ルート CA または下位 CA の作成を選択できます。通常、この決定はパブリックキーインフラストラクチャの設計および組織のセキュリティポリシーに基づいて行います。このチュートリアルではルート CA を作成する方法を簡単に説明します。

AD CS ロールを Windows Server に追加して CA のプライベートキーを作成するには

1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. Windows Server で [サーバーマネージャー] を起動します。
3. [サーバー マネージャー] ダッシュボードで、[役割と機能の追加] を選択します。
4. [開始する前に] 情報を読み、[次へ] を選択します。
5. [インストールのタイプ] ページで、[ロールベースまたは機能ベースのインストール] を選択します。次いで、[次へ] を選択します。
6. [サーバーの選択] で、[Select a server from the server pool (サーバープールからサーバーを選択する)] を選択します。次いで、[次へ] を選択します。
7. [Server Roles (サーバーの役割位)] で、以下を実行します。
   a. [Active Directory Certificate Services] を選択します。
   b. [Add features that are required for Active Directory Certificate Services (Active Directory Certificate Services に必要な機能を追加する)] で、[機能の追加] を選択します。
   c. [次へ] を選択してサーバーロールの選択を完了します。
8. [機能] で、デフォルトを受け入れて、[次へ] を選択します。
9. [AD CS] で、以下を実行します

a. [次へ] をクリックします。

b. [証明機関] を選択してから、[次へ] を選択します。

10. [確認] で確認情報を読み、[インストール] を選択します。ウィンドウを閉じないでください。
11. 強調表示された [Configure Active Directory Certificate Services on the destination server (ターゲットサーバーの Active Directory Certificate Services を設定する)] リンクを選択します。
12. [認証情報] で、表示される認証情報を検証または変更します。次いで、[次へ] を選択します。
13. [役割サービス] で、[証明機関] を選択します。次いで、[次へ] を選択します。
14. [Setup Type (セットアップタイプ)] で、[Standalone CA (スタンドアロン CA)] を選択します。次いで、[次へ] を選択します。
15. [CA Type (CA タイプ)] で、[Root CA (ルート CA)] を選択します。次いで、[次へ] を選択します。

> **Note**
>
> パブリックキーインフラストラクチャの設計と組織のセキュリティポリシーに基づいて、ルート CA または下位 CA の作成を選択できます。このチュートリアルではルート CA を作成する方法を簡単に説明します。

16. [プライベートキー] で、[Create a new private key (新しいプライベートキーを作成する)] を選択します。次いで、[次へ] を選択します。
17. [暗号化] で、以下の操作を実行します。

    a. [Select a cryptographic provider (暗号化プロバイダーを選択する)] で、メニューから [Cavium Key Storage Provider (Cavium キーストレージプロバイダー)] のいずれかを選択します。これらは、 AWS CloudHSM キーストレージプロバイダーです。たとえば、[RSA#Cavium Key Storage Provider (RSA#Cavium キーストレージプロバイダー)] を選択できます。

    b. [Key length (キーの長さ)] で、キーの長さのオプションを 1 つ選択します。

    c. [Select the hash algorithm for signing certificates issued by this CA (この CA によって発行された証明書に署名するためのハッシュアルゴリズムを選択する)] で、ハッシュアルゴリズムのオプションを 1 つ選択します。

    [次へ] をクリックします。

18. [CA Name (CA 名)] で、以下を実行します。

a. (省略可能) 共通名を編集します。

b. (省略可能) 識別子名サフィックスを入力します。

[次へ] をクリックします。

19. [有効期間] で、期間を年、月、週、または日で指定します。次いで、[次へ] を選択します。
20. [Certificate Database (証明書データベース)] では、デフォルト値をそのままにするか、必要に応じてデータベースとデータベースログの場所を変更できます。次いで、[次へ] を選択します。
21. [確認] で、CA に関する情報を確認し、[構成する] を選択します。
22. [閉じる] を選択し、もう一度 [閉じる] を選択します。

これで、 を持つ Windows Server CA ができました AWS CloudHSM。CA を使用して証明書署名リクエスト (CSR) に署名する方法の詳細については、「CSR への署名」を参照してください。

Windows Server CA ステップ 3: を使用して Windows Server CA で証明書署名リクエスト (CSR) に署名する AWS CloudHSM

で Windows Server CA を使用して AWS CloudHSM 、証明書署名リクエスト (CSR) に署名できます。これらのステップを完了するには、有効な CSR が必要です。CSR は以下を含むいくつかの方法で作成できます。

- OpenSSL の使用
- Windows Server インターネット インフォメーション サービス (IIS) マネージャを使用する
- Microsoft マネジメントコンソールの証明書スナップインを使用する
- Windows で certreq コマンドラインユーティリティを使用する

CSR を作成する手順は、このチュートリアルの範囲外です。CSR がある場合は、Windows Server CA を使用して署名できます。

Windows Server CA を使用して CSR に署名するには

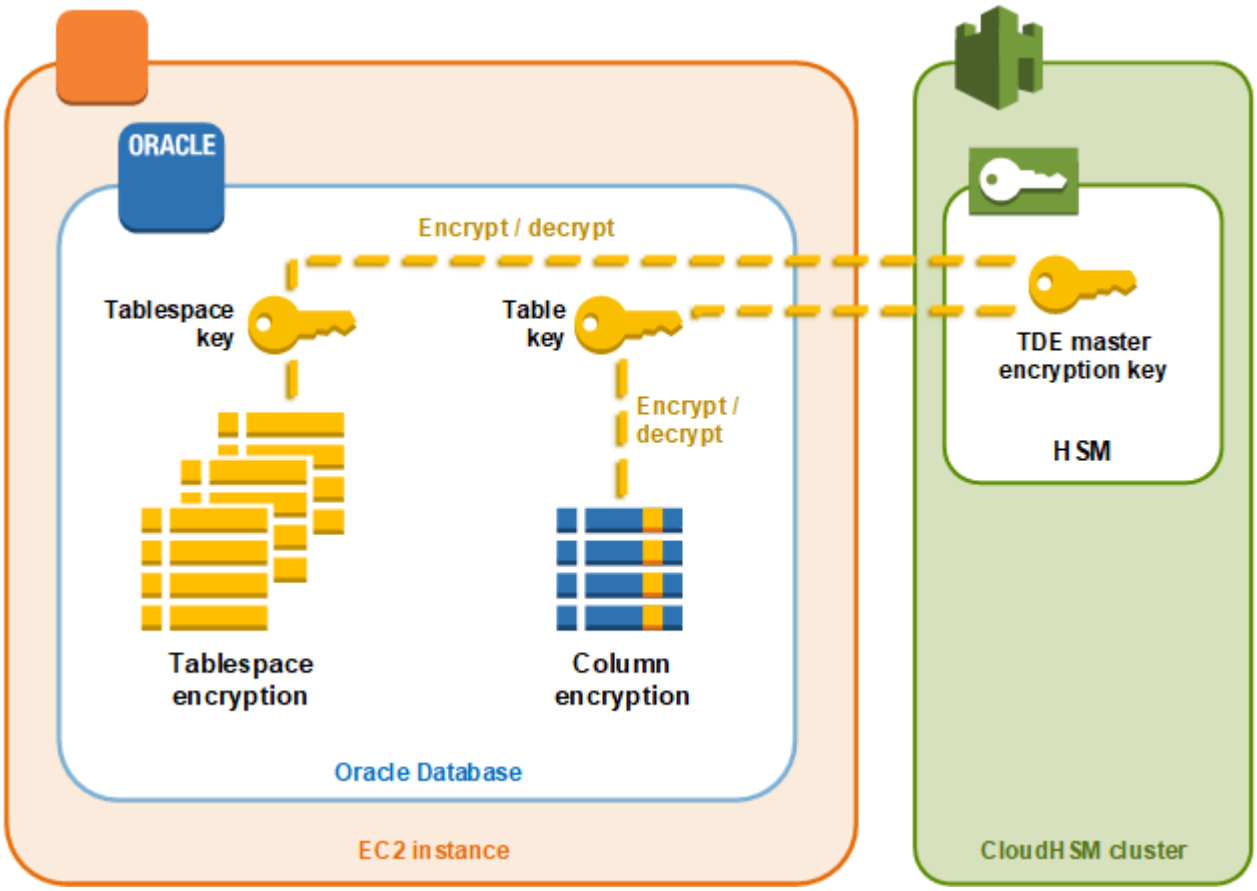
1. Windows Server に接続していない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. Windows Server で [サーバーマネージャー] を起動します。
3. [サーバー マネージャー] ダッシュボードの右上隅で、[ツール]、[証明機関] の順に選択します。

4. [証明機関] ウィンドウで、コンピュータ名を選択します。
5. [アクション] メニューで、[すべてのタスク]、[新しい要求の送信] の順に選択します。
6. CSR ファイルを選択して、[開く] を選択します。
7. [証明機関] ウィンドウで、[保留中の要求] をダブルクリックします。
8. 保留中のリクエストを選択します。次に、[アクション] メニューで、[すべてのタスク]、[発行] の順に選択します。
9. [証明機関] ウィンドウで、[Issued Requests (発行済みの要求)] をダブルクリックします。
10. (オプション) 署名付き証明書をファイルにエクスポートするには、次のステップを実行します。
    a. [証明機関] ウィンドウで、証明書をダブルクリックします。
    b. [詳細] タブ、[Copy to File (ファイルにコピー)] の順に選択します。
    c. [Certificate Export Wizard (証明書のエクスポートウィザード)] の手順に従います。

これで、 を持つ Windows Server CA と AWS CloudHSM、Windows Server CA によって署名された有効な証明書ができました。

AWS CloudHSMでの Oracle Database の透過的なデータ暗号化 (TDE)

透過的なデータ暗号化 (TDE) を使用して、データベースファイルを暗号化します。TDE を使用すると、データベースソフトウェアはデータをディスクに保存する前に暗号化します。データベースのテーブル列またはテーブルスペースのデータは、テーブルキーまたはテーブルスペースキーで暗号化されています。一部バージョンの Oracle のデータベースソフトウェアには TDE を提供します。Oracle TDE では、これらのキーは TDE マスター暗号化キーを使用して暗号化されます。TDE マスター暗号化キーを AWS CloudHSM クラスターの HSMs に保存することで、セキュリティを強化できます。



このソリューションでは、Amazon EC2 インスタンスにインストールされている Oracle Database を使用します。Oracle Database は PKCS #11 のAWS CloudHSM ソフトウェアライブラリと統合し、TDE マスターキーをクラスター内の HSM に保存します。

> **Important**
>
> - Amazon EC2 インスタンスに Oracle Database をインストールすることを推奨します。

Oracle TDE と AWS CloudHSMを統合するには、次のステップを実行します。

と Oracle TDE の統合を設定するには AWS CloudHSM

1. 「前提条件の設定」の手順に従って、環境を準備します。
2. の手順に従ってデータベースの設定、 AWS CloudHSM クラスターと統合するように Oracle Database を設定します。

を使用した Oracle TDE AWS CloudHSM: 前提条件の設定

Oracle TDE を と統合するには AWS CloudHSM、以下が必要です。

- 少なくとも 1 つの HSM を持つアクティブな AWS CloudHSM クラスター。
- 次のソフトウェアがインストールされた Amazon Linux オペレーティングシステムを実行している Amazon EC2 インスタンス。 :
  - AWS CloudHSM クライアントおよびコマンドラインツール。
  - PKCS #11 用の AWS CloudHSM ソフトウェアライブラリ。
  - Oracle Database. は Oracle TDE 統合 AWS CloudHSM をサポートしています。Client SDK 5.6 以降は、Oracle Database 19c 用 Oracle TDE をサポートしています。Client SDK 3 は Oracle データベースバージョン 11g および 12c の Oracle TDE をサポートします。
- クラスター内の HSM で TDE マスター暗号化キーを所有および管理するための暗号化ユーザー (CU)。

これらの前提条件のすべてを設定するには、以下のステップを実行します。

Oracle TDE と の統合の前提条件を設定するには AWS CloudHSM

1. 「開始」のステップを完了します。これを行うと、1 つの HSM を含むアクティブなクラスターが提供されます。Amazon Linux オペレーティングシステムで実行される Amazon EC2 インスタンスも作成されます。 AWS CloudHSM クライアントツールとコマンドラインツールもインストールされ、設定されます。
2. (オプション) 他の HSM をクラスターに追加します。詳細については、「HSM の追加」を参照してください。
3. Amazon EC2 クライアントインスタンスに接続し、以下を実行します。
   a. PKCS #11 用の AWS CloudHSM ソフトウェアライブラリをインストールします。
   b. Oracle Database をインストールします。詳細については、Oracle Database のドキュメントを参照してください。Client SDK 5.6 以降は、Oracle Database 19c 用 Oracle TDE をサポートしています。Client SDK 3 は Oracle データベースバージョン 11g および 12c の Oracle TDE をサポートします。
   c. cloudhsm_mgmt_util コマンドラインツールを使用し、クラスターで暗号化ユーザー (CU) を作成します。CU の作成に関する詳細については、CMU で HSM ユーザーを管理する方法と HSM ユーザーの管理 を参照してください。

以上のステップを完了すると、データベースの設定 できます。

を使用した Oracle TDE AWS CloudHSM: データベースを設定し、マスター暗号化キーを生成する

Oracle TDE を AWS CloudHSM クラスターと統合するには、以下のトピックを参照してください。

1. Oracle Database 設定の更新により、クラスターの HSM を外部セキュリティモジュールとして使用します。外部セキュリティモジュールの詳細については、Oracle Database Advanced Security ガイドの「透過的データ暗号化の概要」を参照してください。
2. クラスターの HSM で Oracle TDE マスター暗号化キーの生成を行います。

Oracle Database 設定の更新

クラスターで HSM を外部セキュリティモジュールとして使用するように Oracle Database の設定を更新するには、次のステップを実行します。外部セキュリティモジュールの詳細については、Oracle Database Advanced Security ガイドの「透過的データ暗号化の概要」を参照してください。

Oracle 設定を更新するには

1. Amazon EC2 クライアントインスタンスに接続します。これは、Oracle Database をインストールした先のインスタンスです。
2. `sqlnet.ora` というファイルのバックアップコピーを作成します。このファイルの場所については、Oracle のドキュメントを参照してください。
3. テキストエディタを使用して、`sqlnet.ora` というファイルを編集します。次の行を追加します。ファイルの既存の行が `encryption_wallet_location` で始まる場合は、既存の行を次の行に置き換えます。

   ```
   encryption_wallet_location=(source=(method=hsm))
   ```

   ファイルを保存します。
4. 次のコマンドを実行して、Oracle Database が AWS CloudHSM PKCS #11 ソフトウェアライブラリのライブラリファイルを検索するディレクトリを作成します。

   ```
   sudo mkdir -p /opt/oracle/extapi/64/hsm
   ```

5. 次のコマンドを実行して、PKCS #11 ファイルの AWS CloudHSM ソフトウェアライブラリを前のステップで作成したディレクトリにコピーします。

```
sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11.so /opt/oracle/extapi/64/hsm/
```

> **Note**
>
> /opt/oracle/extapi/64/hsm ディレクトリに含めるライブラリファイルは 1 つに限られます。そのディレクトリに存在する他のファイルを削除します。

6. 次のコマンドを実行して、/opt/oracle ディレクトリおよびその内容すべての所有権を変更します。

```
sudo chown -R oracle:dba /opt/oracle
```

7. Oracle Database を起動します。

Oracle TDE マスター暗号化キーの生成

クラスターの HSM で Oracle TDE マスターキーを生成するには、次の手順を実行します。

マスターキーを生成するには

1. 次のコマンドを使用して、Oracle SQL*Plus を開きます。プロンプトが表示されたら、Oracle Database のインストール時に設定したシステムパスワードを入力します。

```
sqlplus / as sysdba
```

> **Note**
>
> Client SDK 3 の場合、マスターキーを生成するたびに CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE 環境変数を設定する必要があります。この変数は、マスターキーの生成にのみ必要です。詳細については、サードパーティーアプリケーションの統合に関する既知の問題 の「問題：Oracleは CKA_MODIFIABLE マスターキーの生成中に PCKS #11 属性を設定しますが、HSM はそれをサポートしていません」を参照してください。

2. 次の例に示すように、マスター暗号化キーを作成する SQL ステートメントを実行します。使用しているバージョンの Oracle Database に対応するステートメントを使用します。*<CU user name>* を暗号化ユーザー (CU) のユーザー名に置き換えます。*<password>* を CU パスワードに置き換えます。

> **Important**
>
> 次のコマンドは 1 回のみ実行します。コマンドを実行するたびに、新しいマスター暗号化キーが作成されます。

- Oracle Database バージョン 11 の場合は、次の SQL ステートメントを実行します。

```
SQL> alter system set encryption key identified by "<CU user name>:<password>";
```

- Oracle Database バージョン 12 およびバージョン 19c の場合は、次の SQL ステートメントを実行します。

```
SQL> administer key management set key identified by "<CU user
 name>:<password>";
```

レスポンスが `System altered` または `keystore altered` の場合は、Oracle TDE のマスターキーが正常に生成および設定されています。

3. (オプション) 次のコマンドを実行して Oracle ウォレットのステータスを確認します。

```
SQL> select * from v$encryption_wallet;
```

ウォレットが開いていない場合は、次のいずれかのコマンドを使用して開きます。*<CU user name>* を暗号化ユーザー (CU) の名前に置き換えます。*<password>* を CU パスワードに置き換えます。

- Oracle 11 の場合は、次のコマンドを実行してウォレットを開きます。

```
SQL> alter system set encryption wallet open identified by "<CU user
 name>:<password>";
```

手動でウォレットを閉じるには、次のコマンドを実行します。

```
SQL> alter system set encryption wallet close identified by "<CU user
 name>:<password>";
```

- Oracle 12 および Oracle 19c の場合は、次のコマンドを実行してウォレットを開きます。

```
SQL> administer key management set keystore open identified by "<CU user
 name>:<password>";
```

手動でウォレットを閉じるには、次のコマンドを実行します。

```
SQL> administer key management set keystore close identified by "<CU user
 name>:<password>";
```

SignTool で Microsoft を使用してファイルに署名 AWS CloudHSM する

暗号化やパブリックキー基盤 (PKI) では、デジタル署名は、データが信頼されたエンティティより送信されたことを確認することを目的として使用されます。署名は、データが送信中に改ざんされていないことも示します。署名とは、送信者のプライベートキーを使用して生成された暗号化ハッシュを指します。受信者は、ハッシュ署名を送信者のパブリックキーで復号することで、データの整合性を検証できます。また、送信者は、デジタル証明書を管理する責任があります。デジタル証明書は、送信者のプライベートキーの所有者を証明し、復号に必要なパブリックキーを受信者に渡します。プライベートキーが送信者によって所有されている限り、署名は信頼できます。 は、これらのキーを排他的なシングルテナントアクセスで保護するために、安全な FIPS 140-2 レベル 3 検証済みハードウェア AWS CloudHSM を提供します。

多くの組織では SignTool、Microsoft を使用しています。Microsoft は、コード署名プロセスを簡素化するために、ファイルに署名、検証、タイムスタンプするコマンドラインツールです。を使用して AWS CloudHSM 、キーペアが で必要になるまで安全に保存できるため SignTool、データに署名するためのワークフローを簡単に自動化できます。

以下のトピックでは、 SignTool で を使用する方法の概要を説明します AWS CloudHSM。

トピック

- SignTool AWS CloudHSM ステップ 1: 前提条件を設定する Microsoft
- AWS CloudHSM ステップ 2: 署名証明書を作成する SignTool Microsoft

- AWS CloudHSM ステップ 3: ファイルに署名 SignTool する Microsoft

SignTool AWS CloudHSM ステップ 1: 前提条件を設定する Microsoft

SignTool で Microsoft を使用するには AWS CloudHSM、以下が必要です。

- Windows オペレーティングシステムが実行されている Amazon EC2 クライアントインスタンス。
- 認証機関 (CA)。自己管理、またはサードパーティープロバイダーが作成したもの。
- EC2 インスタンスと同じ仮想パブリッククラウド (VPC) 内のアクティブな AWS CloudHSM クラスター。クラスターには、少なくとも 1 つの HSM が存在している必要があります。
- AWS CloudHSM クラスター内のキーを所有および管理するための暗号ユーザー (CU)。
- 未署名のファイルまたは実行可能ファイル。
- Microsoft Windowsソフトウェア開発キット (SDK)。

Windows AWS CloudHSM で を使用するための前提条件を設定するには SignTool

1. このガイドの「開始方法」セクションの指示に従って、Windows EC2 インスタンスと AWS CloudHSM クラスターを起動します。
2. 独自の Windows Server CA をホストする場合は、「 で Windows Server を認証機関として設定する」のステップ 1 と 2 AWS CloudHSMに従います。それ以外の場合は、公的に信頼されたサードパーティーの CA をそのまま使用します。
3. 次のバージョンの Microsoft Windows SDK を Windows EC2 インスタンスにダウンロードしてインストールします。

   - Microsoft Windows SDK 10
   - Microsoft Windows SDK 8.1
   - Microsoft Windows SDK 7

   SignTool 実行可能ファイルは、デスクトップアプリケーションのインストール機能用の Windows SDK Signing Tool に含まれます。不要な場合は、他の機能をインストール対象から除外することができます。デフォルトのインストール場所は次のとおりです。

```
C:\Program Files (x86)\Windows Kits\<SDK version>\bin\<version number>\<CPU
 architecture>\signtool.exe
```

Microsoft Windows SDK、 AWS CloudHSM クラスター、および CA を使用して署名証明書を作成できるようになりました。

AWS CloudHSM ステップ 2: 署名証明書を作成する SignTool Microsoft

EC2 インスタンスに Windows SDK をダウンロードしたら、これを使用して証明書署名リクエスト (CSR) を生成することができます。CSR は、未署名の証明書であり、署名用に最終的に CA に渡されます。この例では、Windows SDK に含まれる`certreq` 実行可能ファイルを使用して、CSR を生成します。

`certreq` 実行可能ファイルを使用して CSR を生成するには

1. Windows EC2 インスタンスに接続されていない場合は、接続します。詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスに接続する」を参照してください。 Amazon EC2
2. 以下の行に含まれる `request.inf` ファイルを作成します。`Subject` 情報をお客様の組織の情報に置き換えます。各パラメータの説明については、「Microsoft のドキュメント」を参照してください。

```
[Version]
Signature= $Windows NT$
[NewRequest]
Subject = "C=<Country>,CN=<www.website.com>,O=<Organization>,OU=<Organizational-
Unit>,L=<City>,S=<State>"
RequestType=PKCS10
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
ProviderName = Cavium Key Storage Provider
KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
MachineKeySet = True
Exportable = False
```

3. `certreq.exe` を実行します。この例では、CSR を `request.csr` として保存します。

```
certreq.exe -new request.inf request.csr
```

内部的には、 AWS CloudHSM クラスターに新しいキーペアが生成され、そのペアのプライベートキーを使用して CSR が作成されます。

4. CA に CSR を送ります。Windows Server CA を使用している場合は、次のステップを行います。

   a. 次のコマンドを入力して、CA ツールを開きます。

   ```
   certsrv.msc
   ```

   b. 新しいウィンドウで、CA サーバーの名前を右クリックします。[すべてのタスク]、[Submit new request (新しいリクエストの送信)] の順に選択します。

   c. `request.csr` の場所に移動し、[開く] を選択します。

   d. サーバー CA メニューから、[保留中のリクエスト] フォルダを表示します。先ほど作成したリクエストを右クリックし、[すべてのタスク] で [問題] を選択します。

   e. [Issued Certificates (発行済みの証明書)] フォルダ ([保留中のリクエスト] フォルダの上) に移動します。

   f. [開く] を選択して証明書を表示し、[詳細] タブを選択します。

   g. [Copy to File (ファイルにコピー)] を選択して、証明書のエクスポートウィザードを起動します。DER でエンコードされた X.509 ファイルを `signedCertificate.cer` として安全な場所に保存します。

   h. CA ツールを終了し、次のコマンドを使用して、証明書ファイルを Windows の Personal Certificate Store に移動します。他のアプリケーションで使用できます。

   ```
   certreq.exe -accept signedCertificate.cer
   ```

これで、インポートしたファイルを使用して、ファイルに署名する ことができます。

AWS CloudHSM ステップ 3: ファイルに署名 SignTool する Microsoft

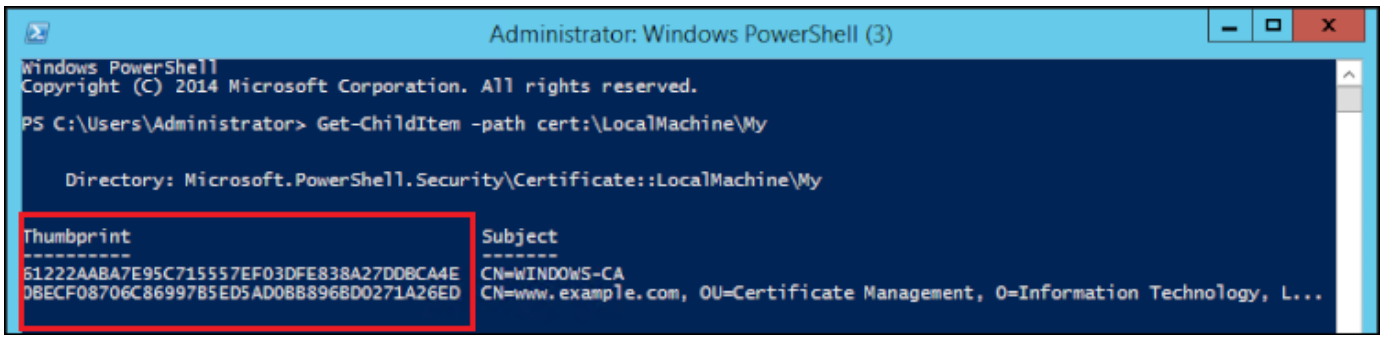
これで、 SignTool とインポートした証明書を使用してサンプルファイルに署名する準備ができました。そのためには、証明書の SHA-1 ハッシュ、またはサムプリントを把握しておく必要があります。サムプリントは、 が SignTool によって検証された証明書のみを使用するようにするために使用します AWS CloudHSM。この例では、 PowerShell を使用して証明書のハッシュを取得します。また、CA の GUI または Windows SDK の `certutil` 実行可能ファイルを使用することもできます。

証明書のサムプリントを取得し、それを使用してファイルに署名するには

1. 管理者 PowerShell として を開き、次のコマンドを実行します。

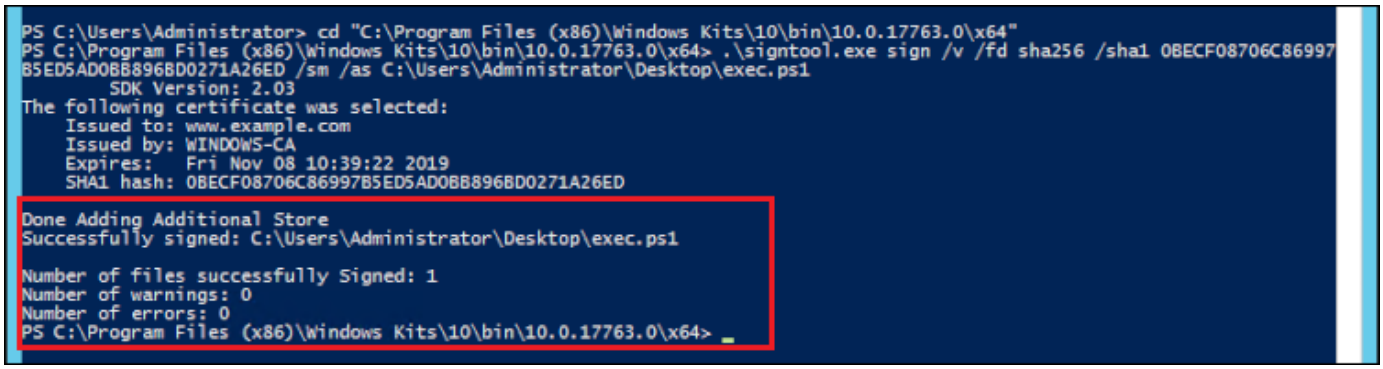
```
Get-ChildItem -path cert:\LocalMachine\My
```

返った Thumbprint をコピーします。



2. PowerShell を含む 内のディレクトリに移動しますSignTool.exe。デフォルトの場所は C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64 です。
3. 最後に、次のコマンドを実行してファイルに署名します。コマンドが成功すると、 は成功メッセージ PowerShell を返します。

```
signtool.exe sign /v /fd sha256 /sha1 <thumbprint> /sm C:\Users\Administrator
\Desktop\<test>.ps1
```



4. (オプション) ファイルの署名を検証するには、次のコマンドを使用します。

```
signtool.exe verify /v /pa C:\Users\Administrator\Desktop\<test>.ps1
```

Java キーツールとJarsigner

AWS CloudHSM は、クライアント SDK 3 およびクライアント SDK 5 を介して Java Keytool および Jarsigner ユーティリティとの統合を提供します。これらのツールを使用する手順は、現在ダウンロードしている Client SDK のバージョンによって異なります。

- Client SDK 5 を使用して Java Keytool および Jarsigner と統合する
- Client SDK 3 を使用して Java Keytool および Jarsigner と統合する

Client SDK 5 を使用して Java Keytool および Jarsigner と統合する

AWS CloudHSM key store は、HSM のキーに関連付けられた証明書を、 `keytool`や などのサードパーティーツールを使用して使用する専用 JCE キーストアです`jarsigner`。証明書はパブリックで非機密データであるため、 AWS CloudHSM HSM に証明書を保存しません。 AWS CloudHSM キーストアは証明書をローカルファイルに格納し、証明書を HSM の対応するキーにマッピングします。

AWS CloudHSM キーストアを使用して新しいキーを生成すると、ローカルキーストアファイルにはエントリが生成されず、キーは HSM に作成されます。同様に、 AWS CloudHSM キーストアを使用してキーを検索すると、検索が HSM に渡されます。証明書を AWS CloudHSM キーストアに保存すると、プロバイダーは対応するエイリアスを持つキーペアが HSM に存在することを確認し、提供された証明書を対応するキーペアに関連付けます。

トピック

- 前提条件
- AWS CloudHSM keytool でのキーストアの使用
- Jarsigner での AWS CloudHSM キーストアの使用
- 既知の問題

前提条件

AWS CloudHSM キーストアを使用するには、まず AWS CloudHSM JCE SDK を初期化して設定する必要があります。

手順 1: JCE をインストールする

AWS CloudHSM クライアントの前提条件を含む JCE をインストールするには、Java ライブラリ をインストールするステップに従います。

手順 2: 環境変数に HSM ログイン認証情報を追加する

HSM ログイン認証情報を格納する環境変数を設定します。

Linux

```
$ export HSM_USER=<HSM user name>
```

```
$ export HSM_PASSWORD=<HSM password>
```

Windows

```
PS C:\> $Env:HSM_USER=<HSM user name>
```

```
PS C:\> $Env:HSM_PASSWORD=<HSM password>
```

> **Note**
>
> AWS CloudHSM JCE にはさまざまなログインオプションがあります。サードパーティーアプリケーションで AWS CloudHSM キーストアを使用するには、環境変数で暗黙的なログインを使用する必要があります。アプリケーションコードによる明示的なログインを使用する場合は、 AWS CloudHSM キーストアを使用して独自のアプリケーションを構築する必要があります。詳細については、「AWS CloudHSM キーストアの使用」の記事を参照してください。

手順 3: JCE プロバイダーを登録する

JCE プロバイダーを Java CloudProvider 設定に登録するには、次の手順に従います。

1. Java インストールで `java.security` 設定ファイルを開き、編集します。
2. `java.security` 設定ファイル
   で、`com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider` を最後のプロバイダーとして追加します。例えば、`java.security` ファイルに 9 つのプロバイダーがある場合は、セクションの最後のプロバイダーとして次のプロバイダーを追加します。

   ```
   security.provider.10=com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider
   ```

> **Note**
>
> AWS CloudHSM プロバイダーをより高い優先度として追加すると、ソフトウェアに安全にオフロードされる可能性のあるオペレーションに AWS CloudHSM プロバイダーが優先されるため、システムのパフォーマンスに悪影響を及ぼす可能性があります。ベストプラクティスとして、 AWS CloudHSM またはソフトウェアベースのプロバイダーのいずれであっても、オペレーションに使用するプロバイダーを常に指定します。

> **Note**
>
> AWS CloudHSM キーストアで keytool を使用してキーを生成するときに `-providerName`、`-providerclass`、`-providerpath` コマンドラインオプションを指定すると、エラーが発生する可能性があります。

AWS CloudHSM keytool でのキーストアの使用

keytool は、一般的なキーおよび証明書タスク向けの一般的なコマンドラインユーティリティです。keytool に関する完全なチュートリアルは、 AWS CloudHSM ドキュメントの範囲外です。この記事では、 をキーストアを介した信頼のルート AWS CloudHSM として使用する場合に、さまざまな AWS CloudHSM keytool 関数で使用する特定のパラメータについて説明します。

keytool をキー AWS CloudHSM ストアで使用する場合は、任意の keytool コマンドに次の引数を指定します。

Linux

```
-storetype CLOUDHSM -J-classpath< '-J/opt/cloudhsm/java/*'>
```

Windows

```
-storetype CLOUDHSM -J-classpath<'-J"C:\Program Files\Amazon\CloudHSM\java\*"'>
```

キーストアを使用して新しい AWS CloudHSM キーストアファイルを作成する場合は、「」を参照してくださいの使用 AWS CloudHSM KeyStore。既存のキーストアを使用するには、keytool の –keystore 引数を使用して、その名前 (パスを含む) を指定します。keytool コマンドで存在しないキー

ストアファイルを指定すると、 AWS CloudHSM キーストアは新しいキーストアファイルを作成します。

keytool で新しいキーを作成する

keytool を使用して、 AWS CloudHSMの JCE SDK でサポートされている RSA、AES、DESede タイプのキーを生成できます。

> **Important**
>
> keytool で生成されたキーはソフトウェアで生成され、抽出可能な永続キー AWS CloudHSM として にインポートされます。

エクスポートできないキーを keytool の外部で生成したうえで、対応する証明書をキーストアにインポートすることを強くお勧めします。keytool と Jarsigner を介して抽出可能な RSA キーまたは EC キーを使用する場合、プロバイダーは からキーをエクスポート AWS CloudHSM し、署名オペレーションにローカルでキーを使用します。

AWS CloudHSM クラスターに複数のクライアントインスタンスが接続されている場合は、1 つのクライアントインスタンスのキーストアに証明書をインポートしても、他のクライアントインスタンスで証明書が自動的に使用可能にならないことに注意してください。各クライアントインスタンスでキーおよび関連する証明書を登録するには、「the section called “keytool を使用して CSR を生成する”」の説明に従って Java アプリケーションを実行する必要があります。または、1 つのクライアントで必要な変更を行い、結果のキーストアファイルを他のすべてのクライアントインスタンスにコピーすることもできます。

例 1: 対称型 AES-256 キーを生成し、作業ディレクトリの「my_keystore.store」という名前のキーストアファイルに保存します。*<secret label>* を独自のラベルに置き換えます。

Linux

```
$ keytool -genseckey -alias <secret label> -keyalg aes \
 -keysize 256 -keystore my_keystore.store \
 -storetype CloudHSM -J-classpath '-J/opt/cloudhsm/java/*' \
```

Windows

```
PS C:\> keytool -genseckey -alias <secret label> -keyalg aes `
 -keysize 256 -keystore my_keystore.store `
```

```
 -storetype CloudHSM -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

例 2: RSA 2048 キーペアを生成し、作業ディレクトリの「my_keystore.store」という名前のキーストアファイルに保存します。*<RSA key pair label>* を独自のラベルに置き換えます。

Linux

```
$ keytool -genkeypair -alias <RSA key pair label> \
 -keyalg rsa -keysize 2048 \
 -sigalg sha512withrsa \
 -keystore my_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -genkeypair -alias <RSA key pair label> `
 -keyalg rsa -keysize 2048 `
 -sigalg sha512withrsa `
 -keystore my_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

サポートされている署名アルゴリズムのリストは、Java ライブラリにあります。

keytool を使用してキーを削除する

AWS CloudHSM キーストアはキーの削除をサポートしていません。Destroyable インターフェイスの destroy メソッドを使用してキーを削除できます。

```
((Destroyable) key).destroy();
```

keytool を使用して CSR を生成する

OpenSSL Dynamic Engine を使用すると、証明書署名要求 (CSR) を柔軟に生成できます。次のコマンドは、keytool を使用して、エイリアス `my-key-pair` を持つキーペアの CSR を生成します。

Linux

```
$ keytool -certreq -alias <key pair label> \
```

```
 -file my_csr.csr \
 -keystore my_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -certreq -alias <key pair label> `
 -file my_csr.csr `
 -keystore my_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

> **Note**
>
> keytool のキーペアを使用するには、指定されたキーストアファイルにそのキーペアのエントリが必要です。keytool の外部で生成されたキーペアを使用する場合は、キーおよび証明書のメタデータをキーストアにインポートする必要があります。キーストアデータをインポートする手順については、「the section called “keytool を使用して中間証明書とルート証明書を AWS CloudHSM キーストアにインポートする ”」を参照してください。

keytool を使用して中間証明書とルート証明書を AWS CloudHSM キーストアにインポートする

CA 証明書をインポートするには、新しくインポートした証明書で完全な証明書チェーンの検証を有効にする必要があります。次のコマンドでは、例を示しています。

Linux

```
$ keytool -import -trustcacerts -alias rootCAcert \
 -file rootCAcert.cert -keystore my_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -import -trustcacerts -alias rootCAcert `
 -file rootCAcert.cert -keystore my_keystore.store `
 -storetype CLOUDHSM `
```

```
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

複数のクライアントインスタンスを AWS CloudHSM クラスターに接続する場合、1 つのクライアントインスタンスのキーストアに証明書をインポートしても、他のクライアントインスタンスで証明書が自動的に使用できるようになります。各クライアントインスタンスで証明書をインポートする必要があります。

keytool を使用して AWS CloudHSM キーストアから証明書を削除する

次のコマンドは、Java keytool のキーストアから証明書を削除する方法の例を示しています。

Linux

```
$ keytool -delete -alias mydomain \
 -keystore my_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -delete -alias mydomain `
 -keystore my_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

複数のクライアントインスタンスを AWS CloudHSM クラスターに接続する場合、1 つのクライアントインスタンスのキーストアで証明書を削除しても、他のクライアントインスタンスから証明書は自動的に削除されません。各クライアントインスタンスで証明書を削除する必要があります。

keytool を使用して作業用証明書を AWS CloudHSM キーストアにインポートする

証明書署名要求 (CSR) が署名されると、それを AWS CloudHSM キーストアにインポートし、適切なキーペアに関連付けることができます。次のコマンドでは、例を示しています。

Linux

```
$ keytool -importcert -noprompt -alias <key pair label> \
 -file my_certificate.crt \
 -keystore my_keystore.store \
```

```
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -importcert -noprompt -alias <key pair label> `
 -file my_certificate.crt `
 -keystore my_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

エイリアスは、キーストア内の関連付けられた証明書を持つキーペアである必要があります。キーが keytool の外部で生成される場合や、別のクライアントインスタンスで生成される場合は、まずキーおよび証明書のメタデータをキーストアにインポートする必要があります。

証明書チェーンは検証可能である必要があります。証明書を検証できない場合は、チェーンを検証できるように、署名 (証明機関) 証明書をキーストアにインポートする必要があります。

keytool を使用した証明書のエクスポート

次の例では、バイナリ X.509 形式の証明書を生成します。人間が読める証明書を エクスポートするには、`-exportcert` コマンドに `-rfc` を追加します。

Linux

```
$ keytool -exportcert -alias <key pair label> \
 -file my_exported_certificate.crt \
 -keystore my_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -exportcert -alias <key pair label> `
 -file my_exported_certificate.crt `
 -keystore my_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

Jarsigner での AWS CloudHSM キーストアの使用

Jarsigner は、HSM に安全に保存されているキーを使用して JAR ファイルに署名するための一般的なコマンドラインユーティリティです。Jarsigner に関する完全なチュートリアルは、 AWS CloudHSM ドキュメントの範囲外です。このセクションでは、 AWS CloudHSM キーストアを通じて信頼のルート AWS CloudHSM として を使用して署名に署名および検証するために使用する Jarsigner パラメータについて説明します。

キーと証明書のセットアップ

Jarsigner を使用して JAR ファイルに署名する前に、次の手順を設定または完了していることを確認してください。

1. 「AWS CloudHSM キーストアの前提条件」のガイダンスに従います。
2. 現在のサーバーまたはクライアントインスタンスのキーストアに保存する必要がある署名 AWS CloudHSM キー、関連する証明書、証明書チェーンを設定します。でキーを作成し AWS CloudHSM 、関連するメタデータを AWS CloudHSM キーストアにインポートします。keytool を使用してキーおよび証明書を設定する場合は、「the section called “keytool で新しいキーを作成する”」を参照してください。複数のクライアントインスタンスを使用して JAR に署名する場合は、キーを作成し、証明書チェーンをインポートします。次に、結果のキーストアファイルを各クライアントインスタンスにコピーします。新しいキーを頻繁に生成する場合は、各クライアントインスタンスに証明書を個別にインポートする方が簡単です。
3. 証明書チェーン全体が検証可能である必要があります。証明書チェーンを検証できるようにするには、CA 証明書と中間証明書を AWS CloudHSM キーストアに追加する必要がある場合があります。Java コードを使用して証明書チェーンを検証する方法については、「the section called “AWS CloudHSM とJarsignerを使用してJARファイルに署名する”」のコードスニペットを参照してください。必要に応じて、keytool を使用して証明書をインポートできます。keytool を使用する手順については、「the section called “keytool を使用して中間証明書とルート証明書を AWS CloudHSM キーストアにインポートする ”」を参照してください。

AWS CloudHSM とJarsignerを使用してJARファイルに署名する

JAR ファイルに署名するには、次のコマンドを使用します。

Linux;

OpenJDK 8 の場合

```
jarsigner -keystore my_keystore.store \
```

```
 -signedjar signthisclass_signed.jar \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass.jar <key pair label>
```

OpenJDK 11、OpenJDK 17、および OpenJDK 21 の場合

```
jarsigner -keystore my_keystore.store \
 -signedjar signthisclass_signed.jar \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass.jar <key pair label>
```

Windows

OpenJDK8 向け

```
jarsigner -keystore my_keystore.store `
 -signedjar signthisclass_signed.jar `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java
\jdk1.8.0_331\lib\tools.jar' `
  "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass.jar <key pair label>
```

OpenJDK 11、OpenJDK 17、および OpenJDK 21 の場合

```
jarsigner -keystore my_keystore.store `
 -signedjar signthisclass_signed.jar `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*'`
  "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass.jar <key pair label>
```

署名付き JAR を確認するには、次のコマンドを使用します。

Linux

OpenJDK8 向け

```
jarsigner -verify \
 -keystore my_keystore.store \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass_signed.jar <key pair label>
```

OpenJDK 11、OpenJDK 17、および OpenJDK 21 の場合

```
jarsigner -verify \
 -keystore my_keystore.store \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass_signed.jar <key pair label>
```

Windows

OpenJDK 8 の場合

```
jarsigner -verify `
 -keystore my_keystore.store `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java
\jdk1.8.0_331\lib\tools.jar' `
 "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass_signed.jar <key pair label>
```

OpenJDK 11、OpenJDK 17、および OpenJDK 21 の場合

```
jarsigner -verify `
 -keystore my_keystore.store `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*`
 "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass_signed.jar <key pair label>
```

既知の問題

1. Keytool と Jarsigner では EC キーはサポートされていません。

Client SDK 3 を使用して Java Keytool および Jarsigner と統合する

AWS CloudHSM key store は、HSM のキーに関連付けられた証明書を、 `keytool`や などのサードパーティーツールを使用して使用する専用 JCE キーストアです`jarsigner`。証明書はパブリックで非機密データであるため、 AWS CloudHSM HSM に証明書を保存しません。 AWS CloudHSM キーストアは証明書をローカルファイルに格納し、証明書を HSM の対応するキーにマッピングします。

AWS CloudHSM キーストアを使用して新しいキーを生成すると、ローカルキーストアファイルにはエントリが生成されず、キーは HSM に作成されます。同様に、 AWS CloudHSM キーストアを使用してキーを検索すると、検索が HSM に渡されます。証明書を AWS CloudHSM キーストアに保存すると、プロバイダーは対応するエイリアスを持つキーペアが HSM に存在することを確認し、提供された証明書を対応するキーペアに関連付けます。

トピック

前提条件

AWS CloudHSM キーストアを使用するには、まず AWS CloudHSM JCE SDK を初期化して設定する必要があります。

手順 1: JCE をインストールする

AWS CloudHSM クライアントの前提条件を含む JCE をインストールするには、Java ライブラリ をインストールするステップに従います。

手順 2: 環境変数に HSM ログイン認証情報を追加する

HSM ログイン認証情報を格納する環境変数を設定します。

```
export HSM_PARTITION=PARTITION_1
export HSM_USER=<HSM user name>
export HSM_PASSWORD=<HSM password>
```

> **Note**
>
> CloudHSM JCE には、さまざまなログインオプションがあります。サードパーティーアプリケーションで AWS CloudHSM キーストアを使用するには、環境変数で暗黙的なログインを使用する必要があります。アプリケーションコードによる明示的なログインを使用する場合は、 AWS CloudHSM キーストアを使用して独自のアプリケーションを構築する必要があります。詳細については、「 AWS CloudHSM キーストアの使用」の記事を参照してください。

手順 3: JCE プロバイダーを登録する

JCE プロバイダーを登録するには、Java CloudProvider 設定で行います。

1. Java インストールで java.security 設定ファイルを開き、編集します。
2. java.security 設定ファイルで、`com.cavium.provider.CaviumProvider` を最後のプロバイダーとして追加します。たとえば、java.security ファイルに 9 つのプロバイダーがある場合は、セクションの最後のプロバイダーとして次のプロバイダーを追加します。Cavium プロバイダーの優先順位を高く設定すると、システムのパフォーマンスに悪影響を与える可能性があります。

   ```
   security.provider.10=com.cavium.provider.CaviumProvider
   ```

> **Note**
>
> パワーユーザーは、keytool を使用する際、セキュリティ構成ファイルを更新する代わりに `-providerName`、`-providerclass` および `-providerpath` のコマンドラインオプションを指定することに慣れているかもしれませんが、 AWS CloudHSM キーストアでキーを生成するときにコマンドラインオプションを指定しようとすると、エラーが発生します。

AWS CloudHSM keytool でのキーストアの使用

keytool は、Linux システム上の一般的なキーおよび証明書タスク向けの一般的なコマンドラインユーティリティです。keytool に関する完全なチュートリアルは、 AWS CloudHSM ドキュメントの範囲外です。この記事では、 をキーストアを介した信頼のルート AWS CloudHSM として使用する場合に、さまざまな AWS CloudHSM keytool 関数で使用する特定のパラメータについて説明します。

keytool をキー AWS CloudHSM ストアで使用する場合は、任意の keytool コマンドに次の引数を指定します。

```
-storetype CLOUDHSM \
  -J-classpath '-J/opt/cloudhsm/java/*' \
  -J-Djava.library.path=/opt/cloudhsm/lib
```

キーストアを使用して新しい AWS CloudHSM キーストアファイルを作成する場合は、「」を参照してくださいの使用 AWS CloudHSM KeyStore。既存のキーストアを使用するには、keytool の –keystore 引数を使用して、その名前 (パスを含む) を指定します。keytool コマンドで存在しないキーストアファイルを指定すると、 AWS CloudHSM キーストアは新しいキーストアファイルを作成します。

keytool で新しいキーを作成する

keytool を使用して、 AWS CloudHSMの JCE SDK でサポートされている任意のタイプのキーを生成できます。キーと長さの完全なリストについては、Java ライブラリの「サポートされるキー」の記事を参照してください。

> **Important**
>
> keytool で生成されたキーはソフトウェアで生成され、抽出可能な永続キー AWS CloudHSM として にインポートされます。

抽出不可能なキーを HSM で直接作成し、それらを keytool または Jarsigner で使用する手順については、「既存のキーを AWS CloudHSM Key Store に登録する」のコードサンプルを参照してください。エクスポートできないキーを keytool の外部で生成したうえで、対応する証明書をキーストアにインポートすることを強くお勧めします。keytool と jarsigner を介して抽出可能な RSA キーまたは EC キーを使用する場合、プロバイダーは からキーをエクスポート AWS CloudHSM し、署名オペレーションにローカルでキーを使用します。

CloudHSM クラスターに複数のクライアントインスタンスが接続されている場合、1 つのクライアントインスタンスのキーストアに証明書をインポートしても、他のクライアントインスタンスで自動的に使用できるようにはなりません。各クライアントインスタンスでキーおよび関連する証明書を登録するには、「keytool を使用して CSR を生成する」の説明に従って Java アプリケーションを実行する必要があります。または、1 つのクライアントで必要な変更を行い、結果のキーストアファイルを他のすべてのクライアントインスタンスにコピーすることもできます。

例 1: 対称型 AES-256 キーを生成し、作業ディレクトリの「my_keystore.store」という名前のキーストアファイルに保存します。*<secret label>* を独自のラベルに置き換えます。

```
keytool -genseckey -alias <secret label> -keyalg aes \
  -keysize 256 -keystore my_keystore.store \
  -storetype CloudHSM -J-classpath '-J/opt/cloudhsm/java/*' \
  -J-Djava.library.path=/opt/cloudhsm/lib/
```

例 2: RSA 2048 キーペアを生成し、作業ディレクトリの「my_keystore.store」という名前のキーストアファイルに保存します。*<RSA key pair label>* を独自のラベルに置き換えます。

```
keytool -genkeypair -alias <RSA key pair label> \
        -keyalg rsa -keysize 2048 \
        -sigalg sha512withrsa \
        -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

例 3: p256 ED キーを生成し、作業ディレクトリの「my_keystore.store」という名前のキーストアファイルに保存します。*<ec key pair label>* を独自のラベルに置き換えます。

```
keytool -genkeypair -alias <ec key pair label> \
        -keyalg ec -keysize 256 \
        -sigalg SHA512withECDSA \
        -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

サポートされている署名アルゴリズムのリストは、Java ライブラリにあります。

keytool を使用してキーを削除する

AWS CloudHSM キーストアはキーの削除をサポートしていません。キーを削除するには、 のコマンドラインツール AWS CloudHSMの `deleteKey`関数を使用する必要がありますdeleteKey。

keytool を使用して CSR を生成する

OpenSSL Dynamic Engine を使用すると、証明書署名要求 (CSR) を柔軟に生成できます。次のコマンドは、keytool を使用して、エイリアス `my-key-pair` を持つキーペアの CSR を生成します。

```
keytool -certreq -alias <key pair label> \
        -file my_csr.csr \
        -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

> **Note**
>
> keytool のキーペアを使用するには、指定されたキーストアファイルにそのキーペアのエントリが必要です。keytool の外部で生成されたキーペアを使用する場合は、キーおよび証明書のメタデータをキーストアにインポートする必要があります。キーストアデータのインポート手順については、AWS CloudHSM 「Keytool を使用して中間証明書とルート証明書をキーストアにインポートする」を参照してください。

keytool を使用して中間証明書とルート証明書を AWS CloudHSM キーストアにインポートする

CA 証明書をインポートするには、新しくインポートした証明書で完全な証明書チェーンの検証を有効にする必要があります。次のコマンドでは、例を示しています。

```
keytool -import -trustcacerts -alias rootCAcert \
        -file rootCAcert.cert -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

複数のクライアントインスタンスを AWS CloudHSM クラスターに接続する場合、1 つのクライアントインスタンスのキーストアに証明書をインポートしても、他のクライアントインスタンスで証明書が自動的に使用できるようになります。各クライアントインスタンスで証明書をインポートする必要があります。

keytool を使用して AWS CloudHSM キーストアから証明書を削除する

次のコマンドは、Java keytool のキーストアから証明書を削除する方法の例を示しています。

```
keytool -delete -alias mydomain -keystore \
        -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

複数のクライアントインスタンスを AWS CloudHSM クラスターに接続する場合、1 つのクライアントインスタンスのキーストアで証明書を削除しても、他のクライアントインスタンスから証明書は自動的に削除されません。各クライアントインスタンスで証明書を削除する必要があります。

keytool を使用して作業用証明書を AWS CloudHSM キーストアにインポートする

証明書署名要求 (CSR) が署名されると、それを AWS CloudHSM キーストアにインポートし、適切なキーペアに関連付けることができます。次のコマンドでは、例を示しています。

```
keytool -importcert -noprompt -alias <key pair label> \
        -file my_certificate.crt \
        -keystore my_keystore.store
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

エイリアスは、キーストア内の関連付けられた証明書を持つキーペアである必要があります。キーが keytool の外部で生成される場合や、別のクライアントインスタンスで生成される場合は、まずキーおよび証明書のメタデータをキーストアにインポートする必要があります。証明書メタデータをインポートする手順については、AWS CloudHSM 「 キーストアで既存のキーを登録する」のコードサンプルを参照してください。

証明書チェーンは検証可能である必要があります。証明書を検証できない場合は、チェーンを検証できるように、署名 (証明機関) 証明書をキーストアにインポートする必要があります。

keytool を使用した証明書のエクスポート

次の例では、バイナリ X.509 形式の証明書を生成します。人間が読める証明書を工クスポートするには、`-exportcert` コマンドに `-rfc` を追加します。

```
keytool -exportcert -alias <key pair label> \
        -file my_exported_certificate.crt \
        -keystore my_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

jarsigner での AWS CloudHSM キーストアの使用

Jarsigner は、HSM に安全に保存されているキーを使用して JAR ファイルに署名するための一般的なコマンドラインユーティリティです。Jarsigner に関する完全なチュートリアルは、 AWS CloudHSM ドキュメントの範囲外です。このセクションでは、 AWS CloudHSM キーストアを通じて信頼のルート AWS CloudHSM として を使用して署名に署名および検証するために使用する Jarsigner パラメータについて説明します。

キーと証明書のセットアップ

Jarsigner を使用して JAR ファイルに署名する前に、次の手順を設定または完了していることを確認してください。

1. 「AWS CloudHSM キーストアの前提条件」のガイダンスに従います。
2. 現在のサーバーまたはクライアントインスタンスのキーストアに保存する必要がある署名 AWS CloudHSM キー、関連する証明書、証明書チェーンを設定します。でキーを作成し AWS CloudHSM 、関連するメタデータを AWS CloudHSM キーストアにインポートします。「キーストアで既存のキーを登録する」のコードサンプルを使用して、 AWS CloudHSMメタデータをキーストアにインポートします。keytool を使用してキーおよび証明書を設定する場合は、「keytool

で新しいキーを作成する」を参照してください。複数のクライアントインスタンスを使用して JAR に署名する場合は、キーを作成し、証明書チェーンをインポートします。次に、結果のキーストアファイルを各クライアントインスタンスにコピーします。新しいキーを頻繁に生成する場合は、各クライアントインスタンスに証明書を個別にインポートする方が簡単です。

3. 証明書チェーン全体が検証可能である必要があります。証明書チェーンを検証できるようにするには、CA 証明書と中間証明書を AWS CloudHSM キーストアに追加する必要がある場合があります。Java コードを使用して証明書チェーンを検証する手順については、「 を使用して JAR ファイルに署名する」のコードスニペット AWS CloudHSM と「Jarsigner」を参照してください。必要に応じて、keytool を使用して証明書をインポートできます。keytool を使用する手順については、「Keytool を使用して中間証明書とルート証明書を AWS CloudHSM Key Store にインポートする」を参照してください。

AWS CloudHSM と Jarsigner を使用して JAR ファイルに署名する

JAR ファイルに署名するには、次のコマンドを使用します。

```
jarsigner -keystore my_keystore.store \
        -signedjar signthisclass_signed.jar \
        -sigalg sha512withrsa \
        -storetype CloudHSM \
        -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
        -J-Djava.library.path=/opt/cloudhsm/lib \
        signthisclass.jar <key pair label>
```

署名付き JAR を確認するには、次のコマンドを使用します。

```
jarsigner -verify \
        -keystore my_keystore.store \
        -sigalg sha512withrsa \
        -storetype CloudHSM \
        -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
        -J-Djava.library.path=/opt/cloudhsm/lib \
        signthisclass_signed.jar <key pair label>
```

既知の問題

以下は、既知の問題の一覧です。

- keytool を使用してキーを生成する場合、プロバイダー設定の最初のプロバイダーを にすることはできません CaviumProvider。

- keytool を使用してキーを生成する場合、セキュリティ構成ファイル内の最初の (サポートされている) プロバイダーを使用してキーを生成します。これは通常、ソフトウェアプロバイダーです。その後、生成されたキーにはエイリアスが与えられ、キー追加プロセス中に永続的 (トークン) キーとして AWS CloudHSM HSM にインポートされます。
- キーストアで AWS CloudHSM keytool を使用する場合は、コマンドラインで `-providerName`、`-providerclass`、または `-providerpath`オプションを指定しないでください。これらのオプションは、「キーストアの前提条件」の説明に従って、セキュリティプロバイダーファイルで指定します。
- keytool と Jarsigner を介して抽出不可能な EC キーを使用する場合、SunEC プロバイダーを java.security ファイル内のプロバイダーのリストから削除する、または無効にする必要があります。keytool と Jarsigner を介して抽出可能な EC キーを使用する場合、プロバイダーは HSM AWS CloudHSM からキービットをエクスポートし、署名オペレーションにローカルでキーを使用します。keytool または Jarsigner でエクスポート可能なキーを使用することはお勧めしません。

既存のキーを AWS CloudHSM キーストアに登録する

属性とラベル付けのセキュリティと柔軟性を最大限に高めるため、key_mgmt_util を使用して署名キーを生成することをお勧めします。Java アプリケーションを使用して、 AWS CloudHSMでキーを生成することもできます。

次のセクションでは、HSM で新しいキーペアを生成し、 AWS CloudHSM キーストアにインポートされた既存のキーを使用して登録する方法を示すコードサンプルを示します。インポートされたキーは、keytool や Jarsigner などのサードパーティー製ツールで使用できます。

既存のキーを使用するには、新しいキーを生成するのではなく、ラベルでキーを検索するようにコードサンプルを変更します。ラベルでキーを検索するためのサンプルコードは、 の KeyUtilitiesRunner.java サンプルにあります GitHub。

> **Important**
>
> に保存されたキー AWS CloudHSM をローカルキーストアに登録しても、キーはエクスポートされません。キーが登録されると、キーストアはキーのエイリアス (またはラベル) を登録し、 AWS CloudHSMでローカルに保存された証明書オブジェクトとのキーペアを関連付けます。キーペアがエクスポート不可として作成されている限り、キービットが HSM から離れることはありません。

```
                        //
 // Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 //
 // Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 // software and associated documentation files (the "Software"), to deal in the
 Software
 // without restriction, including without limitation the rights to use, copy, modify,
 // merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
 // permit persons to whom the Software is furnished to do so.
 //
 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 // INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 // PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 // OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 // SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 //

package com.amazonaws.cloudhsm.examples;

import com.cavium.key.CaviumKey;
import com.cavium.key.parameter.CaviumAESKeyGenParameterSpec;
import com.cavium.key.parameter.CaviumRSAKeyGenParameterSpec;
import com.cavium.asn1.Encoder;
import com.cavium.cfm2.Util;

import javax.crypto.KeyGenerator;

import java.io.ByteArrayInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;

import java.math.BigInteger;

import java.security.*;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
```

```
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.KeyStore.PasswordProtection;
import java.security.KeyStore.PrivateKeyEntry;
import java.security.KeyStore.Entry;

import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;

//
// KeyStoreExampleRunner demonstrates how to load a keystore, and associate a
 certificate with a
// key in that keystore.
//
// This example relies on implicit credentials, so you must setup your environment
 correctly.
//
// https://docs.aws.amazon.com/cloudhsm/latest/userguide/java-library-
install.html#java-library-credentials
//

public class KeyStoreExampleRunner {

    private static byte[] COMMON_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x03 };
    private static byte[] COUNTRY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x06 };
    private static byte[] LOCALITY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x07 };
    private static byte[] STATE_OR_PROVINCE_NAME_OID = new byte[] { (byte) 0x55,
 (byte) 0x04, (byte) 0x08 };
    private static byte[] ORGANIZATION_NAME_OID = new byte[] { (byte) 0x55, (byte)
 0x04, (byte) 0x0A };
    private static byte[] ORGANIZATION_UNIT_OID = new byte[] { (byte) 0x55, (byte)
 0x04, (byte) 0x0B };

    private static String helpString = "KeyStoreExampleRunner%n" +
           "This sample demonstrates how to load and store keys using a keystore.%n%n"
 +
           "Options%n" +
           "\t--help\t\t\tDisplay this message.%n" +
           "\t--store <filename>\t\tPath of the keystore.%n" +
```

```
            "\t--password <password>\t\tPassword for the keystore (not your CU
password).%n" +
            "\t--label <label>\t\t\tLabel to store the key and certificate under.%n" +
            "\t--list\t\t\tList all the keys in the keystore.%n%n";

    public static void main(String[] args) throws Exception {
        Security.addProvider(new com.cavium.provider.CaviumProvider());
        KeyStore keyStore = KeyStore.getInstance("CloudHSM");

        String keystoreFile = null;
        String password = null;
        String label = null;
        boolean list = false;
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
            switch (args[i]) {
                case "--store":
                    keystoreFile = args[++i];
                    break;
                case "--password":
                    password = args[++i];
                    break;
                case "--label":
                    label = args[++i];
                    break;
                case "--list":
                    list = true;
                    break;
                case "--help":
                    help();
                    return;
            }
        }

        if (null == keystoreFile || null == password) {
            help();
            return;
        }

        if (list) {
            listKeys(keystoreFile, password);
            return;
        }
```

```
        if (null == label) {
            label = "Keystore Example Keypair";
        }

        //
        // This call to keyStore.load() will open the pkcs12 keystore with the supplied
        // password and connect to the HSM. The CU credentials must be specified using
        // standard CloudHSM login methods.
        //
        try {
            FileInputStream instream = new FileInputStream(keystoreFile);
            keyStore.load(instream, password.toCharArray());
        } catch (FileNotFoundException ex) {
            System.err.println("Keystore not found, loading an empty store");
            keyStore.load(null, null);
        }

        PasswordProtection passwd = new PasswordProtection(password.toCharArray());
        System.out.println("Searching for example key and certificate...");

        PrivateKeyEntry keyEntry = (PrivateKeyEntry) keyStore.getEntry(label, passwd);
        if (null == keyEntry) {
            //
            // No entry was found, so we need to create a key pair and associate a
certificate.
            // The private key will get the label passed on the command line. The
keystore alias
            // needs to be the same as the private key label. The public key will have
":public"
            // appended to it. The alias used in the keystore will We associate the
certificate
            // with the private key.
            //
            System.out.println("No entry found, creating...");
            KeyPair kp = generateRSAKeyPair(2048, label + ":public", label);
            System.out.printf("Created a key pair with the handles %d/%d%n",
((CaviumKey) kp.getPrivate()).getHandle(), ((CaviumKey) kp.getPublic()).getHandle());

            //
            // Generate a certificate and associate the chain with the private key.
            //
            Certificate self_signed_cert = generateCert(kp);
            Certificate[] chain = new Certificate[1];
            chain[0] = self_signed_cert;
```

```
            PrivateKeyEntry entry = new PrivateKeyEntry(kp.getPrivate(), chain);

            //
            // Set the entry using the label as the alias and save the store.
            // The alias must match the private key label.
            //
            keyStore.setEntry(label, entry, passwd);

            FileOutputStream outstream = new FileOutputStream(keystoreFile);
            keyStore.store(outstream, password.toCharArray());
            outstream.close();

            keyEntry = (PrivateKeyEntry) keyStore.getEntry(label, passwd);
        }

        long handle = ((CaviumKey) keyEntry.getPrivateKey()).getHandle();
        String name = keyEntry.getCertificate().toString();
        System.out.printf("Found private key %d with certificate %s%n", handle, name);
    }

    private static void help() {
        System.out.println(helpString);
    }

    //
    // Generate a non-extractable / non-persistent RSA keypair.
    // This method allows us to specify the public and private labels, which
    // will make KeyStore alises easier to understand.
    //
    public static KeyPair generateRSAKeyPair(int keySizeInBits, String publicLabel,
String privateLabel)
            throws InvalidAlgorithmParameterException, NoSuchAlgorithmException,
NoSuchProviderException {

        boolean isExtractable = false;
        boolean isPersistent = false;
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("rsa", "Cavium");
        CaviumRSAKeyGenParameterSpec spec = new
CaviumRSAKeyGenParameterSpec(keySizeInBits, new BigInteger("65537"), publicLabel,
privateLabel, isExtractable, isPersistent);

        keyPairGen.initialize(spec);

        return keyPairGen.generateKeyPair();
```

```
    }

    //
    // Generate a certificate signed by a given keypair.
    //
    private static Certificate generateCert(KeyPair kp) throws CertificateException {
        CertificateFactory cf = CertificateFactory.getInstance("X509");
        PublicKey publicKey = kp.getPublic();
        PrivateKey privateKey = kp.getPrivate();
        byte[] version = Encoder.encodeConstructed((byte) 0,
Encoder.encodePositiveBigInteger(new BigInteger("2"))); // version 1
        byte[] serialNo = Encoder.encodePositiveBigInteger(new BigInteger(1,
Util.computeKCV(publicKey.getEncoded())));

        // Use the SHA512 OID and algorithm.
        byte[] signatureOid = new byte[] {
            (byte) 0x2A, (byte) 0x86, (byte) 0x48, (byte) 0x86, (byte) 0xF7, (byte)
0x0D, (byte) 0x01, (byte) 0x01, (byte) 0x0D };
        String sigAlgoName = "SHA512WithRSA";

         byte[] signatureId = Encoder.encodeSequence(
                                        Encoder.encodeOid(signatureOid),
                                        Encoder.encodeNull());

         byte[] issuer = Encoder.encodeSequence(
                                    encodeName(COUNTRY_NAME_OID, "<Country>"),
                                    encodeName(STATE_OR_PROVINCE_NAME_OID, "<State>"),
                                    encodeName(LOCALITY_NAME_OID, "<City>"),
                                    encodeName(ORGANIZATION_NAME_OID,
"<Organization>"),
                                    encodeName(ORGANIZATION_UNIT_OID, "<Unit>"),
                                    encodeName(COMMON_NAME_OID, "<CN>")
                                );

         Calendar c = Calendar.getInstance();
         c.add(Calendar.DAY_OF_YEAR, -1);
         Date notBefore = c.getTime();
         c.add(Calendar.YEAR, 1);
         Date notAfter = c.getTime();
         byte[] validity = Encoder.encodeSequence(
                                        Encoder.encodeUTCTime(notBefore),
                                        Encoder.encodeUTCTime(notAfter)
                                    );
         byte[] key = publicKey.getEncoded();
```

```
        byte[] certificate = Encoder.encodeSequence(
                                        version,
                                        serialNo,
                                        signatureId,
                                        issuer,
                                        validity,
                                        issuer,
                                        key);
        Signature sig;
        byte[] signature = null;
        try {
            sig = Signature.getInstance(sigAlgoName, "Cavium");
            sig.initSign(privateKey);
            sig.update(certificate);
            signature = Encoder.encodeBitstring(sig.sign());

        } catch (Exception e) {
            System.err.println(e.getMessage());
            return null;
        }

        byte [] x509 = Encoder.encodeSequence(
                        certificate,
                        signatureId,
                        signature
                        );
        return cf.generateCertificate(new ByteArrayInputStream(x509));
    }

     //
     // Simple OID encoder.
     // Encode a value with OID in ASN.1 format
     //
     private static byte[] encodeName(byte[] nameOid, String value) {
        byte[] name = null;
        name = Encoder.encodeSet(
                    Encoder.encodeSequence(
                            Encoder.encodeOid(nameOid),
                            Encoder.encodePrintableString(value)
                    )
                );
        return name;
     }
```

```
    //
    // List all the keys in the keystore.
    //
    private static void listKeys(String keystoreFile, String password) throws Exception
 {
        KeyStore keyStore = KeyStore.getInstance("CloudHSM");

        try {
            FileInputStream instream = new FileInputStream(keystoreFile);
            keyStore.load(instream, password.toCharArray());
        } catch (FileNotFoundException ex) {
            System.err.println("Keystore not found, loading an empty store");
            keyStore.load(null, null);
        }

        for(Enumeration<String> entry = keyStore.aliases(); entry.hasMoreElements();) {
            System.out.println(entry.nextElement());
        }
    }

}
```

その他のサードパーティベンダーとの統合

一部のサードパーティーベンダーは、信頼のルート AWS CloudHSM として をサポートしています。これはつまり、CloudHSM クラスターで基になるキーを作成、保存しながら、選択したソフトウェアソリューションを利用できるということです。その結果、 のワークロードは、CloudHSM のレイテンシー、可用性、信頼性、伸縮性のメリットに依存する AWS 可能性があります。次の表は、CloudHSM をサポートするサードパーティベンダーのリストです。

> **Note**
>
> AWS は、サードパーティーベンダーを支持または表明しません。

- Hashicorp Vault は、組織間のコラボレーションおよびガバナンスを可能にするために設計されたシークレット管理ツールです。保護を強化するために、信頼のルート AWS CloudHSM として AWS Key Management Service と をサポートしています。
- Thycotic Secrets Server を使用すると、機密性の高い認証情報を特権アカウント間で管理できます。信頼のルート AWS CloudHSM として をサポートします。
- P6R の KMIP アダプターを使用すると、標準の KMIP インターフェイスを介して AWS CloudHSM インスタンスを利用できます。
- PrimeKey EJBCA は PKI 用の一般的なオープンソースソリューションです。これにより、 でキーペアを安全に作成して保存できます AWS CloudHSM。
- Box KeySafe は、厳格なセキュリティ、プライバシー、および規制コンプライアンス要件を持つ多くの組織に、クラウドコンテンツの暗号化キー管理を提供します。お客様は、 KeySafe でキーを直接保護 AWS Key Management Service することも、 AWS KMS カスタムキーストア AWS CloudHSM 経由で で間接的に保護することもできます。
- Insyde Software は、ファームウェア署名の信頼のルート AWS CloudHSM として をサポートします。
- F5 BIG-IP LTM は、信頼のルート AWS CloudHSM として をサポートします。
- Cloudera Navigator Key HSM を使用すると、CloudHSM クラスターを使用してCloudera Navigator Key Trustee Server のキーを作成および保存できます。
- Venafi 信頼保護プラットフォーム は、AWS CloudHSM キー生成と保護により、TLS、SSH、およびコードサイニングのための包括的なマシン ID 管理を提供します。

モニタリング AWS CloudHSM

Client SDK に組み込まれている ログ記録機能に加えて、 AWS CloudTrail、Amazon CloudWatch Logs、Amazon を使用して CloudWatch をモニタリングすることもできます AWS CloudHSM。

クライアント SDK ログ

クライアント SDK ロギングを使用して、作成したアプリケーションからの診断およびトラブルシューティング情報をモニタリングします。

CloudTrail

CloudTrail を使用して、クラスターの作成と削除、ハードウェアセキュリティモジュール (HSM)、リソースタグなど、 AWS アカウント内のすべての API コールをモニタリングします。

CloudWatch ログ

CloudWatch ログを使用して、HSM ユーザーの作成と削除、ユーザーパスワードの変更、キーの作成と削除などのイベントを含む HSM インスタンスからのログをモニタリングします。

CloudWatch

CloudWatch を使用して、クラスターの状態をリアルタイムでモニタリングします。

トピック

- Client SDK ログの操作
- AWS CloudTrail および の使用 AWS CloudHSM
- Amazon CloudWatch Logs と AWS CloudHSM 監査ログの使用
- の CloudWatch メトリクスの取得 AWS CloudHSM

Client SDK ログの操作

Client SDK によって生成されたログを取得できます。クライアント SDK 3 AWS CloudHSM とクライアント SDK 5 を使用したログ記録の実装を提供します。

トピック

- Client SDK 5 ログ記録
- Client SDK 3 ログ記録

Client SDK 5 ログ記録

クライアント SDK 5 ログには、コンポーネントのために名付けられたファイル中の各コンポーネントのための情報が含まれています。クライアント SDK 5 の設定ツールを使用して、各コンポーネントのログを構成できます。

ファイルの場所を指定しない場合、システムはログをデフォルトの場所に書き込みます。

PKCS #11 library

- Linux

  ```
  /opt/cloudhsm/run/cloudhsm-pkcs11.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-pkcs11.log
  ```

OpenSSL Dynamic Engine

- Linux

  ```
  stderr
  ```

JCE provider

- Linux

  ```
  /opt/cloudhsm/run/cloudhsm-jce.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-jce.log
  ```

Client SDK 5 のログ記録を構成する方法については、「Client SDK 5 Configure tool」を参照してください

Client SDK 3 ログ記録

Client SDK 3 ログには、 AWS CloudHSM クライアントデーモンからの詳細な情報が含まれています。ログの場所は、 クライアントデーモンを実行するクライアントを実行する Amazon EC2 クライアントインスタンスのオペレーティングシステムによって異なります。

Amazon Linux

Amazon Linux では、 AWS CloudHSM クライアントログは という名前のファイルに書き込まれます`/opt/cloudhsm/run/cloudhsm_client.log`。logrotate などを使用して、これらのログをローテーションして管理します。

Amazon Linux 2

Amazon Linux 2 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

CentOS 7

CentOS 7 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

CentOS 8

CentOS 8 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

RHEL 7

Red Hat Enterprise Linux 7 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

RHEL 8

Red Hat Enterprise Linux 8 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

Ubuntu 16.04

Ubuntu 16.04 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

Ubuntu 18.04

Ubuntu 18.04 では、 AWS CloudHSM クライアントログが収集され、ジャーナル に保存されます。journalctl を使用して、これらのログを表示および管理できます。例えば、次のコマンドを使用して AWS CloudHSM クライアントログを表示します。

```
journalctl -f -u cloudhsm-client
```

Windows

- Windows クライアント 1.1.2+ の場合:

  AWS CloudHSM クライアントログは、プログラム`cloudhsm.log`ファイルフォルダ () のファイルに書き込まれます AWS CloudHSM `C:\Program Files\Amazon\CloudHSM\`。各ログファイル名には、 AWS CloudHSM クライアントがいつ起動されたかを示すタイムスタンプが付きます。

- Windows クライアント 1.1.1 以前の場合:

  クライアントのログはファイルに書き込まれません。ログは、コマンドプロンプトまたは AWS CloudHSM クライアントを開始した PowerShell ウィンドウに表示されます。

AWS CloudTrail および の使用 AWS CloudHSM

AWS CloudHSM は、 と統合されています。これは AWS CloudTrail、ユーザー、ロール、またはのサービスによって実行されたアクションを記録する AWS サービスです AWS CloudHSM。 は、のすべての API コールをイベント AWS CloudHSM として CloudTrail キャプチャします。キャプチャされた呼び出しには、 AWS CloudHSM コンソールからの呼び出しと AWS CloudHSM API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、 の CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます AWS CloudHSM。証跡を設定しない場合でも、 CloudTrail コンソールのイベント履歴 で最新のイベントを表示できます。によって収集された情報を使用して CloudTrail、 に対するリクエスト AWS CloudHSM、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。 AWS CloudHSM API オペレーションの完全なリストについては、「 API リファレンス」の「アクション」を参照してください。 AWS CloudHSM

AWS CloudHSM の情報 CloudTrail

CloudTrail AWS アカウントを作成すると、 がアカウントで有効になります。でアクティビティが発生すると AWS CloudHSM、そのアクティビティは CloudTrail イベント履歴 の他の AWS サービスイベントとともにイベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

のイベントなど、 AWS アカウント内のイベントの継続的な記録については AWS CloudHSM、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで作成した証跡がすべての AWS リージョンに適用されます。証跡は、AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail サポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからのログファイルの受信 CloudTrail

CloudTrail は AWS CloudHSM 、 DescribeClustersや などの読み取り専用オペレーション、、ListTags、 などの管理オペレーションを含むすべてのオペレーションを記録しますInitializeClusterCreatHsmDeleteBackup。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストが root または AWS Identity and Access Management (IAM) ユーザーの認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、CloudTrail userIdentity Element」を参照してください。

AWS CloudHSM ログファイルエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 CreateHsmアクションを示す CloudTrail AWS CloudHSM ログエントリを示しています。


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAJZVM5NEGZSTCITAMM:ExampleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/AdminRole/ExampleSession",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIY22AX6VRYNBGJSA",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2017-07-11T03:48:44Z"
            },
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "AROAJZVM5NEGZSTCITAMM",
                "arn": "arn:aws:iam::111122223333:role/AdminRole",
                "accountId": "111122223333",
                "userName": "AdminRole"
            }
        }
    },
    "eventTime": "2017-07-11T03:50:45Z",
    "eventSource": "cloudhsm.amazonaws.com",
    "eventName": "CreateHsm",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.179",
    "userAgent": "aws-internal/3",
    "requestParameters": {
        "availabilityZone": "us-west-2b",
        "clusterId": "cluster-fw7mh6mayb5"
    },
    "responseElements": {
        "hsm": {
            "eniId": "eni-65338b5a",
            "clusterId": "cluster-fw7mh6mayb5",
            "state": "CREATE_IN_PROGRESS",
            "eniIp": "10.0.2.7",
            "hsmId": "hsm-6lz2hfmnzbx",
            "subnetId": "subnet-02c28c4b",
            "availabilityZone": "us-west-2b"
        }
    },
    "requestID": "1dae0370-65ec-11e7-a770-6578d63de907",
    "eventID": "b73a5617-8508-4c3d-900d-aa8ac9b31d08",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Amazon CloudWatch Logs と AWS CloudHSM 監査ログの使用

アカウントの HSM がコマンド AWS CloudHSM ラインツールまたはソフトウェアライブラリ からコマンドを受信すると、コマンドの実行が監査ログ形式で記録されます。HSM の監査ログには、HSM の作成/削除、HSM へのログイン/ログアウト、ユーザーおよびキーの管理を含む、クライアント初期化のすべての管理コマンドが含まれます。このログは、HSM の状態を変更したアクションの信頼性のある記録を提供します。

AWS CloudHSM は HSM 監査ログを収集し、ユーザーに代わって Amazon CloudWatch Logs に送信します。Logs の機能を使用して、 CloudWatch ログの検索とフィルタリング、Amazon S3 へのログデータのエクスポートなど、 AWS CloudHSM 監査ログを管理できます。Amazon CloudWatch コンソールで HSM 監査ログを操作することも、 AWS CLIおよび CloudWatch Logs CloudWatch SDKs で Logs コマンドを使用することもできます。

トピック

- HSM 監査ログの記録の仕組み
- CloudWatch Logs での HSM 監査ログの表示
- HSM 監査ログの解釈
- HSM 監査ログのリファレンス

HSM 監査ログの記録の仕組み

監査ログ記録は、すべての AWS CloudHSM クラスターで自動的に有効になります。無効にしたり無効にしたりすることはできません。また、 がログを CloudWatch ログにエクスポート AWS CloudHSM できない設定はありません。各ログイベントには、タイムスタンプとイベントの順序を示すシーケンス番号があり、ログの改ざんを検出するために役立ちます。

HSM インスタンスごとに独自のログが生成されます。別々の HSM イベントの監査ログは、同じクラスターにある場合でも異なることがあります。たとえば、各クラスターの最初の HSM のみが HSM の初期化を記録します。初期化イベントは、バックアップからクローンされた HSM のログ記録には表示されません。同様に、キーを作成する場合、キーを生成する HSM がキーの生成イベントを記録します。クラスターの別の HSM は、同期を介してキーを受け取るときに、イベントを記録します。

AWS CloudHSM はログを収集し、 アカウントの CloudWatch ログに投稿します。ユーザーに代わって CloudWatch Logs サービスと通信するために、 はサービスにリンクされたロール AWS CloudHSM を使用します。ロールに関連付けられている IAM ポリシーにより AWS CloudHSM 、 は監査ログを CloudWatch ログに送信するために必要なタスクのみを実行できます。

> **Important**
>
> 2018 年 1 月 20 日以前にクラスターを作成し、サービスにリンクされたロールをまだ作成していない場合には、手動でこのロールを 1 つ作成する必要があります。これは、 が AWS CloudHSM クラスターから監査ログを受信 CloudWatch するために必要です。サービスにリンクされたロールの作成の詳細については、[サービスにリンクされたロールを理解する]

と、[IAM ユーザーガイド] の [サービスにリンクされたロールを作成する] を参照してください。

CloudWatch Logs での HSM 監査ログの表示

Amazon CloudWatch Logs は、監査ログをロググループに、ロググループ内でログストリーム に整理します。各ログエントリはイベント です。 は、クラスターごとに 1 つのロググループを作成し、クラスター内の HSM ごとに 1 つのログストリーム AWS CloudHSM を作成します。 CloudWatch Logs コンポーネントを作成したり、設定を変更したりする必要はありません。

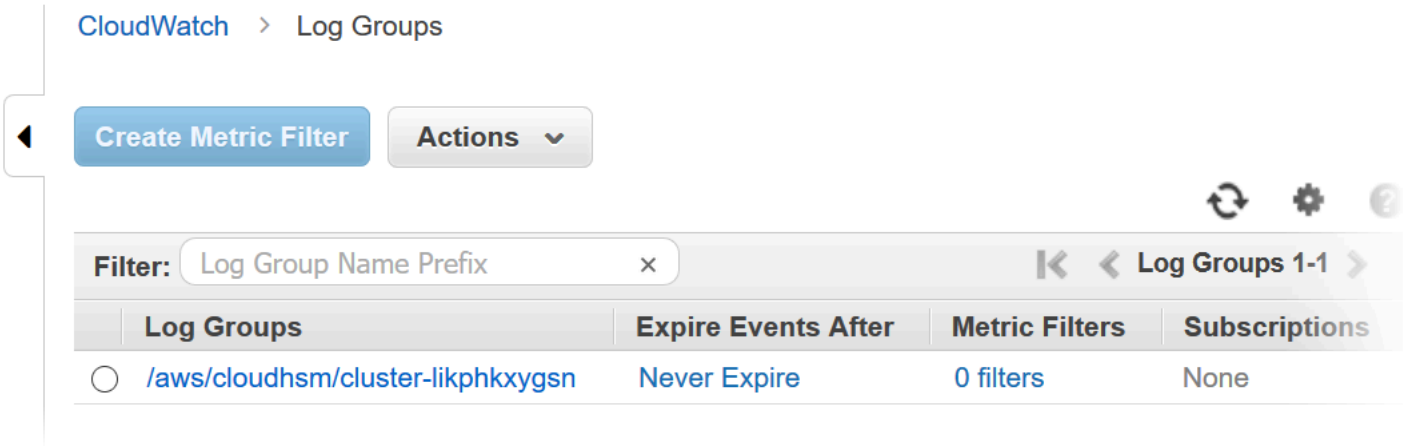
- ロググループの名前は `/aws/cloudhsm/<cluster ID>` です (たとえば、`/aws/cloudhsm/cluster-likphkxygsn`)。 AWS CLI または PowerShell コマンドでロググループ名を使用する場合は、必ず二重引用符で囲みます。
- ログストリーム名は HSM ID です (たとえば、`hsm-nwbbiqbj4jk`)。

  一般的には、各 HSM に 1 つのログストリームがあります。ただし、HSM ID を変更するすべてのアクション (HSM が失敗して置き換えられた場合など) は新しいログストリームを作成します。

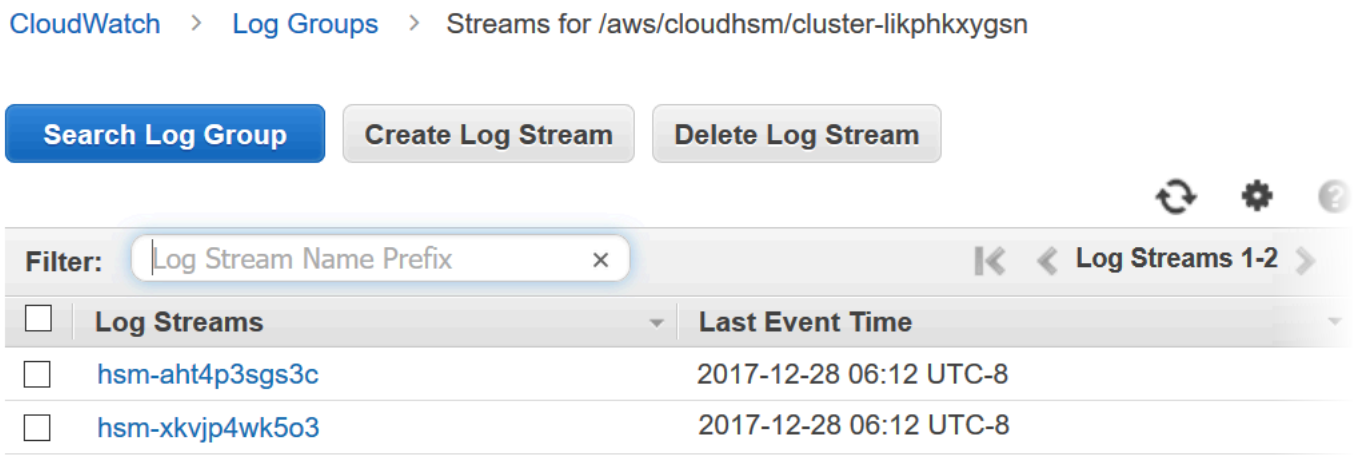
CloudWatch ログの概念の詳細については、「Amazon Logs ユーザーガイド」の「概念」を参照してください。 CloudWatch

HSM の監査ログは、 の CloudWatch 「ログ」ページ AWS Management Console、 のCloudWatch 「ログ」コマンド AWS CLI、CloudWatch 「ログコマンドレット PowerShell」、またはCloudWatch 「ログ SDKs」から表示できます。手順については、「Amazon Logs ユーザーガイド」の「ログデータの表示」を参照してください。 CloudWatch

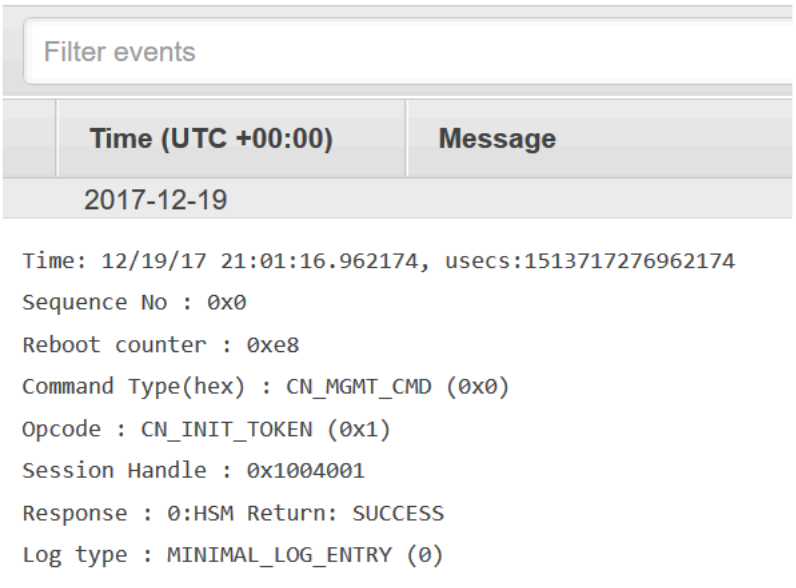
たとえば、次の図は AWS Management Consoleの `cluster-likphkxygsn` クラスターのロググループを示しています。



クラスターのロググループ名を選択すると、このクラスターの各 HSM のログストリームを表示することができます。つぎの図は、`cluster-likphkxygsn` クラスターの HSM のログストリームを示しています。

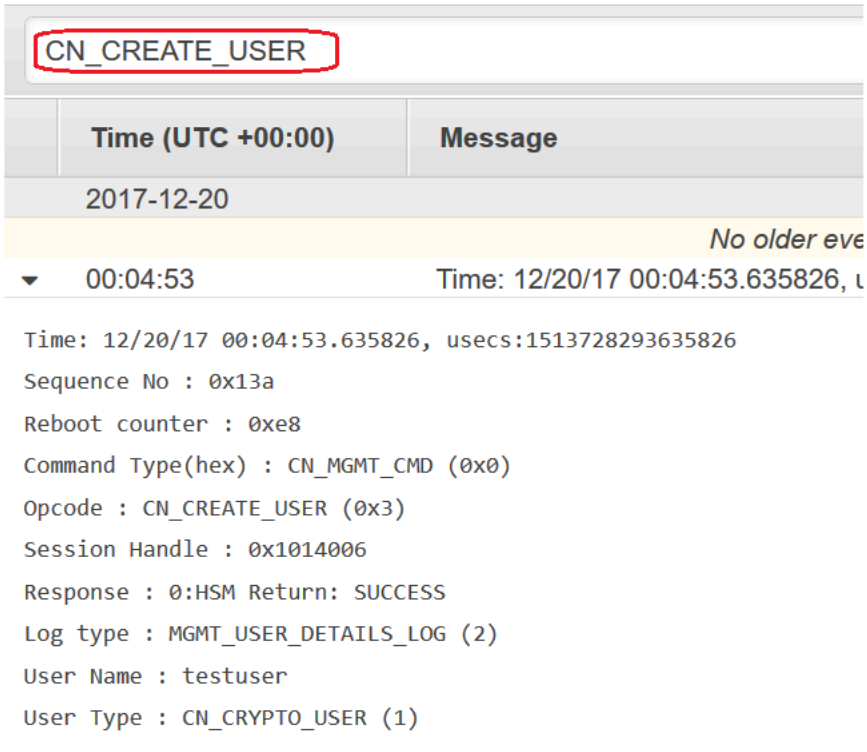


HSM のログストリーム名を選択すると、監査ログのイベントを表示することができます。たとえば、0x0 のシーケンス番号と `CN_INIT_TOKEN` の `Opcode` があるこのイベントは、通常の場合、各クラスターの最初の HSM の最初のイベントです。これには、このクラスターで HSM が初期化されたことが記録されています。



CloudWatch ログのすべての機能を使用して、監査ログを管理できます。たとえば、イベントのフィルター機能を使用すると、イベント内で特定のテキスト (CN_CREATE_USER Opcode など) を検索できます。

特定のテキストを含まないすべてのイベントを検索するには、テキストの前に負符号 (-) を追加します。たとえば、CN_CREATE_USER を含まないイベントを見つけるには、-CN_CREATE_USER を入力します。



HSM 監査ログの解釈

HSM 監査ログ内のイベントには、標準フィールドがあります。一部のイベントタイプには、イベントに関する有益な情報をキャプチャする追加のフィールドがあります。たとえば、ユーザーログインおよびユーザー管理イベントには、ユーザーのユーザー名とユーザータイプが含まれています。キー管理コマンドには、キーハンドルが含まれます。

いくつかのフィールドは、特に重要な情報を提供しています。Opcode は、記録している管理コマンドを識別します。Sequence No は、イベントをログストリームで識別して、記録された順序を示します。

たとえば、次のログイベント例は、HSM のログストリームで 2 番目のイベント (Sequence No: 0x1) です。これには、HSM が生成するパスワード暗号化キーが示され、これは起動ルーチンの一部です。

```
Time: 12/19/17 21:01:17.140812, usecs:1513717277140812
Sequence No : 0x1
Reboot counter : 0xe8
```

```
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GEN_PSWD_ENC_KEY (0x1d)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

以下のフィールドは、監査ログのすべての AWS CloudHSM イベントに共通です。

時間

イベントが発生した時間 (UTC タイムゾーン)。この時間は、人間が読み取れる時間と Unix 時間 (マイクロ秒) で表示されます。

再起動カウンタ

HSM ハードウェアが再起動されたときに増加する、32 ビットの永続的な序数カウンタ。

ログストリームのすべてのイベントには、同じ再起動カウンタ値があります。ただし、再起動カウンタは同じクラスターの別々の HSM インスタンスでは異なることがあるため、このカウンタはログストリームに固有ではないことがあります。

シーケンスなし

ログイベントごとに増加する 64 ビット序数カウンタ。各ログストリームの最初のイベントのシーケンス番号は 0 x 0 です。Sequence No 値でギャップがないようにする必要があります。シーケンス番号は、ログストリーム内でのみ一意です。

コマンドタイプ

コマンドのカテゴリを示す 16 進値です。 AWS CloudHSM ログストリームのコマンドには、CN_MGMT_CMD (0x0) あるいは CN_CERT_AUTH_CMD (0x9) のコマンドタイプがあります。

Opcode

実行された管理コマンドを識別します。 AWS CloudHSM 監査ログ0pcodeの値のリストについては、「」を参照してくださいHSM 監査ログのリファレンス。

セッションハンドル

コマンドが実行され、イベントがログされたセッションを識別します。

レスポンス

レスポンスを管理コマンドに記録します。Response フィールドで SUCCESS および ERROR 値を検索できます。

ログタイプ

コマンドを記録したログの AWS CloudHSM ログタイプを示します。

- `MINIMAL_LOG_ENTRY (0)`
- `MGMT_KEY_DETAILS_LOG (1)`
- `MGMT_USER_DETAILS_LOG (2)`
- `GENERIC_LOG`

監査ログイベントの例

ログストリーム内のイベントには、作成から削除までの HSM の履歴が記録されます。ログを使用して HSM のライフサイクルを確認し、オペレーションを把握することができます。イベントを解釈するときに、管理コマンドあるいはアクションを示す Opcode とイベントの順序を示す Sequence No に注目します。

トピック

- 例: クラスターの最初の HSM を初期化する
- ログインとログアウトイベント
- 例: ユーザーの作成と削除
- 例: キーペアの作成と削除
- 例: キーの生成と同期
- 例: キーのエクスポート
- 例: キーのインポート
- 例: キーの共有と共有解除

例: クラスターの最初の HSM を初期化する

各クラスターの最初の HSM の監査ログストリームは、クラスターの他の HSM のログストリームとは大幅に異なります。各クラスターの最初の HSM の監査ログは、この HSM の作成と初期化を記録します。クラスターに追加する HSM のログはバックアップから生成され、ログインイベントで始まります。

> **Important**
>
> 次の初期化エントリは、CloudHSM 監査 CloudWatch ログ記録機能のリリース (2018 年 8 月 30 日) 前に初期化されたクラスターのログには表示されません。詳細については、「ドキュメント履歴」を参照してください。

次のイベント例には、クラスターの最初の HSM のログストリームを表示しています。ログの最初のイベント (Sequence No 0x0 がついているもの) は、HSM(CN_INIT_TOKEN) を初期化するコマンドを表しています。このレスポンスは、コマンドが正常に実行されたことを示します (Response : 0: HSM Return: SUCCESS)。

```
Time: 12/19/17 21:01:16.962174, usecs:1513717276962174
Sequence No : 0x0
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INIT_TOKEN (0x1)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

このログストリーム例の 2 番目のイベント (Sequence No 0x1) は、HSM が使用するパスワード暗号化キーを作成するコマンド (CN_GEN_PSWD_ENC_KEY) を記録します。

これは、各クラスターの最初の HSM の一般的な起動シーケンスです。同じクラスターの後続の HSM は最初の HSM のクローンであるため、これらはおなじパスワード暗号化キーを使用します。

```
Time: 12/19/17 21:01:17.140812, usecs:1513717277140812
Sequence No : 0x1
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GEN_PSWD_ENC_KEY (0x1d)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

この例のログストリーミング (Sequence No 0x2) の 3 番目のイベントは、[アプライアンスユーザー (AU)] が作成したもので、 AWS CloudHSM サービスです。HSM ユーザーに関係するイベントには、ユーザー名とユーザータイプ用の追加フィールドが含まれています。

```
Time: 12/19/17 21:01:17.174902, usecs:1513717277174902
Sequence No : 0x2
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_APPLIANCE_USER (0xfc)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : app_user
User Type : CN_APPLIANCE_USER (5)
```

このログストリーム例の 4 番目のイベント (Sequence No 0x3) は、HSM の初期化を完了する CN_INIT_DONE イベントを記録します。

```
Time: 12/19/17 21:01:17.298914, usecs:1513717277298914
Sequence No : 0x3
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INIT_DONE (0x95)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

起動シーケンスの残りのイベントを追跡することができます。これらのイベントには、いくつかの口グイン/ログアウトイベント、およびキー暗号化キー (KEK) の生成が含まれている場合があります。次のイベントは、Precrypto Officer (PRECO) のパスワードを変更するコマンドを記録します。このコマンドは、クラスターをアクティブ化します。

```
Time: 12/13/17 23:04:33.846554, usecs:1513206273846554
Sequence No: 0x1d
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_CHANGE_PSWD (0x9)
Session Handle: 0x2010003
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: admin
User Type: CN_CRYPTO_PRE_OFFICER (6)
```

ログインとログアウトイベント

監査ログを解釈するときに、ユーザーのロギングおよび HSM のログイン/ログアウトを記録するイベントに注目します。このイベントは、どのユーザーがログインとログアウト間のシーケンスに示される管理コマンドの責任者であるかを判断するために役立ちます。

たとえば、このログエントリは admin という名前の crypto officer のログインを記録しています。シーケンス番号 (0x0) は、これがこのログストリームの最初のイベントであることを示しています。

ユーザーが HSM にログインすると、このクラスターの他の HSM にもこのユーザーのログインイベントが記録されます。ログインイベントの開始からすぐに、クラスターの他の HSM のログストリームで該当するログインイベントが見つかります。

```
Time: 01/16/18 01:48:49.824999, usecs:1516067329824999
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)
```

次のイベント例では、admin crypto officer のログアウトを記録しています。シーケンス番号 (0x2) は、これがログストリームの 3 番目のイベントであることを示しています。

ログインしているユーザーがログアウトせずにセッションを終了すると、ログストリームには、CN_LOGOUT イベントの代わりにCN_APP_FINALIZE あるいは終了セッションイベント (CN_SESSION_CLOSE) が含まれます。ログインイベントとは異なり、このログアウトイベントは通常の場合、このコマンドを実行する HSM にのみ記録されます。

```
Time: 01/16/18 01:49:55.993404, usecs:1516067395993404
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGOUT (0xe)
Session Handle : 0x7014000
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
```

```
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)
```

ユーザー名が無効なためにログイン試行が失敗した場合、HSM はログインコマンドで提供されたユーザー名およびユーザータイプを CN_LOGIN イベントに記録します。このレスポンスには、ユーザー名が存在しないことを示すメッセージ 157 が表示されます。

```
Time: 01/24/18 17:41:39.037255, usecs:1516815699037255
Sequence No : 0x4
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 157:HSM Error: user isn't initialized or user with this name doesn't exist
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : ExampleUser
User Type : CN_CRYPTO_USER (1)
```

パスワードが無効なためにログイン試行が失敗した場合、HSM はログインコマンドで提供されたユーザー名およびユーザータイプを CN_LOGIN イベントに記録します。レスポンスには、RET_USER_LOGIN_FAILURE エラーコードを示すエラーメッセージが表示されます。

```
Time: 01/24/18 17:44:25.013218, usecs:1516815865013218
Sequence No : 0x5
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 163:HSM Error: RET_USER_LOGIN_FAILURE
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

例: ユーザーの作成と削除

この例には、crypto officer (CO) がユーザーの作成および削除をしたことを記録するログイベントが示されています。

最初のイベントでは、CO (admin) の HSM へのログインを記録しています。シーケンス番号 (0x0) は、これがログストリームの最初のイベントであることを示しています。このイベントには、ログインしたユーザーのユーザー名とタイプが含まれています。

```
Time: 01/16/18 01:48:49.824999, usecs:1516067329824999
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)
```

ログストリームの次のイベント (シーケンス 0x1) には、CO が新しい暗号化ユーザー (CU) を作成したことが記録されています。このイベントには、新しいユーザーのユーザー名とタイプが含まれています。

```
Time: 01/16/18 01:49:39.437708, usecs:1516067379437708
Sequence No : 0x1
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_USER (0x3)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : bob
User Type : CN_CRYPTO_USER (1)
```

次に、CO は別の crypto officer (alice) を作成します。このシーケンス番号は、このアクションが前のアクションに従っていること (介在するアクションなしで) を示しています。

```
Time: 01/16/18 01:49:55.993404, usecs:1516067395993404
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_CO (0x4)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

後で、admin という名前の CO がログインし、alice という名前の crypto officer を削除しています。HSM は CN_DELETE_USER イベントを記録します。このイベントには、削除されたユーザーのユーザー名とタイプが含まれています。

```
Time: 01/23/18 19:58:23.451420, usecs:1516737503451420
Sequence No : 0xb
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_DELETE_USER (0xa1)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

例: キーペアの作成と削除

この例では、キーペアを作成/削除したことを HSM 監査ログに記録したイベントを示しています。

次のイベントでは、crypto_user という名前の暗号化ユーザー (CU) が HSM にログインしたことを記録しています。

```
Time: 12/13/17 23:09:04.648952, usecs:1513206544648952
Sequence No: 0x28
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_LOGIN (0xd)
Session Handle: 0x2014005
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

次に、CU がキーペア (CN_GENERATE_KEY_PAIR) を生成します。プライベートキーのキーハンドルは 131079 です。パブリックキーのキーハンドルは 131078 です。

```
Time: 12/13/17 23:09:04.761594, usecs:1513206544761594
Sequence No: 0x29
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_GENERATE_KEY_PAIR (0x19)
```

```
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
Public Key Handle: 131078
```

CU はすぐにこのキーペアを削除します。CN_DESTROY_OBJECT イベントは、パブリックキー (131078) の削除を記録しています。

```
Time: 12/13/17 23:09:04.813977, usecs:1513206544813977
Sequence No: 0x2a
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131078
Public Key Handle: 0
```

次に、2 番目の CN_DESTROY_OBJECT イベントに、プライベートキー (131079) の削除が記録されています。

```
Time: 12/13/17 23:09:04.815530, usecs:1513206544815530
Sequence No: 0x2b
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
Public Key Handle: 0
```

最後に、CU がログアウトします。

```
Time: 12/13/17 23:09:04.817222, usecs:1513206544817222
Sequence No: 0x2c
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_LOGOUT (0xe)
Session Handle: 0x2014004
```

```
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

例: キーの生成と同期

この例では、複数の HSM のクラスターでキーを作成した結果が示されています。1 つの HSM で生成されたキーはこの HSM からマスクされたオブジェクトとして抽出され、別の HSM にマスクされたオブジェクトとして挿入されます。

> **Note**
>
> クライアントツールで、キーを同期できない場合があります。または、特定数の HSM のみにキーを同期する min_srv パラメータがコマンドに含まれている場合があります。いずれの場合も、 AWS CloudHSM サービスはキーをクラスター内の他の HSMsに同期します。HSM はクライアント側の管理コマンドのみをログに記録するため、サーバー側の同期はこの HSM ログには記録されません。

まず、このコマンドを受信して実行する HSM のログストリームを検討します。このログストリームは HSM ID (hsm-abcde123456) で名付けられていますが、この HSM ID はログイベントには表示されません。

まず、 testuser Crypto User(CU) が hsm-abcde123456 HSM にログインします。

```
Time: 01/24/18 00:39:23.172777, usecs:1516754363172777
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

CU は exSymKey コマンドを実行して対称キーを生成します。hsm-abcde123456 HSM は 262152 のキーハンドルを使用して対称キーを生成します。HSM はそのログに CN_GENERATE_KEY イベントを記録します。

```
Time: 01/24/18 00:39:30.328334, usecs:1516754370328334
Sequence No : 0x1
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GENERATE_KEY (0x17)
Session Handle : 0xc008004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

hsm-abcde123456 のログストリームの次のイベントには、キーの同期プロセスの最初のステップが記録されています。新しいキー (キーハンドル 262152) は HSM からマスクオブジェクトとして抽出されています。

```
Time: 01/24/18 00:39:30.330956, usecs:1516754370330956
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_EXTRACT_MASKED_OBJECT_USER (0xf0)
Session Handle : 0xc008004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

ここで、同じクラスターの別の HSM である HSM hsm-zyxwv987654 のログストリームを検討します。このログストリームにも、testuser CU のログインイベントが含まれています。時刻値は、ユーザーが hsm-abcde123456 HSM に ログインしたすぐあとで発生したことを示しています。

```
Time: 01/24/18 00:39:23.199740, usecs:1516754363199740
Sequence No : 0xd
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7004004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

HSM のこのログストリームには、CN_GENERATE_KEY イベントがありません。ただし、この HSM へのキーの同期を記録するイベントはあります。CN_INSERT_MASKED_OBJECT_USER イベントは、キー 262152 をマスクされたオブジェクトとして受信したことを記録しています。これで、キー 262152 がクラスターの両方の HSM に存在するようになりました。

```
Time: 01/24/18 00:39:30.408950, usecs:1516754370408950
Sequence No : 0xe
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

CU ユーザーがログアウトすると、この CN_LOGOUT イベントはコマンドを受信する HSM のログストリームのみに表示されます。

例: キーのエクスポート

この例では、crypto user (CU) が複数の HSM があるクラスターからキーをエクスポートすることを記録した監査ログイベントを示しています。

次のイベントは、CU (testuser) が [key_mgmt_util] にログインしたことを記録しています。

```
Time: 01/24/18 19:42:22.695884, usecs:1516822942695884
Sequence No : 0x26
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7004004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

CU は exSymKey コマンドを実行して7、256 ビット AES キーであるキー をエクスポートします。このコマンドは、ラップキーとして HSM でキー 6 (256 ビット AES キー) を使用します。

コマンドを受信した HSM は、エクスポートされたキー 7 の CN_WRAP_KEY イベントを記録します。

```
Time: 01/24/18 19:51:12.860123, usecs:1516823472860123
Sequence No : 0x27
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_WRAP_KEY (0x1a)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 7
Public Key Handle : 0
```

次に、HSM はラップされたキーであるキー 6 の CN_NIST_AES_WRAP イベントを記録します。このキーはラップされ、すぐにラップ解除されますが、HSM は 1 つのイベントのみを記録します。

```
Time: 01/24/18 19:51:12.905257, usecs:1516823472905257
Sequence No : 0x28
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_NIST_AES_WRAP (0x1e)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 6
Public Key Handle : 0
```

この exSymKey コマンドはエクスポートされたキーをファイルに書き込みますが、HSM でキーの変更は行いません。したがって、クラスターの他の HSM のログには対応するイベントはありません。

例: キーのインポート

この例では、クラスターの HSM にキーをインポートしたことを記録する監査ログイベントを示しています。この例では、Crypto User (CU) は imSymKey コマンドを使用して AES キーを HSMs。このコマンドは、キー 6 をラップされたキーとして使用します。

コマンドを受信した HSM は、ラップされたキー 6 の CN_NIST_AES_WRAP イベントをまず記録します。

```
Time: 01/24/18 19:58:23.170518, usecs:1516823903170518
Sequence No : 0x29
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_NIST_AES_WRAP (0x1e)
```

```
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 6
Public Key Handle : 0
```

次に、この HSM はインポートオペレーションを表す CN_UNWRAP_KEY イベントを記録します。インポートされたキーには、11 のキーハンドルが割り当てられます。

```
Time: 01/24/18 19:58:23.200711, usecs:1516823903200711
Sequence No : 0x2a
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_UNWRAP_KEY (0x1b)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

新しいキーが生成あるいはインポートされると、クライアントツールは自動的にこの新しいキーをクラスターの他の HSM に同期する試みを行います。この場合、HSM はキー 11 がマスクオブジェクトとして HSM から抽出されたことを CN_EXTRACT_MASKED_OBJECT_USER イベントに記録します。

```
Time: 01/24/18 19:58:23.203350, usecs:1516823903203350
Sequence No : 0x2b
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_EXTRACT_MASKED_OBJECT_USER (0xf0)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

クラスターの他の HSM のログストリームには、新しくインポートされたキーの到着が示されます。

たとえば、同じクラスターの異なる HSM のログストリームには、このイベントが記録されています。この CN_INSERT_MASKED_OBJECT_USER イベントは、キー 11 を表すマスクされたオブジェクトの到着を記録します。

```
Time: 01/24/18 19:58:23.286793, usecs:1516823903286793
Sequence No : 0xb
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0xc008004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

例: キーの共有と共有解除

この例では、Crypto User (CU) が他の Crypto User (CU) とECC プライベートキーを共有または共有解除したときに記録される監査ログイベントを示します。CU は、shareKey コマンドを使用し、キーのハンドル、ユーザー ID、値 1 (共有) または 0 (共有解除) を行います。

次の例では、コマンドを受け取る HSM は、共有オペレーションを表す CM_SHARE_OBJECT イベントを記録します。

```
Time: 02/08/19 19:35:39.480168, usecs:1549654539480168
Sequence No : 0x3f
Reboot counter : 0x38
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_SHARE_OBJECT (0x12)
Session Handle : 0x3014007
Response : 0:HSM Return: SUCCESS
Log type : UNKNOWN_LOG_TYPE (5)
```

HSM 監査ログのリファレンス

AWS CloudHSM は、監査ログイベントに HSM 管理コマンドを記録します。各イベントには、発生したアクションとそのレスポンスを識別するオペレーションコード (0pcode) 値があります。0pcode 値を使用して、ログの検索、ソート、フィルタリングができます。

次の表は、 AWS CloudHSM 監査ログ0pcodeの値を定義します。

オペレーションコード (Opcode)	説明
[ユーザーログイン]: これらのイベントにはユーザー名とユーザータイプが含まれます	

オペレーションコード (Opcode)	説明
CN_LOGIN (0xd)	ユーザーログイン
CN_LOGOUT (0xe)	ユーザーログアウト
CN_APP_FINALIZE	HSM との接続が閉じられました。この接続からのセッションキーまたはクォーラムトークンはすべて削除されました。
CN_CLOSE_SESSION	HSM とのセッションが閉じられました。このセッションのセッションキーまたはクォーラムトークンはすべて削除されました。
ユーザー管理: これらのイベントにはユーザー名とユーザータイプが含まれます	
CN_CREATE_USER (0x3)	暗号化ユーザー (CU) を作成する
CN_CREATE_CO	Crypto Officer (CO) を作成する
CN_DELETE_USER	ユーザーを削除する
CN_CHANGE_PSWD	ユーザーのパスワードを変更する
CN_SET_M_VALUE	ユーザーアクションのクォーラム認証 (M of N) を設定する
CN_APPROVE_TOKEN	ユーザーアクションのクォーラム認証トークンを承認する
CN_DELETE_TOKEN	1 つ以上のクォーラムトークンを削除する
CN_GET_TOKEN	クォーラムオペレーションを開始する署名トークンをリクエストする
キー管理: このイベントにはキーハンドルが含まれます	
CN_GENERATE_KEY	対称キーの生成
CN_GENERATE_KEY_PAIR (0x19)	非対称キーペアを生成する

オペレーションコード (Opcode)	説明
CN_CREATE_OBJECT	パブリックキー (ラップなし) のインポート
CN_MODIFY_OBJECT	キー属性を設定する
CN_DESTROY_OBJECT (0x11)	セッションキーの削除
CN_TOMBSTONE_OBJECT	トークンキーの削除
CN_SHARE_OBJECT	キーの共有あるいは非共有
CN_WRAP_KEY	キー (wrapKey) の暗号化されたコピーのエクスポート
CN_UNWRAP_KEY	キー (unwrapKey) の暗号化されたコピーのインポート
CN_DERIVE_KEY	既存のキーから対称キーを取得する
CN_NIST_AES_WRAP	AES キーを使用してキーを暗号化または復号する
CN_INSERT_MASKED_OBJECT_USER	クラスター内の別の HSM の属性を含む暗号化キーを挿入します。
CN_EXTRACT_MASKED_OBJECT_USER	HSM からの属性を持つキーをラップ/暗号化して、クラスター内の別の HSM に送信します。
Back up HSMs	
CN_BACKUP_BEGIN	バックアッププロセスを開始する
CN_BACKUP_END	バックアッププロセスを完了しました
CN_RESTORE_BEGIN	バックアップからの復元を開始する
CN_RESTORE_END	バックアップからの復元プロセスを完了しました
Certificate-Based Authentication	

オペレーションコード (Opcode)	説明
CN_CERT_AUTH_STORE_CERT	クラスター証明書を保存します。
HSM Instance Commands	
CN_INIT_TOKEN (0x1)	HSM 初期化プロセスを開始する
CN_INIT_DONE	HSM 初期化プロセスが終了しました
CN_GEN_KEY_ENC_KEY	キー暗号化キー (KEK) の生成
CN_GEN_PSWD_ENC_KEY (0x1d)	パスワード暗号化キー (PEK) の生成
HSM crypto commands	
CN_FIPS_RAND	FIPS 準拠の乱数を生成する

の CloudWatch メトリクスの取得 AWS CloudHSM

を使用して CloudWatch 、 AWS CloudHSM クラスターをリアルタイムでモニタリングします。メトリクスは、リージョン、クラスター ID、またはクラスター ID と HSM ID ごとにグループ化できます。

AWS/CloudHSM 名前空間には、次のメトリクスが含まれます。

メトリクス	説明
HsmUnhealthy	HSM インスタンスが正常に動作していません。 は、異常なインスタンス AWS CloudHSM を自動的に置き換えます。HSM の置き換え中は、パフォーマンスへの影響を抑えるために、クラスターサイズを積極的に拡大することもできます。
HsmTemperature [1]	ハードウェアプロセッサのジャンクション温度です。温度が 110 度に達すると、システムがシャットダウンします。
HsmKeysSessionOccupied	HSM インスタンスにより使用されているセッションキーの数です。

メトリクス	説明
HsmKeysTokenOccupied	HSM インスタンスとクラスターにより使用されるトークンキーの数です。
HsmSslCtxsOccupied [1]	HSM インスタンスに対して現在確立されている end-to-end 暗号化されたチャネルの数。最大 2,048 のチャネルが許容されます。
HsmSessionCount	HSM インスタンスへのオープン接続の数です。最大 2,048 の接続が許容されます。デフォルトでは、クライアントデーモンは、1 つの end-to-end 暗号化されたチャネルで各 HSM インスタンスで 2 つのセッションを開くように設定されています。また、HSM のヘルスをモニタリングするために、HSM で最大 2 つの接続がHSMs AWS CloudHSM 場合があります。
HsmUsersAvailable	作成可能な追加ユーザーの数です。これは、ユーザーの最大数 (に記載HsmUsersMax) から現在までに作成されたユーザーを引いた数に等しくなります。
HsmUsersMax [1]	HSM インスタンスで作成可能なユーザーの最大数です。現在のところ、これは 1,024 です。
InterfaceEth2OctetsInput [1]	現在の HSM への受信トラフィックの累積合計です。
InterfaceEth2OctetsOutput [1]	現在の HSM への送信トラフィックの累積合計です。

- [1] このメトリクスは hsm2m.medium では使用できません。

AWS CloudHSM パフォーマンス

本番稼働用クラスターの場合は、リージョン内の 2 つのアベイラビリティーゾーン間に異なる HSM インスタンスが分散している必要があります。クラスターの負荷テストを行って予測すべきピーク負荷を決定し、高可用性を確保するためにクラスターに HSM を 1 つ追加することを推奨します。新しく生成されるキーの耐久性を必要とするアプリケーションの場合は、リージョン内の異なるアベイラビリティーゾーンに 3 つの HSM インスタンスを分散させることをお勧めします。

パフォーマンスデータ

AWS CloudHSM クラスターのパフォーマンスは、特定のワークロードによって異なります。パフォーマンスを向上させるために、クラスターに HSM インスタンスを追加できます。パフォーマンスは、設定、データサイズ、および EC2 インスタンスにかかる追加のアプリケーション負荷によって異なる場合があります。アプリケーションの負荷テストを行い、スケーリングの必要性を判断することをお勧めします。

次の表は、hsm1.medium インスタンスを使用する EC2 インスタンスで実行される一般的な暗号化アルゴリズムのおおよそのパフォーマンスを示しています。

hsm1.medium のパフォーマンスデータ

操作	2 つの HSM クラスター [1]	3 つの HSM クラスター [2]	6 つの HSM クラスター [3]
RSA 2048 ビットサイン	2,000 オペレーション/秒	3,000 オペレーション/秒	5,000 オペレーション/秒
EC P256 サイン	500 オペレーション/秒	750 オペレーション/秒	1,500 オペレーション/秒

- [1] 1 つの c4.large EC2 インスタンスと EC2 インスタンスと同じ AZ にある 1 つの HSM で実行されている Java マルチスレッドアプリケーションを備えた 2 つの HSM クラスター。
- [2] 1 つの c4.large EC2 インスタンスと EC2 インスタンスと同じ AZ にある 1 つの HSM で実行されている Java マルチスレッドアプリケーションを備えた 3 つの HSM クラスター。
- [3] 1 つの c4.large EC2 インスタンスと EC2 インスタンスと同じ AZ にある 1 つの HSM で実行されている Java マルチスレッドアプリケーションを備えた 6 つの HSM クラスター。

HSM スロットリング

ワークロードがクラスターの HSM 容量を超えると、HSM がビジー状態またはスロットリングされていることを示すエラーメッセージが表示されます。この場合の対処方法の詳細については、HSM スロットリング を参照してください

のセキュリティ AWS CloudHSM

のクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 は、安全に使用できるサービスも提供します。コンプライアンスAWS プログラムコンプライアンスプログラム の一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。に適用されるコンプライアンスプログラムの詳細については AWS CloudHSM、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」「コンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、 の使用時に責任共有モデルを適用する方法を理解するのに役立ちます AWS CloudHSM。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために AWS CloudHSM を設定する方法を示します。また、 AWS CloudHSM リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

内容

でのデータ保護 AWS CloudHSM

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS CloudHSM。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーのよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿されたAWS 責任共有モデルおよび GDPRのブログ記事を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、AWS CloudHSM または SDK を使用して AWS CLIまたは他の AWS のサービス を操作する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

保管中の暗号化

AWS CloudHSM が HSM からバックアップを作成すると、HSM はデータを に送信する前に暗号化します AWS CloudHSM。データは、一意の一時的な暗号化キーを使用して暗号化されます。詳細については、「AWS CloudHSM クラスターバックアップ」を参照してください。

転送中の暗号化

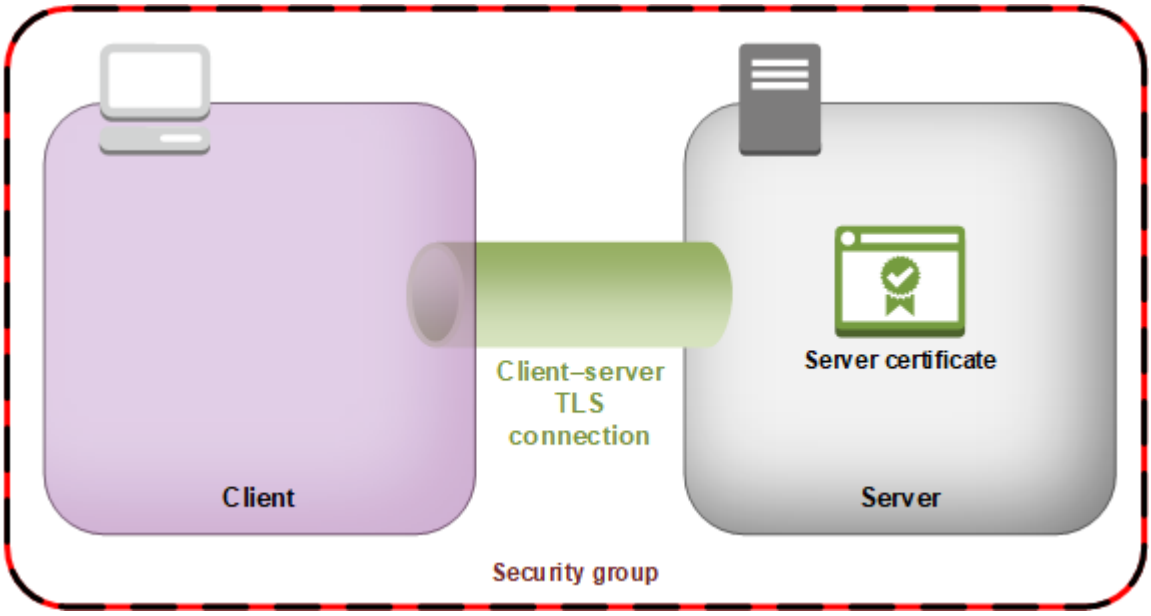
AWS CloudHSM クライアントとクラスター内の HSM 間の通信は、エンドツーエンドで暗号化されます。この通信は、クライアントと HSM によってのみ復号できます。詳細については、「E nd-to-end 暗号化」を参照してください。

AWS CloudHSM クライアントの end-to-end 暗号化

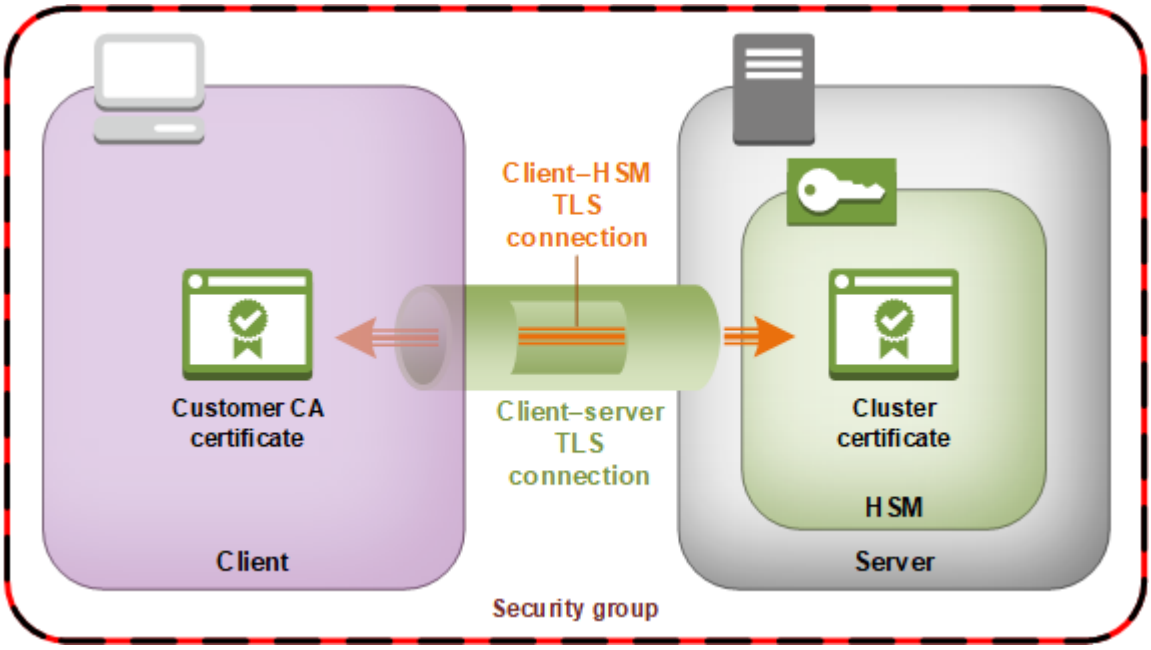
クライアントインスタンスとクラスターの HSM の間の通信はエンドツーエンドで暗号化されます。クライアントと HSM のみが、通信を復号できます。

以下のプロセスでは、クライアントが HSM との end-to-end 暗号化された通信を確立する方法について説明します。

1. クライアントは、HSM ハードウェアをホストするサーバーと Transport Layer Security (TLS) 接続を確立します。クラスターのセキュリティグループによって、セキュリティグループのクライアントインスタンスからのみ、サーバーへのインバウンドトラフィックが許可されます。また、クライアントはサーバーの証明書をチェックして、それが信頼されたサーバーであることを確認します。



2. 次に、クライアントは HSM ハードウェアと暗号化された接続を確立します。HSM には、独自の証明機関 (CA) で署名されたクラスター証明書があり、クライアントには CA のルート証明書があります。クライアントと HSM の暗号化された接続が確立される前に、クライアントはルート証明書に対して HSM のクラスター証明書を確認します。接続が確立されるのは、HSM が信頼済みであることをクライアントが正しく確認した場合のみです。



クラスターバックアップのセキュリティ

AWS CloudHSM が HSM からバックアップを作成すると、HSM はすべてのデータを暗号化してから に送信します AWS CloudHSM。データがプレーンテキスト形式で HSM から外部に出ることはありません。さらに、 はバックアップの復号に使用されるキーにアクセス AWS できない AWS ため、によるバックアップの復号化はできません。

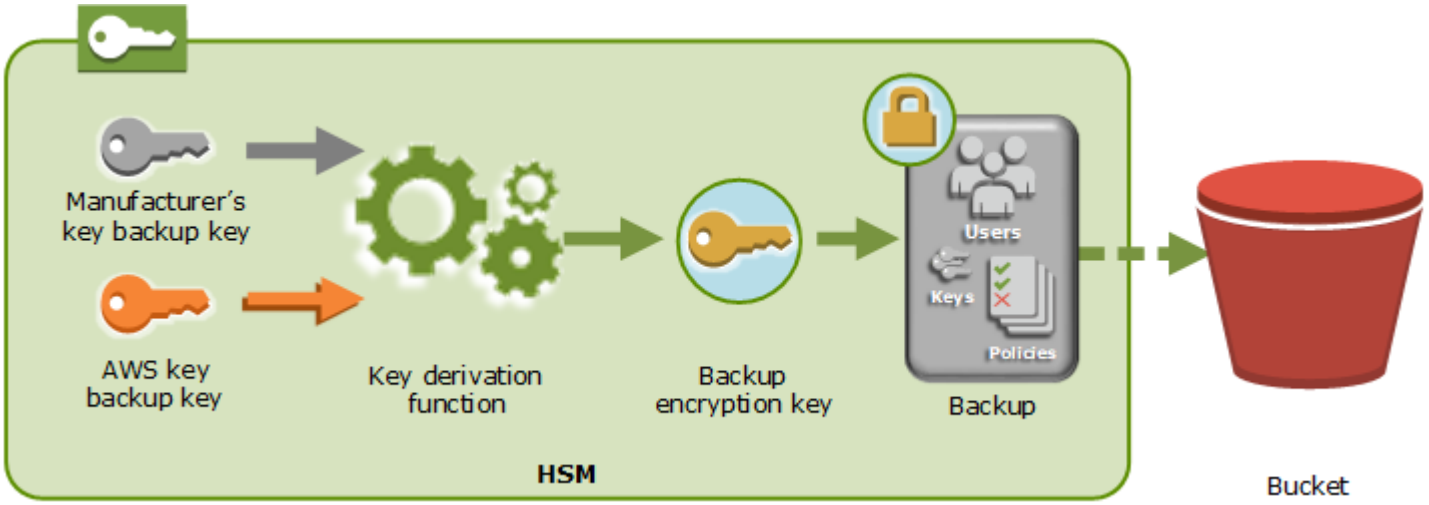
データを暗号化するために、HSM はエフェメラルバックアップキー (EBK) として知られる一意の暗号化キーを一時的に使用します。EBK は、 がバックアップを作成するときに HSM 内で生成される AES 256 ビット暗号化キー AWS CloudHSM です。HSM は EBK を生成し、それを使用して NIST special publication 800-38F に準拠する FIPS 承認 AES キーラップメソッドで HSM のデータを暗号化します。次に、HSM は暗号化されたデータを に渡します AWS CloudHSM。暗号化されたデータには、EBK の暗号化済みコピーが含まれています。

EBK を暗号化するために、HSM は永続的バックアップキー (PBK) として知られる別の暗号化キーを使用します。PBK も、AES 256 ビット暗号化キーです。PBK を生成するために、HSM は NIST

special publication 800-108 に準拠する FIPS 承認キー取得機能 (KDF) をカウンターモードで使用します。この KDF への入力には次のものがあります。

- ハードウェア製造元が HSM に永続的に埋め込んだ、製造元キーバックアップキー (MKBK)。
- AWS キーバックアップキー (AKBK)。 によって最初に設定されるときに HSM に安全にインストールされます AWS CloudHSM。

次の図に暗号化プロセスがまとめてあります。バックアップ暗号化キーは、永続的なバックアップキー (PBK) とエフェメラルバックアップキー (EBK) を指します。



AWS CloudHSM は、同じ製造元が作成した AWS所有HSMs にのみバックアップを復元できます。すべてのバックアップがすべてのユーザー、キー、およびオリジナルの HSM を含んでいるため、復元された HSM はオリジナルと同じ保護およびアクセス制御を含んでいます。復元されたデータは、復元前に HSM にあった可能性がある他のデータをすべて上書きします。

バックアップは暗号化されたデータのみで構成されます。サービスが Amazon S3 にバックアップを保存する前に、サービスは AWS Key Management Service () を使用してバックアップを再度暗号化しますAWS KMS。

の Identity and Access Management AWS CloudHSM

AWS ではセキュリティ認証情報を使用して、ユーザーを識別し、AWS リソースへのアクセスを付与します。 AWS Identity and Access Management (IAM) の機能を使用して、他のユーザー、サービス、アプリケーションが AWS リソースを完全または限定的な方法で使用できるようにします。その際、お客様のセキュリティ認証情報は共有されません。

デフォルトでは、IAM ユーザーには、AWS リソースを作成、表示、変更するためのアクセス権限はありません。ロードバランサーなどのリソースにアクセスすること、およびタスクを実行することを IAM ユーザーに許可するには、次の操作を行います。

1. 必要な特定のリソースと API アクションを使用するアクセス許可を IAM ユーザーに付与する IAM ポリシーを作成します。
2. IAM ユーザーまたは IAM ユーザーが属するグループに、ポリシーをアタッチします。

ポリシーをユーザーまたはユーザーのグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。

たとえば、IAM を使用して、お客様の AWS アカウントでユーザーとグループを作成できます。IAM ユーザーは、人、システム、またはアプリケーションです。その後、ユーザーとグループにアクセス許可を付与すると、IAM ポリシーを使用して指定したリソースに対する特定のアクションを実行できます。

IAM ポリシーを使用したアクセス権限の付与

ポリシーをユーザーまたはユーザーのグループにアタッチする場合、ポリシーによって特定リソースの特定タスクを実行するユーザーの権限が許可または拒否されます。

IAM ポリシーは 1 つ以上のステートメントで構成される JSON ドキュメントです。各ステートメントは、次の例に示すように構成されます。

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect": "effect",
    "Action": "action",
    "Resource": "resource-arn",
    "Condition": {
      "condition": {
        "key":"value"
      }
    }
  }]
}
```

- [Effect (効果)] — effect は、`Allow` または `Deny` にすることができます。デフォルトでは、IAM ユーザーはリソースおよび API アクションを使用するアクセス許可がないため、リクエストはす

べて拒否されます。明示的な許可はデフォルトに優先します。明示的な拒否はすべての許可に優先します。

- Action (アクション) — action は、アクセス許可を付与または拒否する対象とする、特定の API アクションです。アクション条件を指定する方法については、の API アクション AWS CloudHSM を参照してください。
- リソース — action. AWS CloudHSM does の影響を受けるリソースは、リソースレベルのアクセス許可をサポートしていません。すべての AWS CloudHSM リソースを指定するには、* ワイルドカードを使用する必要があります。
- [Condition (条件)] — ポリシーが有効になるタイミングを制御する条件を必要に応じて使用できます。詳細については、「の条件キー AWS CloudHSM」を参照してください。

詳細については、IAM ユーザーガイドを参照してください。

の API アクション AWS CloudHSM

IAM ポリシーステートメントの Action 要素で、 AWS CloudHSM が提供する任意の API アクションを指定できます。次の例に示すように、アクション名の前に小文字の文字列 `cloudhsm:` を指定する必要があります。

```
"Action": "cloudhsm:DescribeClusters"
```

1 つのステートメントで複数のアクションを指定するには、次の例に示すように、アクションをカンマで区切って全体を角括弧で囲みます。

```
"Action": [
    "cloudhsm:DescribeClusters",
    "cloudhsm:DescribeHsm"
]
```

ワイルドカード (*) を使用して複数のアクションを指定することもできます。次の例では、 で始 AWS CloudHSM まる のすべての API アクション名を指定しますList。

```
"Action": "cloudhsm:List*"
```

のすべての API アクションを指定するには AWS CloudHSM、次の例に示すように * ワイルドカードを使用します。

```
"Action": "cloudhsm:*"
```

の API アクションのリストについては AWS CloudHSM、「アクションAWS CloudHSM」を参照してください。

の条件キー AWS CloudHSM

ポリシーを作成するときは、ポリシーをいつ有効にするか制御する条件を指定できます。各条件には 1 つ以上のキーと値のペアが含まれます。グローバル条件キーとサービス固有の条件キーがあります。

AWS CloudHSM には、サービス固有のコンテキストキーはありません。

グローバル条件キーの詳細については、IAM ユーザーガイド の AWS global condition context keys を参照してください。

の事前定義された AWS 管理ポリシー AWS CloudHSM

AWS によって作成された管理ポリシーは、一般的ユースケースに必要なアクセス権限を付与します。これらのポリシーを、 AWS CloudHSM に対して必要なアクセス権に基づいて IAM ユーザーにアタッチできます。

- AWSCloudHSMFullAccess — AWS CloudHSM 機能を使用するために必要なフルアクセスを付与します。
- AWSCloudHSMReadOnlyAccess — AWS CloudHSM 機能への読み取り専用アクセスを許可します。

のカスタマー管理ポリシー AWS CloudHSM

の実行に必要なアクセス許可のみ AWS CloudHSM を含む の IAM 管理者グループを作成することをお勧めします AWS CloudHSM。適切なアクセス許可を持つポリシーをこのグループにアタッチします。必要に応じて、IAM ユーザーをグループに追加します。追加する各ユーザーは、管理者グループからポリシーを継承します。

また、ユーザーが必要とする権限に基づいて、追加のユーザーグループを作成することをお勧めします。これにより、信頼されたユーザーのみが重要な API アクションにアクセスできるようになります。たとえば、クラスターと HSM への読み取り専用アクセスを許可するために使用するユーザーグループを作成できます。このグループでは、ユーザーがクラスターまたは HSM を削除できないため、信頼できないユーザーが運用ワークロードの可用性に影響を与えることがありません。

新しい AWS CloudHSM 管理機能が時間の経過とともに追加されるので、信頼できるユーザーのみがすぐにアクセスできるようになります。作成時に、制限付きのアクセス許可をポリシーに割り当てることで、後に新しい機能のアクセス許可を手動でユーザーに割り当てることができます。

のポリシーの例を次に示します AWS CloudHSM。ポリシーの作成方法とIAM ユーザーグループへのアタッチ方法の詳細については、IAM ユーザーガイド の 「[JSON] タブでのポリシーの作成 」を参照してください。

例

- 読み取り専用のアクセス許可
- パワーユーザーのアクセス許可
- 管理者権限

Example 例: 読み取り専用アクセス許可

このポリシーでは、DescribeClusters および DescribeBackups API アクションへのアクセスを許可します。また、このポリシーには、特定の Amazon EC2 API アクションのアクセス許可が含まれています。このポリシーでは、ユーザーはクラスターまたは HSM を削除することはできません。

```
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
         "cloudhsm:DescribeBackups",
         "cloudhsm:ListTags"
      ],
      "Resource": "*"
   }
}
```

Example 例: パワーユーザーのアクセス許可

このポリシーは、 AWS CloudHSM API アクションのサブセットへのアクセスを許可します。また、このポリシーには特定の Amazon EC2 アクションのアクセス許可が含まれています。このポリシーでは、ユーザーはクラスターまたは HSM を削除することはできません。が アカウントでAWSServiceRoleForCloudHSMサービスにリンクされたロールを自動的に作成 AWS CloudHSM で

きるようにするには、 iam:CreateServiceLinkedRoleアクションを含める必要があります。このロールにより、 はイベント AWS CloudHSM をログに記録できます。詳細については、「のサービスにリンクされたロール AWS CloudHSM」を参照してください。

> **Note**
>
> 各 API の特定の権限を確認するには、「サービス認証リファレンス」の「AWS CloudHSM 向けアクション、リソース、および条件キー」を参照してください。

```
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
         "cloudhsm:DescribeBackups",
         "cloudhsm:CreateCluster",
         "cloudhsm:CreateHsm",
         "cloudhsm:RestoreBackup",
         "cloudhsm:CopyBackupToRegion",
         "cloudhsm:InitializeCluster",
         "cloudhsm:ListTags",
         "cloudhsm:TagResource",
         "cloudhsm:UntagResource",
         "ec2:CreateNetworkInterface",
         "ec2:DescribeNetworkInterfaces",
         "ec2:DescribeNetworkInterfaceAttribute",
         "ec2:DetachNetworkInterface",
         "ec2:DeleteNetworkInterface",
         "ec2:CreateSecurityGroup",
         "ec2:AuthorizeSecurityGroupIngress",
         "ec2:AuthorizeSecurityGroupEgress",
         "ec2:RevokeSecurityGroupEgress",
         "ec2:DescribeSecurityGroups",
         "ec2:DeleteSecurityGroup",
         "ec2:CreateTags",
         "ec2:DescribeVpcs",
         "ec2:DescribeSubnets",
         "iam:CreateServiceLinkedRole"
      ],
      "Resource": "*"
```

```
    }
}
```

Example 例: 管理者権限

このポリシーは、HSM とクラスターを削除するアクションを含む、すべての AWS CloudHSM API アクションへのアクセスを許可します。 HSMs また、このポリシーには特定の Amazon EC2 アクションのアクセス許可が含まれています。が アカウントでAWSServiceRoleForCloudHSMサービスにリンクされたロールを自動的に作成 AWS CloudHSM できるようにするには、`iam:CreateServiceLinkedRole`アクションを含める必要があります。このロールにより、 はイベント AWS CloudHSM をログに記録できます。詳細については、「のサービスにリンクされたロール AWS CloudHSM」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":[
         "cloudhsm:*",
         "ec2:CreateNetworkInterface",
         "ec2:DescribeNetworkInterfaces",
         "ec2:DescribeNetworkInterfaceAttribute",
         "ec2:DetachNetworkInterface",
         "ec2:DeleteNetworkInterface",
         "ec2:CreateSecurityGroup",
         "ec2:AuthorizeSecurityGroupIngress",
         "ec2:AuthorizeSecurityGroupEgress",
         "ec2:RevokeSecurityGroupEgress",
         "ec2:DescribeSecurityGroups",
         "ec2:DeleteSecurityGroup",
         "ec2:CreateTags",
         "ec2:DescribeVpcs",
         "ec2:DescribeSubnets",
         "iam:CreateServiceLinkedRole"
      ],
      "Resource":"*"
   }
}
```

のサービスにリンクされたロール AWS CloudHSM

以前に に作成した IAM ポリシーには `iam:CreateServiceLinkedRole` action. AWS CloudHSM defines という名前のサービスにリンクされたロールのカスタマー管理ポリシー AWS CloudHSMが含まれていますAWSServiceRoleForCloudHSM。ロールは によって事前定義 AWS CloudHSM されており、ユーザーに代わって他の AWS サービスを呼び出す AWS CloudHSM ために必要なアクセス許可が含まれています。ロールは、ロールポリシーと信頼ポリシーのアクセス許可を手動で追加する必要がないため、サービスを簡単に設定できます。

ロールポリシーにより AWS CloudHSM 、 はユーザーに代わって Amazon CloudWatch Logs ロググループとログストリームを作成し、ログイベントを書き込むことができます。これは以下と IAM コンソールで確認できます。

```
{
    "Version": "2018-06-12",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

AWSServiceRoleForCloudHSM ロールの信頼ポリシーにより、 AWS CloudHSM はロールを引き受けることができます。

```
{
  "Version": "2018-06-12",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "cloudhsm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

サービスにリンクされたロールを作成する (自動)

AWS CloudHSM は、 AWS CloudHSM 管理者グループの作成時に定義したアクセス許可に `iam:CreateServiceLinkedRole`アクションを含めると、クラスターの作成時にAWSServiceRoleForCloudHSMロールを作成します。のカスタマー管理ポリシー AWS CloudHSM を参照してください。

すでに 1 つ以上のクラスターがあり、AWSServiceRoleForCloudHSMロールを追加するだけの場合は、コンソール、create-cluster コマンド、または CreateCluster API オペレーションを使用してクラスターを作成できます。次に、コンソール、delete-cluster コマンド、または DeleteCluster API オペレーションを使用して削除します。新しいクラスターを作成すると、サービスにリンクされたロールが作成され、アカウントのすべてのクラスターに適用されます。または、ロールを手動で作成することもできます。詳細については、次のセクションを参照してください。

> **Note**
>
> AWSServiceRoleForCloudHSM ロールを追加するためにのみクラスターを作成する場合は、で説明されているすべてのステップを実行してクラスターの開始方法 AWS CloudHSMを作成する必要はありません。

サービスにリンクされたロールを作成する (手動)

IAM コンソール AWS CLI、または API を使用してAWSServiceRoleForCloudHSMロールを作成できます。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの作成」を参照してください。

サービスにリンクされたロールを編集する

AWS CloudHSM では、AWSServiceRoleForCloudHSMロールを編集することはできません。たとえば、ロールの作成後、さまざまなエンティティが名前でロールを参照する可能性があるため、

名前を変更することはできません。また、ロールのポリシーを変更することもできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

サービスリンクロールの削除

サービスにリンクされたロールは、適用されたクラスターが存在する限り削除できません。ロールを削除するには、まずクラスタ内の各 HSM を削除してからクラスターを削除する必要があります。アカウント内のすべてのクラスターを削除する必要があります。その後、IAM コンソール AWS CLI、または API を使用してロールを削除できます。クラスターの削除の詳細については、「AWS CloudHSM クラスターの削除」を参照してください。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

コンプライアンス

FIPS モードのクラスターの場合、 は PCI-PIN、PCI-3DS、および SOC2 のコンプライアンス要件を満たす FIPS 承認の HSMs AWS CloudHSM を提供します。 AWS CloudHSM また、 では、非 FIPS モードのクラスターを選択することもできます。各 に適用される証明書とコンプライアンス要件の詳細については、「」を参照してくださいAWS CloudHSM クラスターモードと HSM タイプ。

FIPS 検証済みの HSM を利用すると、 AWS クラウドのデータセキュリティに関する企業、契約、規制のコンプライアンス要件を満たすことができます。

FIPS 140-2 への準拠

連邦情報処理規格（Federal Information Processing Standards/FIPS）出版物140-2 は、機密情報を保護する暗号モジュールのセキュリティ要件を規定する米国政府のセキュリティ基準です。が提供する hsm1.medium HSMsタイプ AWS CloudHSM は、FIPS 140-2 レベル 3 認定 (証明書 #4218) です。詳細については、「ハードウェアの FIPS 検証」を参照してください。

PCI DSS コンプライアンス

ペイメントカード業界データセキュリティ基準 (PCI DSS) は、PCI Security Standards Council が管理する専有情報のセキュリティ標準です。が提供する HSMs PCI DSS AWS CloudHSM に準拠しています。

PCI PIN コンプライアンス

PCI PIN は、個人識別番号 (PIN) データ、および ATMs および (POS) ターミナルでのトランザクションに使用される情報を送信、処理、管理するためのセキュリティ要件と point-of-sale 評価基

準を提供します。が提供する hsm1.medium HSMsは、2023 年 1 月以降、PCI PIN に準拠 AWS CloudHSM しています。詳細については、「AWS CloudHSM is now PCI PIN certified」という記事を参照してください。

PCI-3DS コンプライアンス

PCI 3DS (またはスリードメインセキュア、3-D セキュア) は EMV 3D セキュア e コマース決済のデータセキュリティを提供します。PCI 3DS は、オンラインショッピングに別のセキュリティレイヤーを提供します。が提供するタイプ hsm1.medium HSMs AWS CloudHSM は PCI-3DS に準拠しています。

SOC2

SOC2 は、サービス組織がクラウドとデータセンターのセキュリティ管理を実証するのに役立つフレームワークです。AWS CloudHSM は、信頼できるサービス原則に準拠するために、重要な領域に SOC2 コントロールを実装しています。詳細については、AWS SOC のよくある質問ページを参照してください。

AWS CloudHSM PCI-PIN コンプライアンスFAQs

PCI PIN は、個人識別番号 (PIN) データ、および ATMs や point-of-sale (POS) ターミナルでのトランザクションに使用される情報を送信、処理、管理するためのセキュリティ要件と評価基準を提供します。

PCI-PIN Attestation of Compliance (AOC) および Responsibility Summary は、コンプライアンスレポートへのオンデマンドアクセス用のセルフサービスポータルである AWS Artifact を通じて入手できます。詳細については、AWS マネジメントコンソールで AWS Artifact にサインインするか、「AWS Artifact の開始方法」で詳細をご覧ください。

よくある質問

Q: コンプライアンスと責任の証明の概要 (Attestation of Compliance and Responsibility Summary) とはどのようなものですか?

コンプライアンス認証 (AOC) は、認定 PIN 評価者 (QPA) による認証が PCI-PIN 標準の適用可能なコントロール AWS CloudHSM を満たしていることによって生成されます。責任概要マトリックスには、 AWS CloudHSM とその顧客のそれぞれの責任であるコントロールが記載されています。

Q: コンプライアンス AWS CloudHSM 証明書を取得するにはどうすればよいですか？

PCI-PIN Attestation of Compliance (AOC) は、コンプライアンスレポートへのオンデマンドアクセス用のセルフサービスポータルである AWS Artifact を通じて入手できます。詳細については、AWS マネジメントコンソールで AWS Artifact にサインインするか、「AWS Artifact の開始方法」で詳細をご覧ください。

Q: 自分が担当している PCI PIN コントロールはどこで確認できますか?

詳細については、AWS AWS CloudHSM PCI PIN コンプライアンスパッケージの「PCI PIN Responsibility Summary」を参照してください。AWS のコンプライアンスレポートへのオンデマンドアクセス用のセルフサービスポータルである AWS Artifact を通じて、お客様が利用できます。詳細については、AWS マネジメントコンソールで AWS Artifact にサインインするか、「AWS Artifact の開始方法」で詳細をご覧ください。

Q: AWS CloudHSM 顧客として、PCI-PIN 準拠証明書 (AOC) を利用できますか?

PCI-PIN コンプライアンスはお客様自身で管理する必要があります。支払いワークロードがすべての PCI-PIN 管理/要件を満たしていることを確認するには、認定 PIN 評価者 (QPA) による正式な PCI-PIN 認証プロセスを経る必要があります。ただし、AWS が担当するコントロールについては、QPA AWS CloudHSM はさらにテストすることなくコンプライアンス認証 (AOC) に頼ることができます。

Q: キー管理ライフサイクルに関連する PCI-PIN 要件は AWS CloudHSM 担当していますか？

AWS CloudHSM は、HSMs。PCI-PIN 規格の主要な管理ライフサイクル要件はお客様の責任となります。

Q: PCI-PIN に準拠している AWS CloudHSM コントロールはどれですか？

AOC は、QPA によって評価される AWS CloudHSM コントロールを要約します。PCI-PIN Responsibility Summary は、コンプライアンスレポートへのオンデマンドアクセス用のセルフサービスポータルである AWS Artifact を通じて入手できます。

Q: PIN 翻訳や DUKPT などの支払い機能は AWS CloudHSM サポートされていますか？

いいえ、汎用 HSMs AWS CloudHSM を提供します。今後、支払い機能が提供される可能性はあります。サービスは支払い機能を直接実行しませんが、 AWS CloudHSM PCI PIN のコンプライアンス認証により、お客様は で実行されているサービスに対して独自の PCI コンプライアンスを取得できます AWS CloudHSM。ワークロードでの AWS Payment Cryptography サービス利用に興味がある場合は、ブログ「AWS Payment Cryptography による支払い処理のクラウドへの移行」を参照してください。

非推奨通知

FIPS 140、PCI-DSS、PCI-PIN、PCI-3DS、SOC2 の要件に準拠し続けるために、 は時折機能を非推奨に AWS CloudHSM することがあります。このページには、現在適用されている変更が一覧表示されています。

FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨

米国国立標準技術研究所 (NIST) [1] は、トリプル DES (DESede、3DES、DES3) 暗号化と PKCS #1 v1.5 パディングによる RSA キーのラップとアンラップのサポートは 2023 年 12 月 31 日以降は許可されないよう勧告しています。したがって、これらのサポートは、連邦情報処理標準 (FIPS) モードクラスターで 2024 年 1 月 1 日に終了します。これらに対するサポートは、非 FIPs。

このガイダンスは、以下の暗号化オペレーションに適用されます。

- トリプル DES キー生成
  - PKCS #11 ライブラリ向け `CKM_DES3_KEY_GEN`
  - JCE プロバイダー向け `DESede` Keygen
  - `genSymKey`と KMU 向け `-t=21`
- トリプル DES キーによる暗号化 (注: 復号化操作は許可されています)
  - PKCS #11 ライブラリの場合: `CKM_DES3_CBC` 暗号化、`CKM_DES3_CBC_PAD` 暗号化、`CKM_DES3_ECB` 暗号化
  - JCE プロバイダーの場合: `DESede/CBC/PKCS5Padding` 暗号化、`DESede/CBC/NoPadding` 暗号化、`DESede/ECB/Padding` 暗号化、`DESede/ECB/NoPadding` 暗号化
- PKCS #1 v1.5 パディングによる RSA キーのラップ、アンラップ、暗号化、および復号化
  - PKCS #11 SDK 向け `CKM_RSA_PKCS` ラップ、アンラップ、暗号化、および復号化
  - JCE SDK 向け `RSA/ECB/PKCS1Padding` ラップ、アンラップ、暗号化、および復号化
  - KMU 向け `-m 12` 付きの `wrapKey`と`unWrapKey` (注記12はメカニズム `RSA_PKCS` の値)

[1] この変更の詳細については、「暗号アルゴリズムとキー長の利用の変遷」の表 1 と表 5 を参照してください。

の耐障害性 AWS CloudHSM

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長な

ネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。復元性をサポートする AWS CloudHSM 機能の詳細については、「クラスターの高可用性とロードバランシング」を参照してください。

のインフラストラクチャセキュリティ AWS CloudHSM

マネージドサービスである AWS CloudHSM は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

が AWS 公開している API コールを使用して、ネットワーク AWS CloudHSM 経由で にアクセスします。また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

ネットワークの隔離

Virtual Private Cloud (VPC) は、AWS クラウド内の論理的に隔離された領域にある仮想ネットワークです。VPC のプライベートサブネットにクラスターを作成できます。VPC を作成するときにプライベートサブネットを作成できます。詳細については、「仮想プライベートクラウド (VPC) の作成」を参照してください。

HSM を作成するときは、HSM とやり取りできるように、サブネットに Elastic Network Interface (ENI) HSMs AWS CloudHSM を入力します。詳細については、「クラスターアーキテクチャ」を参照してください。

AWS CloudHSM は、クラスター内の HSMs 間のインバウンド通信とアウトバウンド通信を許可するセキュリティグループを作成します。このセキュリティグループを使用して、EC2 インスタンスがクラスター内の HSM と通信するようにできます。詳細については、「クライアントの Amazon EC2 インスタンスのセキュリティグループを設定する」を参照してください。

ユーザーの承認

では AWS CloudHSM、HSM で実行されるオペレーションには、認証された HSM ユーザーの認証情報が必要です。詳細については、「the section called “HSM ユーザーを理解する”」を参照してください。

AWS CloudHSM および VPC エンドポイント

インターフェイス VPC エンドポイント を作成 AWS CloudHSM することで、VPC と の間にプライベート接続を確立できます。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで AWS CloudHSM APIs にプライベートにアクセスできるテクノロジーである を利用しています。VPC 内のインスタンスは、パブリック IP アドレスがなくても AWS CloudHSM APIs。VPC と AWS CloudHSM 間のトラフィックは、Amazon ネットワークを離れません。

各インターフェイスエンドポイントは、サブネット内にある 1 つ、または複数の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink) 」を参照してください。

AWS CloudHSM VPC エンドポイントに関する考慮事項

のインターフェイス VPC エンドポイントを設定する前に AWS CloudHSM、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントのプロパティと制限」を確認してください。

- AWS CloudHSM は、VPC からのすべての API アクションの呼び出しをサポートします。

AWS CloudHSMのインターフェイス VPC エンドポイントの作成

Amazon VPC コンソールまたは AWS Command Line Interface () を使用して、 AWS CloudHSM サービスの VPC エンドポイントを作成できますAWS CLI。詳細については、 Amazon VPC ユーザーガイド のインターフェイスエンドポイントの作成を参照してください。

の VPC エンドポイントを作成するには AWS CloudHSM、次のサービス名を使用します。

```
com.amazonaws.region.cloudhsmv2
```

例えば、米国西部 (オレゴン) リージョン (us-west-2) では、サービス名は次のようになります。

```
com.amazonaws.us-west-2.cloudhsmv2
```

VPC エンドポイントを使いやすくするために、VPC エンドポイントの プライベート DNS ホスト名 を有効にできます。プライベート DNS 名を有効にする オプションを選択すると、標準の AWS CloudHSM DNS ホスト名 (`https://cloudhsmv2.<region>.amazonaws.com`) が VPC エンドポイントに解決されます。

このオプションにより VPC エンドポイントが使いやすくなります。 AWS SDKs と はデフォルトで標準の AWS CloudHSM DNS ホスト名 AWS CLI を使用するため、アプリケーションとコマンドで VPC エンドポイント URL を指定する必要はありません。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

の VPC エンドポイントポリシーの作成 AWS CloudHSM

VPC エンドポイントには、 AWS CloudHSMへのアクセスを制御するエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: AWS CloudHSM アクションの VPC エンドポイントポリシー

のエンドポイントポリシーの例を次に示します AWS CloudHSM。このポリシーは、エンドポイントにアタッチされると、すべてのリソースのすべてのプリンシパルに対して、リストされている AWS CloudHSM アクションへのアクセスを許可します。その他の AWS CloudHSM アクションとそれに対応する IAM アクセス許可の Identity and Access Management AWS CloudHSMについては、「」を参照してください。

```
{
   "Statement":[
      {
         "Principal":"*",
```

```
            "Effect":"Allow",
            "Action":[
               "cloudhsm:DescribeBackups",
               "cloudhsm:DescribeClusters",
               "cloudhsm:ListTags",
            ],
            "Resource":"*"
         }
   ]
}
```

での更新管理 AWS CloudHSM

AWS がファームウェアを管理します。ファームウェアはサードパーティーによってメンテナンスされます。また、FIPS 140-2 レベル 3 コンプライアンスを満たしているかどうかについて NIST の評価を受ける必要があります。インストールできるのは、FIPS キーによって暗号化された署名済みのファームウェアのみです (AWS にはこのキーへのアクセス権がありません)。

トラブルシューティング AWS CloudHSM

で問題が発生した場合は AWS CloudHSM、以下のトピックが解決に役立ちます。

トピック

既知の問題

AWS CloudHSM には以下の既知の問題があります。詳細情報についてはトピックを選択してください。

トピック

すべての HSM インスタンスの既知の問題

以下の問題は、key_mgmt_util コマンドラインツール、PKCS #11 SDK、JCE SDK、OpenSSL SDK のいずれを使用しているかにかかわらず、すべての AWS CloudHSM ユーザーに影響します。

トピック

[問題]: AESキーラッピングでは、ゼロパディング付きのキーラップのスタンダード準拠の実装を提供する代わりに、PKCS#5 パディングを使用します

さらに、パディングなしおよびゼロパディングありのキーラップはサポートされていません。

- 影響: 内でこのアルゴリズムを使用してラップおよびアンラップしても、影響はありません AWS CloudHSM。ただし、 でラップされたキーは、ノーパディング仕様への準拠が予想される他の HSMs またはソフトウェア内でラップ解除 AWS CloudHSM することはできません。これは、標準に準拠したラップ解除中に、8 バイトのパディングデータがキーデータの最後に追加される可能性

があるためです。外部でラップされたキーを AWS CloudHSM インスタンスに適切にラップ解除することはできません。

- 回避策: AWS CloudHSM インスタンスで PKCS #5 パディングありの AES キーラップを使用してラップされたキーを外部でラップ解除するには、キーを使用する前に余分なパディングを省きます。これを行うには、ファイルエディターで余分なバイトをトリミングするか、コード内の新しいバッファにキーバイトだけをコピーします。
- 解決策のステータス: 3.1.0 クライアントおよびソフトウェアリリースで、 AWS CloudHSM に AES キーラップの標準に準拠したオプションが用意されています。詳細については、「AES キーラップ」を参照してください。

問題: クライアントデーモンがクラスターに正常に接続には、その設定ファイル少なくとも 1 つの有効な IP アドレスが必要です

- 影響: クラスター内のすべての HSM を削除し、新しい IP アドレスを取得する別の HSM を追加した場合、クライアントデーモンは元の IP アドレスで HSM を検索し続けます。
- 回避策: 断続的なワークロードを実行する場合は、 CreateHsm関数の `IpAddress`引数を使用して、Elastic Network Interface (ENI) を元の値に設定することをお勧めします。ENI はアベイラビリティーゾーン (AZ) に固有である点に注意してください。代わりに、`/opt/cloudhsm/daemon/1/cluster.info` ファイルを削除した後、新しい HSM クライアントの IP アドレスにクライアント設定をリセットできます。`client -a <IP address>` コマンドを使用できます。詳細については、「クライアントのインストールと設定 AWS CloudHSM (Linux)」またはAWS CloudHSM 「クライアントのインストールと設定 (Windows)」を参照してください。

問題: Client SDK 3 AWS CloudHSM を使用してハッシュ化および署名できるデータには 16 KB の上限がありました

- 解決策のステータス: サイズが 16 KB 未満のデータは、ハッシュ用に引き続き HSM に送信されます。16 ~ 64 KB のサイズのデータをローカルやソフトウェアでハッシュする機能が追加されました。データバッファが 64KB を超える場合、クライアント SDK 5 は明示的に失敗します。修正を利用するには、クライアントと SDK (5.0.0 以降のバージョン) を更新する必要があります。

問題: インポートされたキーをエクスポート不可として指定できませんでした

- 解決策のステータス: この問題は修正されています。修正を反映させるためにお客様側で必要なアクションはありません。

問題: key_mgmt_util の wrapKey コマンドと unWrapKey コマンドのデフォルトのメカニズムが削除されました

- 解決策: wrapKey または unWrapKey コマンドを使用する場合は、 -mオプションを使用してメカニズムを指定する必要があります。詳細については、 wrapKey または unWrapKey 記事の例を参照してください。

問題: クラスターに HSM が 1 つしかない場合、HSM フェイルオーバーが正しく動作しません

- 影響: クラスター内に 1 つしかない HSM インスタンスの接続が失われると、後で回復しても、クライアントはインスタンスに再接続しません。
- 回避方法: 本番稼働用クラスターに少なくとも 2 つの HSM インスタンスを用意することをお勧めします。この構成を使用すれば、この問題の影響を受けません。HSM が 1 つしかないクラスターの場合、クライアントデーモンをバウンスして接続を復元します。
- 解決策のステータス: この問題は、 AWS CloudHSM クライアント 1.1.2 のリリースで解決されています。修正のメリットを享受するには、このクライアントにアップグレードする必要があります。

問題: クラスター内の HSM のキー容量を短期間で超えた場合、クライアントが処理されないエラー状態に陥ります

- 影響: クライアントが処理されないエラー状態になると、フリーズし、再起動が必要になります。
- 回避方法: スループットをテストして、クライアントが処理できない速さでセッションキーを作成していないか、確認します。速さを落とすには、クラスターに HSM を追加するか、セッションキーの作成を遅くします。
- 解決策のステータス: この問題は、 AWS CloudHSM クライアント 1.1.2 のリリースで解決されています。修正のメリットを享受するには、このクライアントにアップグレードする必要があります。

問題: 800 バイトを超える HMAC キーを使ったダイジェストオペレーションはサポートされていません

- 影響: 800 バイトを上回る HMAC キーが HSM で生成されたり、HSM にインポートされたりする可能性があります。ただし、このような大きなキーを JCE または key_mgmt_util を介してダイ

ジェストオペレーションに使用すると、オペレーションが失敗します。PKCS11 を使用している場合、HMAC キーのサイズは 64 バイトに制限されます。

- 回避方法: HSM のダイジェストオペレーションに HMAC キーを使用する場合は、必ずサイズが 800 バイト以下のものを使用します。
- 解決策のステータス: 現時点ではありません。

問題：クライアント SDK 3 で配布された client_info ツールが、オプションの出力引数で指定されたパスの内容を削除します

- 影響: 指定した出力パスの下にある既存のファイルとサブディレクトリはすべて、永久に失われる可能性があります
- 防止策: `-output` *`path`* ツールを使用する際、オプションの引数 `client_info` を使用しないでください。
- 解決策の現状: この問題は、クライアント SDK 3.3.2 のリリース によって解決されています。修正のメリットを享受するには、このクライアントにアップグレードする必要があります。

問題: コンテナ化された環境で **`--cluster-id`** 引数を使用して SDK 5 設定ツールを実行すると、エラーが表示されます

設定ツールで--cluster-id 引数を使用すると、次のエラーが表示されます。

```
No credentials in the property bag
```

このエラーは、インスタンスメタデータサービスのバージョン 2 (IMDSv2) の更新が原因で発生します。詳細については、「IMDSv2」のドキュメントを参照してください。

- 影響: この問題は、コンテナ化された環境で SDK バージョン 5.5.0 以降の設定ツールを実行し、EC2 インスタンスメタデータを利用して認証情報を提供するユーザーに影響を及ぼします。
- 回避方法: PUT レスポンスホップ制限を少なくとも 2 に設定します。その方法のガイダンスについては、「インスタンスメタデータオプションの設定」を参照してください。

問題: 「提供された pfx ファイルから証明書/キーを作成できませんでした。エラー：NotPkcs8"

- 影響: 証明書とプライベートキーを使用して SSL を再設定する SDK 5.11.0 ユーザーは、プライベートキーが PKCS8 形式でない場合、失敗します。

- 回避策: openssl コマンドを使用して、カスタム SSL プライベートキーを PKCS8 形式に変換できます。`openssl pkcs8 -topk8 -inform PEM -outform PEM -in ssl_private_key -out ssl_private_key_pkcs8`
- 解決ステータス: この問題は、クライアント SDK 5.12.0 リリース で解決されています。修正を利用するには、このクライアントバージョン 以降にアップグレードする必要があります。

hsm2m.medium インスタンスの既知の問題

以下の問題は、すべての hsm2m.medium インスタンスに影響します。

トピック

- 問題: PBKDF2 の反復回数の増加によるログインレイテンシーの増加
- 問題: キーの信頼された属性を設定しようとする を使用する CO は、クライアント SDK 5.12.0 以前では失敗する

問題: PBKDF2 の反復回数の増加によるログインレイテンシーの増加

- 影響: セキュリティを強化するために、hsm2m.medium はログインリクエスト中にパスワードベースのキー導出関数 2 (PBKDF2) を 60,000 回繰り返し実行しますが、hsm1.medium では 1,000 回実行します。この増加により、ログインリクエストごとに最大 2 秒 (2 秒) のレイテンシーが増加する可能性があります。

  AWS CloudHSM Client SDKs のデフォルトのタイムアウトは 20 秒です。ログインリクエストがタイムアウトし、エラーが発生する可能性があります。
- 回避策: 可能であれば、ログイン中のレイテンシーが長くなるのを避けるため、同じアプリケーションでログインリクエストをシリアル化します。
- 解決ステータス: Client SDK の将来のバージョンでは、このレイテンシーの増加を考慮してログインリクエストのデフォルトタイムアウトが増加します。

問題: キーの信頼された属性を設定しようとする を使用する CO は、クライアント SDK 5.12.0 以前では失敗する

- 影響: キーの信頼された属性を設定しようとする CO ユーザーは、 を示すエラーを受け取りますUser type should be CO or CU。

- 解決策: Client SDK の将来のバージョンでは、この問題を解決します。更新はユーザーガイドの で迷惑になりますドキュメント履歴。

PKCS#11 ライブラリの既知の問題

トピック

- 問題: PKCS #11 ライブラリのバージョン 3.0.0 での AES キーラップが、使用前に IV を検証しません
- 問題: PKCS #11 SDK 2.0.4 以前のバージョンでは、AES キーのラップとラップ解除に常に 0xA6A6A6A6A6A6A6A6 のデフォルトの IV が使用されていました。
- 問題: CKA_DERIVE 属性はサポート外だったため、処理されませんでした
- 問題: CKA_SENSITIVE 属性はサポート外だったため、処理されませんでした
- 問題: マルチパートのハッシュおよび署名がサポートされていません
- 問題: C_GenerateKeyPair は、プライベートテンプレートで標準に従った方法では、CKA_MODULUS_BITS や CKA_PUBLIC_EXPONENT を処理しません
- 問題: C_Encrypt メカニズムを使用している場合、 C_Decrypt および CKM_AES_GCM API オペレーションのバッファが 16 KB を超えることができません
- 問題: 楕円曲線ディフィーヘルマン (ECDH) キーの導出が、HSM 内で部分的に実行されます
- 問題: CentOS6 や RHEL 6 などの EL6 プラットフォームで secp256k1 署名の検証が失敗する
- 問題 : 関数呼び出しの順序が正しくないと、失敗する代わりに未定義の結果が得られる。
- 問題 : SDK 5 では読み取り専用セッションは対応していません
- 問題: cryptoki.hヘッダーファイルは Windows 専用です

問題: PKCS #11 ライブラリのバージョン 3.0.0 での AES キーラップが、使用前に IV を検証しません

長さが 8 バイトより短い IV を指定すると、使用前に予測不可能なバイトが埋め込まれます。

> **Note**
>
> これは `CKM_AES_KEY_WRAP` メカニズムがある `C_WrapKey` にのみ影響します。

- 影響: PKCS #11 バージョン 3.0.0 で 8 バイトより短い IV を指定した場合、キーをラップ解除できない可能性があります。
- 回避方法:
  - PKCS #11 バージョン 3.0.1 以降にアップグレードすることを強くお勧めします。これにより、AES キーラップ時に IV の長さが適切に適用されます。ラップコードを修正して NULL IV を渡すか、0xA6A6A6A6A6A6A6A6 のデフォルトの IV を指定します。詳細情報は、「トラブルシューティングガイド : AESキーラップ用非対応長さのCustom IV」を参照してください。
  - 8 バイトより短い IV を使用して PKCS #11 ba-jon 3.0.0 でキーをラップした場合は、弊社 サポートデスク へご連絡ください。
- 解決策のステータス: この問題は PKCS #11 SDK バージョン 3.0.1 で解決されています。AES キーラップを使用してキーをラップするには、NULL または 8 バイトの長さの IV を指定します。

問題: PKCS #11 SDK 2.0.4 以前のバージョンでは、AES キーのラップとラップ解除に常に **0xA6A6A6A6A6A6A6A6** のデフォルトの IV が使用されていました。

ユーザーが指定した IV はそのまま無視されていました。

> **Note**
>
> これは CKM_AES_KEY_WRAP メカニズムがある C_WrapKey にのみ影響します。

- 影響:
  - PKCS #11 SDK 2.0.4 以前のバージョンとユーザーが指定した IV を使用した場合、キーは 0xA6A6A6A6A6A6A6A6 のデフォルトの IV でラップされます。
  - PKCS #11 SDK 3.0.0 以降とユーザーが指定した IV を使用した場合、キーはユーザーが指定した IV でラップされます。
- 回避方法:
  - PKCS #11 SDK 2.0.4 以前でラップされたキーをラップ解除するには、0xA6A6A6A6A6A6A6A6 のデフォルトの IV を使用します。
  - PKCS #11 SDK 3.0.0 以降でラップされたキーをラップ解除するには、ユーザーが指定した IV を使用します。
- 解決策のステータス: ラップおよびラップ解除コードを修正して NULL IV を渡すか、0xA6A6A6A6A6A6A6A6 のデフォルトの IV を指定することを強くお勧めします。

問題: **CKA_DERIVE** 属性はサポート外だったため、処理されませんでした

- 解決策のステータス: FALSE が設定されている場合は、CKA_DERIVE を受け付けるように修正を実装し ます。 AWS CloudHSMにキー取得関数サポートが追加されるまで、TRUE に設定された CKA_DERIVE はサポートされません。この修正を適用するには、クライアントと SDK をバージョン 1.1.1 以上に更新する必要があります。

問題: **CKA_SENSITIVE** 属性はサポート外だったため、処理されませんでした

- 解決策のステータス: CKA_SENSITIVE 属性を受け入れ、適切に遵守するように修正を実装しました。この修正を適用するには、クライアントと SDK をバージョン 1.1.1 以上に更新する必要があります。

問題: マルチパートのハッシュおよび署名がサポートされていません

- 影響: C_DigestUpdate および C_DigestFinal は実装されません。C_SignFinal も実装されていないため、NULL 以外のバッファでは CKR_ARGUMENTS_BAD でエラーが発生します。
- 回避策: アプリケーション内でデータをハッシュし、ハッシュの署名 AWS CloudHSM にのみ使用します。
- 解決策のステータス: クライアントと SDK を修正し、マルチパートハッシュを正しく実装する予定です。更新は AWS CloudHSM フォーラムとバージョン履歴ページで告知されます。

問題: **C_GenerateKeyPair** は、プライベートテンプレートで標準に従った方法では、**CKA_MODULUS_BITS** や **CKA_PUBLIC_EXPONENT** を処理しません

- 影響: プライベートテンプレートに または C_GenerateKeyPair が含まれている場合、CKA_TEMPLATE_INCONSISTENTCKA_MODULUS_BITS は CKA_PUBLIC_EXPONENT を返します。代わりに、すべてのフィールドが FALSE に設定されているプライベートキーを生成します。キーは使用できません。
- 解決策: アプリケーションによって、エラーコードに加えて使用状況フィールドの値をチェックすることをお勧めします。
- 解決策のステータス: 修正を実装し、間違ったプライベートキーテンプレートが使用されている場合に適切なエラーメッセージを返すようにします。更新された PKCS#11 ライブラリは、バージョン履歴ページで告知されます。

問題: **C_Encrypt** メカニズムを使用している場合、 **C_Decrypt** および **CKM_AES_GCM** API オペレーションのバッファが 16 KB を超えることができません

AWS CloudHSM は、マルチパート AES-GCM 暗号化をサポートしていません。

- 影響: CKM_AES_GCM メカニズムを使用して 16 KB を超えるデータを暗号化することができません。
- 回避策: CKM_AES_CBC、CKM_AES_CBC_PAD などの代替メカニズムを使用するか、データを複数部分に分割し、AES_GCM を使用して各部分を個別に暗号化できます。を使用している場合はAES_GCM、データの分割とその後の暗号化を管理する必要があります。 AWS CloudHSM はマルチパート AES-GCM 暗号化を実行しません。FIPS では、AES-GCM の初期化ベクター (IV) を HSM で生成する必要があります。したがって、AES-GCM 暗号化データの各部分の IV は異なります。
- 解決策のステータス: SDK を修正し、データバッファが大きすぎる場合は明示的に失敗するようにします。C_EncryptUpdate および C_DecryptUpdate API オペレーションに CKR_MECHANISM_INVALID を返します。マルチパート暗号化を使用しなくても大きいバッファをサポートできる代替方法を評価しています。更新は、 AWS CloudHSM フォーラムとバージョン履歴ページで発表されます。

問題: 楕円曲線ディフィーヘルマン (ECDH) キーの導出が、HSM 内で部分的に実行されます

EC プライベートキーは常に HSM にありますが、キーの取得手順は複数のステップで実行されます。その結果、各ステップの中間結果がクライアントに存在します。

- 影響: Client SDK 3 では、 CKM_ECDH1_DERIVEメカニズムを使用して派生したキーが最初にクライアントで使用でき、次に HSM にインポートされます。その後、キーのハンドルがアプリケーションに返されます。
- 回避策: AWS CloudHSMに SSL/TLS のオフロードを実装すると、この制限が問題とはならない場合があります。アプリケーションでキーが常に FIPS 境界内に留まる必要がある場合、ECDH キー取得に依存しない代替プロトコルの使用を検討してください。
- 解決策のステータス: ECDH キー取得を完全に HSM 内部で実行するオプションを開発中です。更新の実装は、利用可能になり次第バージョン履歴ページで告知されます。

問題: CentOS6 や RHEL 6 などの EL6 プラットフォームで secp256k1 署名の検証が失敗する

これは、CloudHSM PKCS#11 ライブラリが、OpenSSL を使用して EC 曲線データを検証することにより、検証操作の初期化中にネットワークの呼び出しを回避するために発生します。Secp256k1 は EL6 プラットフォームのデフォルトの OpenSSL パッケージでサポートされていないため、初期化は失敗します。

- 影響: Secp256k1 署名検証が EL6 プラットフォームで失敗します。検証呼び出しは、`CKR_HOST_MEMORY` エラーで失敗します。
- 回避策: PKCS#11 アプリケーションで secp256k1 の署名を検証する必要がある場合は、Amazon Linux 1 または任意の EL7 プラットフォームを使用することをお勧めします。または、secp256k1 曲線 をサポートする OpenSSL パッケージのバージョンにアップグレードすることもできます。
- 解決策のステータス: ローカル曲線検証が利用できない場合に HSM にフォールバックするための修正を実装中です。更新された PKCS#11 ライブラリは、バージョン履歴ページで告知されます。

問題 : 関数呼び出しの順序が正しくないと、失敗する代わりに未定義の結果が得られる。

- 影響 : 誤った一連の関数を呼び出すと、個々の関数呼び出しの返しが成功しても、最終的な結果は正しくありません。例えば、復号化されたデータが元のプレーンテキストと一致しない場合や、署名が検証できない場合があります。この問題は、シングルパートとマルチパートの両方のオペレーションに影響します。

  誤った関数シーケンスの例 :

  - `C_EncryptInit` / `C_EncryptUpdate` の後に `C_Encrypt` が続きます
  - `C_DecryptInit` / `C_DecryptUpdate` の後に `C_Decrypt` が続きます
  - `C_SignInit` / `C_SignUpdate` の後に `C_Sign` が続きます
  - `C_VerifyInit` / `C_VerifyUpdate` の後に `C_Verify` が続きます
  - `C_FindObjectsInit` は `C_FindObjectsInit` の後に続きます
- 防止策: アプリケーションを PKCS #11 仕様に準拠して、シングルパートとマルチパートの両方のオペレーションに適切な関数呼び出しを使用する必要があります。この状況では、アプリケーションが CloudHSM PKCS #11 ライブラリに依存して エラーを返す必要はありません。

問題 : SDK 5 では読み取り専用セッションは対応していません

- 問題: SDK 5 では、C_OpenSession で読み取り専用セッションを開くことはできません。
- 影響: C_OpenSession 未対応で CKF_RW_SESSION を呼び出ししようとした場合、呼び出しは失敗し、エラー CKR_FUNCTION_FAILED が返されます。
- 防止策: セッションを開く際、CKF_SERIAL_SESSION | CKF_RW_SESSION 関数呼び出しに C_OpenSession フラグを渡す必要があります。

問題: **cryptoki.h**ヘッダーファイルは Windows 専用です

- 問題: Linux の AWS CloudHSM Client SDK 5 バージョン 5.0.0 から 5.4.0 では、 ヘッダーファイルは Windows オペレーティングシステムとのみ互換性/opt/cloudhsm/include/pkcs11/cryptoki.hがあります。
- 影響: Linux ベースのオペレーティングシステム上のアプリケーションにこのヘッダーファイルを含めようとすると、問題が発生する可能性があります。
- 解決ステータス: このヘッダーファイルの Linux 互換バージョンを含む AWS CloudHSM Client SDK 5 バージョン 5.4.1 以降にアップグレードします。

JCE SDK の既知の問題

トピック

- 問題: 非対称キーペアを使用する場合、明示的にキーを作成またはインポートしない場合でも、キー容量が占有されます。
- 問題: JCE KeyStore は読み取り専用です
- 問題: AES-GCM 暗号化のバッファが 16,000 バイトを超えることはできません
- 問題: 楕円曲線ディフィーヘルマン (ECDH) キーの導出が、HSM 内で部分的に実行されます
- 問題: キーサイズパラメータをビット数ではなくバイト数として KeyGenerator KeyAttribute 誤って解釈する
- 問題: Client SDK 5 から「不正な反射アクセスオペレーションが発生しました」という警告が表示されます
- 問題: JCE セッションプールが使い果たされています
- 問題: getKey オペレーションによるクライアント SDK 5 メモリリーク

問題: 非対称キーペアを使用する場合、明示的にキーを作成またはインポートしない場合でも、キー容量が占有されます。

- 影響: この問題により、HSM が予期せずキー領域を使い果たすことがあります。この問題は、アプリケーションが暗号化オペレーションに CaviumKey オブジェクトではなく標準の JCE キーオブジェクトを使用する場合に発生します。標準の JCE キーオブジェクトを使用する場合、CaviumProvider によって暗黙的にそのキーが HSM にインポートされ、アプリケーションが終了するまでセッションキーによってこのキーは削除されません。その結果、アプリケーションの実行中にキーが蓄積され、HSM の空きキー領域が不足して、アプリケーションがフリーズする可能性があります。
- 回避策: CaviumSignature クラス、CaviumCipher クラス、CaviumMac クラス、または CaviumKeyAgreement クラスを使用する場合は、標準の JCE キーオブジェクトではなく CaviumKey としてキーを指定してください。

  CaviumKey クラスを使用して通常のキーを ImportKey に手動で変更し、オペレーションの完了後にキーを手動で削除できます。
- 解決策のステータス: 暗黙的なインポートを適切に管理するために、CaviumProvider を更新中です。修正は、利用可能なバージョン履歴ページで告知されます。

問題: JCE KeyStore は読み取り専用です

- 影響: 現在、HSM でサポートされていないオブジェクトタイプを JCE KeyStore に保存することはできません。具体的には、キーストアに証明書を保存することはできません。これにより、キーストア内で証明書を見つけることを期待する jarsigner などのツールとの相互運用性が妨げられます。
- [Workaround(回避策)]: キーストアではなく、ローカルファイルまたは S3 バケットの場所から証明書をロードするようにコードを修正することができます。
- [Resolution status(解決策のステータス)]: キーストアに証明書ストレージのサポートを追加しています。機能は、利用可能なバージョン履歴ページで告知されます。

問題: AES-GCM 暗号化のバッファが 16,000 バイトを超えることはできません

マルチパート AES-GCM 暗号化は対応していません。

- 影響 : AES-GCM を使用して 16,000 バイトを超えるデータを暗号化することができません。

- 回避策: AES-CBC などの代替メカニズムを使用するか、データを複数部分に分割して各部分を別々に暗号化できます。データを分割する場合、分割された暗号化テキストとその復号化を管理する必要があります。FIPS では AES-GCM の初期化ベクター (IV) を HSM で生成する必要があるため、AES-GCM で暗号化された部分的データはそれぞれ IV が異なります。
- 解決策のステータス: SDK を修正し、データバッファが大きすぎる場合は明示的に失敗するようにします。マルチパート暗号化を使用しなくても大きいバッファをサポートできる代替方法を評価しています。更新は AWS CloudHSM フォーラムとバージョン履歴ページで告知されます。

問題: 楕円曲線ディフィーヘルマン (ECDH) キーの導出が、HSM 内で部分的に実行されます

EC プライベートキーは常に HSM にありますが、キーの取得手順は複数のステップで実行されます。その結果、各ステップの中間結果がクライアントに存在します。ECDH キー導出サンプルは Java コードサンプルで入手できます。

- 影響: Client SDK 3 は ECDH 機能を JCE に追加します。`KeyAgreement` クラスを使用して を取得すると SecretKey、まずクライアントで使用でき、次に HSM にインポートされます。その後、キーのハンドルがアプリケーションに返されます。
- 回避策: で SSL/TLS オフロードを実装している場合 AWS CloudHSM、この制限は問題にならない可能性があります。アプリケーションでキーが常に FIPS 境界内に留まる必要がある場合、ECDH キー取得に依存しない代替プロトコルの使用を検討してください。
- 解決策のステータス: ECDH キー取得を完全に HSM 内部で実行するオプションを開発中です。利用可能な場合は、バージョン履歴ページで更新された実装をお知らせします。

問題: キーサイズパラメータをビット数ではなくバイト数として KeyGenerator KeyAttribute 誤って解釈する

KeyGenerator クラスの `init`関数または列挙型 の `SIZE` 属性を使用してキーを生成する場合、API は 引数がキーバイト数であることを誤って想定します。代わりにキービット数である必要があります。 AWS CloudHSM KeyAttribute

- 影響: Client SDK バージョン 5.4.0 ~ 5.4.2 では、指定した API にキーサイズがバイトとして提供されることを誤って想定しています。
- 回避策: Client SDK バージョン 5.4.0 から 5.4.2 を使用している場合、 KeyGenerator クラスまたは KeyAttribute 列挙型を使用して AWS CloudHSM JCE プロバイダーを使用してキーを生成する前に、キーサイズをビットからバイトに変換します。

- 解決ステータス: クライアント SDK バージョンを 5.5.0 以降にアップグレードします。これには、KeyGenerator クラスまたは KeyAttribute 列挙型を使用してキーを生成するときに、ビット単位でキーサイズを正しく想定する修正が含まれています。

問題: Client SDK 5 から「不正な反射アクセスオペレーションが発生しました」という警告が表示されます

Java 11 で Client SDK 5 を使用すると、CloudHSM から次の Java 警告が表示されます。

```
```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by
 com.amazonaws.cloudhsm.jce.provider.CloudHsmKeyStore (file:/opt/cloudhsm/java/
cloudhsm-jce-5.6.0.jar) to field java.security .KeyStore.keyStoreSpi
WARNING: Please consider reporting this to the maintainers of
 com.amazonaws.cloudhsm.jce.provider.CloudHsmKeyStore
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective
 access operations
WARNING: All illegal access operations will be denied in a future release
```
```

これらの警告は影響しません。この問題は認識済みであり、現在、問題の解決に取り組んでいます。解決策も回避策も必要ありません。

問題: JCE セッションプールが使い果たされています

影響: 次のメッセージが表示されると、JCE で操作を実行できなくなる可能性があります。

```
com.amazonaws.cloudhsm.jce.jni.exception.InternalException: There are too many
 operations
happening at the same time: Reached max number of sessions in session pool: 1000
```

回避方法:

- 影響が出ている場合は、JCE アプリケーションを再起動してください。
- オペレーションを実行する場合、オペレーションへの参照が失われる前に JCE オペレーションを完了する必要がある場合があります。

> **Note**
>
> オペレーションによっては、完了方法が必要な場合があります。

操作	完了方法
暗号	暗号化モードまたは復号モードの `doFinal()` ラップモードの `wrap()` アンラップモードの `unwrap()`
KeyAgreement	`generateSecret()` または `generateSecret(String)`
KeyPairGenerator	`generateKeyPair()` 、`genKeyPair()` 、または `reset()`
KeyStore	メソッド不要
MAC	`doFinal()` または `reset()`
MessageDigest	`digest()` または `reset()`
SecretKeyFactory	メソッド不要
SecureRandom	メソッド不要
署名	サインモードの `sign()` 検証モードの `verify()`

解決策のステータス: この問題は、Client SDK 5.9.0 以降で解決に向けて積極的に取り組んでいます。この問題を解決するには、Client SDK を以下のバージョンのいずれかにアップグレードしてください。

問題: getKey オペレーションによるクライアント SDK 5 メモリリーク

- 影響: API `getKey`オペレーションでは、クライアント SDK バージョン 5.10.0 以前の JCE でメモリリークが発生しています。アプリケーションで `getKey` API を複数回使用すると、メモリが増加し、その結果、アプリケーションのメモリフットプリントが増加します。時間が経つと、スロットリングエラーが発生したり、アプリケーションを再起動する必要がある場合があります。
- 回避策: Client SDK 5.11.0 にアップグレードすることをお勧めします。これを実行できない場合は、アプリケーションで `getKey` API を複数回呼び出さないことをお勧めします。代わりに、前の`getKey`オペレーションから以前に返されたキーをできるだけ再利用します。
- 解決ステータス: クライアント SDK バージョンを 5.11.0 以降にアップグレードします。これには、この問題の修正が含まれています。

OpenSSL Dynamic Engine SDK の既知の問題

OpenSSL 動的エンジン SDK の既知の問題があります

トピック

問題: AWS CloudHSM OpenSSL Dynamic Engine を RHEL 6 および CentOS6 にインストールできない

- 影響: OpenSSL Dynamic Engineは OpenSSL 1.0.2[f+] のみをサポートしています。デフォルトでは、RHEL 6 と CentOS 6 には OpenSSL 1.0.1 が付属します。

- 回避方法: RHEL 6 および CentOS 6 の OpenSSL ライブラリをバージョン 1.0.2[f+] にアップグレードします。

[問題] デフォルトでは、HSM への RSA オフロードのみがサポートされています

- [影響]: パフォーマンスを最大限に高めるために、SDK は乱数生成や EC-DH オペレーションなどの追加機能をオフロードするように構成されていません。
- [Workaround (回避策)]: 追加のオペレーションをオフロードする必要がある場合は、サポートケースを通じてお問い合わせください。
- [Resolution status (解決策のステータス)]: オフロードオプションを設定ファイルで設定するための SDK へのサポートを追加しています。更新は、利用可能なバージョン履歴ページで告知されます。

問題: HSM でキーを使用した OAEP パディングによる RSA 暗号化および復号化がサポートされていません。

- 影響: OAEP パディングを使用した RSA 暗号化および復号化の呼び出しは、 divide-by-zero エラーで失敗します。これは、OpenSSL 動的エンジンがオペレーションを HSM にオフロードせずにフェイク PEM ファイルを使用してオペレーションをローカルで呼び出すために発生します。
- 解決策: PKCS #11 ライブラリ または JCE プロバイダー を使用してこの手順を実行できます。
- 解決策のステータス: このオペレーションを正しくオフロードする SDK のサポートを追加する予定です。更新は、利用可能なバージョン履歴ページで告知されます。

[Issue: (問題)] RSA のプライベートキー世代および ECC キーのみが HSM にオフロードされます。

その他のキータイプでは、OpenSSL AWS CloudHSM エンジンは通話処理には使用されません。代わりに、ローカルの OpenSSL エンジンが使用されます。これによって、ソフトウェアでローカルにキーが生成されます。

- [影響 :] フェイルオーバーがサイレントのため、HSM で安全に生成されたキーを受信していないことが確認できません。キーがソフトウェアで OpenSSL によってローカルで生成された場合、文字列 "............++++++" を含む出力トレースが表示されます。オペレーションが HSM にオフロードされた場合には、このトレースは存在しません。キーが生成されていない、あるいは HSM に保存されていないため、キーを今後使用することはできません。]

- [Workaround: (回避方法)] OpenSSL エンジンがサポートするキータイプのみを使用します。他のすべてのキータイプでは、アプリケーションで PKCS #11 または JCE を使用するか、 CLI `key_mgmt_util`で を使用します。

問題: RHEL 8、CentOS 8、 Ubuntu 18.04 LTS にクライアント SDK 3 用の OpenSSL Dynamic Engine をインストールできない

- 影響: デフォルトでは、RHEL 8、CentOS 8、Ubuntu 18.04 LTSは、Client SDK 3 用 OpenSSL Dynamic Engine と互換性がないバージョンを出荷しています。
- 防止策: OpenSSL 動的エンジン 対応の Linux プラットフォームを使用してください。対応プラットフォームの詳細は、対応プラットフォーム を参照してください。
- 解決ステータス: AWS CloudHSM は、クライアント SDK 5 用の OpenSSL Dynamic Engine でこれらのプラットフォームをサポートします。詳細については、対応プラットフォーム および OpenSSL Dynamic Engine を参照してください。

問題: RHEL 9 (9.2+) での SHA-1 署名と検証の非推奨化

- 影響: 暗号化目的での SHA-1 メッセージダイジェストの使用は、RHEL 9 (9.2+) では廃止されました。その結果、OpenSSL Dynamic Engine を使用した SHA-1 による署名および検証オペレーションは失敗します。
- 回避策: シナリオで、既存またはサードパーティーの暗号化署名の署名/検証に SHA-1 を使用する必要がある場合は、詳細については、「RHEL セキュリティの強化: RHEL 9 (9.2+) および RHEL 9 (9.2+) リリースノートでの SHA-1 の廃止について」を参照してください。 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/9.0_release_notes/overview

問題： AWS CloudHSM OpenSSL Dynamic Engine が OpenSSL v3.x の FIPS プロバイダーと互換性がない

- 影響: FIPS プロバイダーが AWS CloudHSM OpenSSL バージョン 3.x で有効になっているときに OpenSSL Dynamic Engine を使用しようとすると、エラーが発生します。
- 回避策: AWS CloudHSM OpenSSL バージョン 3.x で OpenSSL Dynamic Engine を使用するには、「デフォルト」プロバイダーが設定されていることを確認します。OpenSSL ウェブサイト のデフォルトプロバイダーの詳細をご覧ください。

Amazon Linux 2 を実行する Amazon EC2 インスタンスに関する既知の問題

問題: Amazon Linux 2 バージョン 2018.07 では、現在 AWS CloudHSM SDKs と互換性のない更新された**ncurses**パッケージ (バージョン 6) を使用しています

AWS CloudHSM cloudhsm_mgmt_util または key_mgmt_util を実行すると、次のエラーが返されます。

```
/opt/cloudhsm/bin/cloudhsm_mgmt_util: error while loading shared libraries:
 libncurses.so.5: cannot open shared object file: No such file or directory
```

- 影響: Amazon Linux 2 バージョン 2018.07 で実行されているインスタンスは、すべての AWS CloudHSM ユーティリティを使用することはできません。
- 防止策 : Amazon Linux 2 EC2 インスタンスで次のコマンドを発行して、対応している ncurses パッケージ (バージョン 5) をインストールします。

  ```
  sudo yum update && yum install ncurses-compat-libs
  ```

- 解決策のステータス: この問題は、 AWS CloudHSM クライアント 1.1.2 のリリースで解決されています。修正のメリットを享受するには、このクライアントにアップグレードする必要があります。

サードパーティアプリケーションの統合の既知の問題

問題: Client SDK 3 で、マスターキーの生成時に Oracle が設定する PKCS #11 属性 **CKA_MODIFIABLE** がサポートされていません

この制限は PKCS #11 ライブラリで定義されています。詳細については、「サポートされている PKCS #11 属性」の注釈 1 を参照してください。

- 影響 : Oracle マスターキーの作成に失敗する。
- 回避方法 : 新しいマスターキーを作成するときに、特別な環境変数 CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE を TRUE に設定します。この環境変数は、マスターキーの生成にのみ必要であり、この環境変数を他のものに使用する必要はありません。たとえば、作成した最初のマスターキーにこの変数を使用し、マスターキーのエディションのローテー

ションを行う場合にのみ、この環境変数を再度使用します。詳細については、「Oracle TDE マスター暗号化キーの生成」を参照してください。

- 解決策のステータス : HSM ファームウェアを改善して、CKA_MODIFABLE 属性を完全にサポートしています。更新は AWS CloudHSM フォーラムとバージョン履歴ページで発表されます。

Client SDK 3 キー同期の失敗

クライアント SDK 3 でクライアント側の同期が失敗した場合、 はベストエフォートレスポンス AWS CloudHSM を行い、作成された (現在は不要な) 不要なキーをクリーンアップします。このプロセスでは、不要なキーマテリアルを直ちに除去すること、あるいは後で除去するために不要な材料をマーキングすることを含んでいます。どちらの場合も、解決するためにはお客様からのアクションは必要ありません。まれに、 AWS CloudHSM が不要なキーマテリアルを削除できず、マークが付いていない場合は、キーマテリアルを削除する必要があります。

問題: トークンキーの生成、インポート、またはアンラップ操作を試行し、tombstone への失敗を指定するエラーが表示される。

```
2018-12-24T18:28:54Z liquidSecurity ERR: print_node_ts_status:
[create_object_min_nodes]Key: 264617 failed to tombstone on node:1
```

原因： AWS CloudHSM 不要なキーマテリアルの削除とマークに失敗しました。

解決方法: クラスター内の HSM には、不要とマークされていない不要なキーマテリアルが含まれています。キーマテリアルを手動で削除する必要があります。不要なキーマテリアルを手動で削除するには、[key_mgmt_util (KMU)]、または[PKCS #11] あるいは JCE プロバイダーからの API を使用します。詳細については、deleteKey または Client SDK を参照してください。

トークンキーの耐久性を高めるために、 は、クライアント側の同期設定で指定された最小数の HSMs で成功しないキー作成オペレーションを AWS CloudHSM 失敗させます。詳細については、Key Synchronization in AWS CloudHSM を参照してください。

Client SDK 3: pkpspeed ツールを使用して HSM のパフォーマンスを検証する

このトピックでは、クライアント SDK 3 で HSM パフォーマンスを検証する方法について説明します。

AWS CloudHSM クラスター内の HSMs のパフォーマンスを確認するには、クライアント SDK 3 に含まれている pkpspeed (Linux) または pkpspeed_blocking (Windows) ツールを使用できます。pkpspeed ツールは理想的な条件下で実行され、PKCS11 などの SDK を経由せずに HSM を直接呼び出して操作を実行します。スケーリングのニーズを判断するために、アプリケーションの負荷を個別にテストすることを推奨します。ランダム (I)、(R)、EC ポイントマルチル ModExp (Y) のテストを実行することはお勧めしません。

Linux EC2 インスタンスにクライアントをインストールする方法の詳細については、「AWS CloudHSM クライアントのインストールと設定 (Linux)」を参照してください。Windows インスタンスにクライアントをインストールする方法の詳細については、「AWS CloudHSM クライアントのインストールと設定 (Windows)」を参照してください。

AWS CloudHSM クライアントをインストールして設定したら、次のコマンドを実行して起動します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

クライアントソフトウェアをインストール済みの場合は、必要に応じて最新バージョンの pkpspeed をダウンロードしてインストールします。pkpspeed ツールは、Linux の `/opt/cloudhsm/bin/pkpspeed` または Windows の `C:\Program Files\Amazon\CloudHSM\` にあります。

pkpspeed を使用するには、pkpspeed コマンドあるいは pkpspeed_blocking.exe を実行し、HSM の Crypto User (CU) のユーザー名とパスワードを指定します。次に、以下の推奨事項を考慮に入れながら、使用するオプションを設定します。

レコメンデーションをテストする

- RSA の署名および検証オペレーションのパフォーマンスをテストするには、Linux または Windows のオプション B の RSA_CRT 暗号を選択します。RSA は選択しないでください (option A in Windows のオプション A)。暗号は同じですが、RSA_CRT はパフォーマンス用に最適化されています。
- 少数のスレッドで開始します。AES パフォーマンスをテストする場合、通常、1 つのスレッドで十分に最大のパフォーマンスが示されます。RSA パフォーマンスをテストする場合 (RSA_CRT) は、通常、3〜4 つのスレッドで十分です。

pkpspeed ツールの設定可能なオプション

- FIPS モード: は常に FIPS モード AWS CloudHSM です (詳細については、AWS CloudHSM FAQsを参照してください)。これは、 AWS CloudHSM 「 ユーザーガイド」に記載されている CLI ツールを使用し、FIPS モードのステータスを示す getHSMInfo コマンドを実行することで確認できます。
- テストタイプ (ブロッキングとノンブロッキング): スレッド方式でのオペレーションの実行方法を指定します。ノンブロッキングを使用すると良い数値が得られる可能性が高くなるでしょう。これは、スレッドと同時実行性を利用するためです。
- スレッド数: テストを実行するスレッドの数。
- テスト実行時間 (最大 = 600): pkpspeed は「オペレーション/秒」で測定された結果を生成し、テストが実行される毎秒この値を報告します。例えば、テストを 5 秒間実行する場合、出力は次のサンプル値のようになります。
  - `OPERATIONS/second 821/1`
  - `OPERATIONS/second 833/1`
  - `OPERATIONS/second 845/1`
  - `OPERATIONS/second 835/1`
  - `OPERATIONS/second 837/1`

pkpspeed ツールで実行できるテスト

- AES GCM: AES GCM モードの暗号化をテストします。
- ベーシック 3DES CBC: 3DES CBC モードの暗号化をテストします。今後の変更については、以下の注記「1」を参照してください。
- ベーシック AES: AES CBC/ECB 暗号化をテストします。
- ダイジェスト: ハッシュダイジェストをテストします。
- ECDSA サイン: ECDSA サインをテストします。
- ECDSA 検証: ECDSA 検証をテストします。
- FIPS ランダム: FIPS 準拠の乱数の生成をテストします (注: これはブロッキングモードでのみ使用できます)。
- HMAC: HMAC をテストします。
- ランダム: FIPS 140-2 HSM を使用しているため、このテストは関係ありません。
- RSA 非 CRT と RSA_CRT の比較: RSA サインと検証オペレーションをテストします。

- RSA OAEP 暗号: RSA OAEP の暗号化をテストします。
- RSA OAEP 復号: RSA OAEP の復号化をテストします。
- RSA プライベート復号化非 CRT: RSA プライベートキー暗号化 (非最適化) をテストします。
- RSA プライベート復号化 CRT: RSA プライベートキー暗号化 (最適化) をテストします。
- RSA PSS サイン: RSA PSS サインをテストします。
- RSA PSS 検証: RSA PSS 検証をテストします。
- RSA パブリックキー暗号: RSA パブリックキー暗号化をテストします。

RSA パブリックキー暗号化、RSA プライベート復号化非 CRT、RSA プライベートキー復号化 CRT でも、ユーザーに次の回答を求めるプロンプトが表示されます。

```
Do you want to use static key [y/n]
```

y が入力された場合、事前に計算されたキーが HSM にインポートされます。

n が入力された場合、新しいキーが生成されます。

[1] NIST のガイダンスに従い、これは 2023 年以降、FIPS モードのクラスターでは許可されません。非 FIPS モードのクラスターの場合、2023 年以降も許可されます。詳細については、「FIPS 140 コンプライアンス: 2024 年 メカニズムの非推奨」を参照してください。

例

以下の例では、RSA オペレーションと AES オペレーションにおける HSM のパフォーマンスをテストするために pkpspeed (Linux) または pkpspeed_blocking (Windows) で選択できるオプションを示します。

Example - pkpspeed を使用した RSA パフォーマンスのテスト

ここの例は、Windows、Linux、および互換性のあるオペレーティングシステムで実行できます。

Linux

これらの手順は、Linux および互換性のあるオペレーティングシステムで使用してください。

```
/opt/cloudhsm/bin/pkpspeed -s CU user name -p password

SDK Version: 2.03
```

```
        Available Ciphers:
                AES_128
                AES_256
                3DES
                RSA  (non-CRT. modulus size can be 2048/3072)
                RSA_CRT (same as RSA)
For RSA, Exponent will be 65537

Current FIPS mode is: 00002
Enter the number of thread [1-10]: 3
Enter the cipher: RSA_CRT
Enter modulus length: 2048
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 245 bytes...
[Test duration is 60 seconds]

Do you want to use static key[y/n] (Make sure that KEK is available)?n
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password

Please select the test you want to run

RSA non-CRT------------------->A
RSA CRT----------------------->B
Basic 3DES CBC---------------->C
Basic AES--------------------->D
FIPS Random------------------->H
Random------------------------>I
AES GCM ---------------------->K

eXit------------------------->X
B

Running 4 threads for 25 sec

Enter mod size(2048/3072):2048
Do you want to use Token key[y/n]n
Do you want to use static key[y/n] (Make sure that KEK is available)?  n
OPERATIONS/second               821/1
OPERATIONS/second               833/1
```

```
OPERATIONS/second               845/1
OPERATIONS/second               835/1
OPERATIONS/second               837/1
OPERATIONS/second               836/1
OPERATIONS/second               837/1
OPERATIONS/second               849/1
OPERATIONS/second               841/1
OPERATIONS/second               856/1
OPERATIONS/second               841/1
OPERATIONS/second               847/1
OPERATIONS/second               838/1
OPERATIONS/second               843/1
OPERATIONS/second               852/1
OPERATIONS/second               837/
```

Example - pkpspeed を使用した AES パフォーマンスのテスト

Linux

これらの手順は、Linux および互換性のあるオペレーティングシステムで使用してください。

```
/opt/cloudhsm/bin/pkpspeed -s <CU user name> -p <password>

SDK Version: 2.03

        Available Ciphers:
                AES_128
                AES_256
                3DES
                RSA  (non-CRT. modulus size can be 2048/3072)
                RSA_CRT (same as RSA)
For RSA, Exponent will be 65537

Current FIPS mode is: 00000002
Enter the number of thread [1-10]: 1
Enter the cipher: AES_256
Enter the data size [1-16200]: 8192
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 8192 bytes...
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password
login as USER
Initializing Cfm2 library
        SDK Version: 2.03

 Current FIPS mode is: 00000002
Please enter the number of threads [MAX=400] : 1
Please enter the time in seconds to run the test [MAX=600]: 20


Please select the test you want to run

RSA non-CRT------------------->A
RSA CRT----------------------->B
Basic 3DES CBC---------------->C
Basic AES--------------------->D
FIPS Random------------------->H
Random------------------------>I
AES GCM ---------------------->K

eXit------------------------->X
D

Running 1 threads for 20 sec

Enter the key size(128/192/256):256
Enter the size of the packet in bytes[1-16200]:8192
OPERATIONS/second                9/1
OPERATIONS/second                10/1
OPERATIONS/second                11/1
OPERATIONS/second                10/1
OPERATIONS/second                10/1
OPERATIONS/second                10/...
```

Client SDK 5 ユーザーに矛盾する値が含まれている

`user list` コマンドは、クラスター内のすべてのユーザーとユーザープロパティのリストを返します。ユーザーのプロパティに「inconsistent」という値が付いているものがある場合、そのユーザーはクラスター全体で同期されません。つまり、そのユーザーはクラスター内の異なる HSM に異なる

プロパティで存在することになります。どのプロパティに一貫性がないかによって、異なる修復手順を取ることができます。

以下の表には、1 人のユーザーの不整合を解決する手順が記載されています。1 人のユーザーに複数の不整合がある場合は、以下の手順を上から順に実行して解決してください。不整合があるユーザーが複数いる場合は、ユーザーごとにこのリストを確認し、そのユーザーの不整合を完全に解決してから次のユーザーに進みます。

> **Note**
>
> これらの手順を実行するには、管理者としてログインするのが理想的です。管理者アカウントに一貫性がない場合は、管理者にログインし、すべてのプロパティが一致するまで手順を繰り返してください。管理者アカウントの一貫性が保たれたら、その管理者を使用してクラスター内の他のユーザーを同期できます。

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法

ユーザーの「役割」に「一貫性がない」	<pre>{
	"username":
	"test_user",
	"role":
	"**inconsistent** ",
	
	"locked":
	"false",
	"mfa": [],
	"cluster-
	coverage":
	"full"
	}</pre>
		このユーザーは、一部の HSM CryptoUser では であり、他の HSMs。 HSMs これは、2 つの SDK が同じユーザーを、異なるロールで同時に作成しようとした場合に発生する可能性があります。このユーザーを削除し、目的のロールで再作成する必要があります。	1. 管理者としてログインします。
			2. すべての HSM のユーザーを削除します。
			
			user delete --username **<user's name>** --role admin
			
			user delete --username **<user's name>** --role crypto-user
			3. 目的のロールを持つユーザーを作成します。

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法
			user create --username **<user's name>** --role **<desired role>**

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法

ユーザーの「クラスターカバレッジ」は「一貫性がない」	<pre>{
	"username":
	"test_user",
	
	"role": "crypto-user",
	"locked":
	"false",
	"mfa": [],
	"cluster-coverage":
	"**inconsistent** "
	}</pre>
		このユーザーはクラスター内の HSM のサブセットに存在します。これは、user create が部分的に成功した場合や、user delete が部分的に成功した場合に発生する可能性があります。
		
		このユーザーをクラスターから作成または削除して、前のオペレーションを完了する必要があります。
			ユーザーが存在しないはずの場合は、以下の手順に従ってください。
			
			1. 管理者としてログインします。
			2. 次のコマンドを実行します。
			
			user delete --username**<user's name>** --role admin
			3. 次のコマンドを実行します。
			
			user delete --username**<user's name>** --role crypto-user
			
			そのユーザーが存在しているはずなら、以下の手順に従ってください。
			
			1. 管理者としてログインします。
			2. 次のコマンドを実行します。
			
			user create --username **<user's name>**

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法
			--role **<desired role>**

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法

ユーザーの「ロック済み」パラメータが「不整合」または「true」です	{
	"username": "test_user",
	"role": "crypto-user",
	"locked": **inconsistent**,
	"mfa": [],
	"cluster-coverage": "full"
	}
		このユーザーは HSM のサブセットでロックアウトされています。
		
		これは、ユーザーが間違ったパスワードを使用し、クラスター内の HSM のサブセットにのみ接続した場合に発生する可能性があります。
		
		クラスター全体で一貫性を持たせるには、ユーザーの認証情報を変更する必要があります。
			ユーザーが MFA を有効にしている場合は、以下の手順で行います。
			
			1. 管理者としてログインします。
			2. 次のコマンドを実行して、MFA を一時的に無効にします。
			
			user change-mfa token-sign --username **<user's name>** --role **<desired role>** --disable
			
			3. すべての HSM にログインできるようにユーザーのパスワードを変更します。
			
			user change-password --username **<user's name>** --role **<desired role>**
			
			ユーザーの MFA を有効にする必要がある場合は、以下の手順で行います。

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法
			1. ユーザーにログインして MFA を再度有効にしてもらいます (その場合、ユーザーはトークンに署名し、パブリックキーを PEM ファイルに提供する必要があります)。 user change-mfa token-sign --username **<user's name>** --role **<desired role>** —token <File>

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法

MFA のステータスが「不整合」	<pre>{
	"username":
	"test_user",
	
	"role": "crypto-user",
	"locked":
	"false",
	"mfa": [
	{
	"strategy":
	"token-sign",
	"status":
	"**inconsistent** "
	}
	],
	"cluster-coverage":
	"full"
	}</pre>
		このユーザーは、クラスター内の HSM ごとに異なる MFA フラグを持っています。
		
		これは、MFA オペレーションが HSM のサブセットでのみ完了した場合に発生する可能性があります。
		
		ユーザーのパスワードをリセットし、MFA を再度有効にできるようにする必要があります。
			ユーザーが MFA を有効にしている場合は、以下の手順で行います。
			
			1. 管理者としてログインします。
			2. 次のコマンドを実行して、MFA を一時的に無効にします。
			
			user change-mfa token-sign --username **<user's name>** --role **<desired role>** --disable
			3. また、ユーザーがすべての HSM にログインできるように、ユーザーのパスワードを変更する必要があります。
			
			user change-password --username **<user's name>** --role **<desired role>**
			
			ユーザーの MFA を有効にする必要がある

プロパティに一貫性がありません	ユーザーリストの出力例	影響	復旧方法
			場合は、以下の手順で行います。 1. ユーザーにログインして MFA を再度有効にしてもらいます (その場合、ユーザーはトークンに署名し、パブリックキーを PEM ファイルに提供する必要があります)。 user change-mfa token-sign --username **<user's name>** --role **<desired role>** —token <File>

キーの可用性チェック中にエラーが表示された

問題: HSM から次のエラーが返されます。

```
Key <KEY HANDLE> does not meet the availability requirements - The key must be
 available on at least 2 HSMs before being used.
```

原因: キーの可用性チェックでは、まれではありますが、紛失する可能性のあるキーを探します。このエラーは通常、HSM が 1 つしかないクラスター、または HSM が 2 つあるクラスターで片方が交換されている間に発生します。このような状況では、以下の顧客オペレーションが原因で上記のエラーが発生した可能性があります。

- key generate-symmetric または などのコマンドを使用して新しいキーが生成されましたキー generate-asymmetric-pair 。
- キーリスト オペレーションが開始されました。
- SDK の新しいインスタンスが開始されました。

> **Note**
>
> OpenSSL は SDK の新しいインスタンスを頻繁にフォークします。

解決策/推奨事項: このエラーの発生を防ぐために、以下のアクションから選択してください。

- --disable-key-availability-check パラメータを使用して、設定ツールの構成ファイルで、キーの可用性を「false」に設定します。詳細については、設定ツールの「パラメータ」セクションを参照してください。
- HSM が 2 つあるクラスターを使用する場合は、コードの初期化中を除いて、エラーの原因となった操作は使用しないでください。
- クラスター内の HSM の数を 3 つ以上に増やしてください。

JCE によるキーの抽出

getEncoded 、 getPrivateExponent、または getS は null を返します

`getEncoded`、`getPrivateExponent`、`getS` はデフォルトでは無効になっているため、null を返します。これらを有効にするには、「JCE を使用したキー抽出」を参照してください。

有効にした後に `getEncoded`、`getPrivateExponent`、`getS` が null を返した場合は、キーが適切な前提条件を満たしていません。詳細については、「JCE を使用したキー抽出」を参照してください。

getEncoded 、 getPrivateExponent、または getS が HSM の外部でキーバイトを返す

お客様またはシステムへのアクセス権を持つユーザーがクリアキー抽出を有効にしました。この設定をデフォルトの無効状態にリセットする方法など、詳細については以下のページを参照してください。

- JCE を使用したキー抽出
- キーの保護と HSM からの抽出

HSM スロットリング

ワークロードがクラスターの HSM 容量を超えると、HSM がビジー状態またはスロットリングされていることを示すエラーメッセージが表示されます。この場合、スループットが低下したり、HSM からのリクエストを拒否する割合が高くなることがあります。さらに、HSM は次のビジーエラーを送信する可能性があります。

Client SDK 5 向け

- PKCS11 では、ビジーエラーは `CKR_FUNCTION_FAILED` にマップされます。このエラーは複数の理由で発生する可能性がありますが、HSM スロットリングによってこのエラーが発生すると、ログに次のログ行が表示されます。
  - `[cloudhsm_provider::hsm1::hsm_connection::e2e_encryption::error] Failed to prepare E2E response. Error: Received error response code from Server. Response Code: 187`
  - `[cloudhsm_pkcs11::decryption::aes_gcm] Received error from the server. Error: This operation is already in progress. Internal error code: 0x000000BB`
- JCE では、ビジーエラーは `com.amazonaws.cloudhsm.jce.jni.exception.InternalException: Unexpected error with the Provider: The HSM could not queue the request for processing.` にマップされます。
- 他の SDK のビジーエラーはメッセージ「`Received error response code from Server. Response Code: 187`」を出力します。

Client SDK 3 向け

- PKCS11 では、ビジーエラーは `CKR_OPERATION_ACTIVE` にマップされます。
- JCE では、ビジーエラーは `0xBB (187)` ステータスとして `CFM2Exception` にマップされます。アプリケーションは `CFM2Exception` の `getStatus()` 関数を使用して HSM からどのようなステータスが返されたかを確認できます。

- 他の SDK のビジーエラーはメッセージ「`HSM Error: HSM is already busy generating the keys(or random bytes) for another request.`」を出力します。

解決方法

この問題は、次の 1 つまたは複数のアクションを実行することで解決できます。

- 拒否された HSM オペレーションに対する再試行コマンドをアプリケーションレイヤーに追加します。再試行コマンドを有効にする前に、クラスターがピーク時の負荷に対応できる適切なサイズになっていることを確認してください。

> **Note**
>
> Client SDK 5.8.0 以降では、再試行コマンドはデフォルトでオンになっています。各 SDK の再試行コマンド設定の詳細については、「Client SDK 5 設定ツールの詳細設定」を参照してください。

- 「AWS CloudHSM クラスターでの HSMsの追加または削除」の手順に従って HSM をクラスターに追加してください。

> **Important**
>
> クラスターの負荷テストを行って予測すべきピーク負荷を決定し、高可用性を確保するためにクラスターに HSM を 1 つ追加することを推奨します。

クラスターの HSM で HSM ユーザーを同期する

HSM のユーザー を管理するには、cloudhsm_mgmt_util と呼ばれる AWS CloudHSM コマンドラインツールを使用します。通信時には、必ずツールの設定ファイル内にある HSM を使用します。設定ファイルに含まれていないクラスターの他の HSM は認識されません。

AWS CloudHSM は、クラスター内の他のすべての HSMs 間で HSMsのキーを同期しますが、HSM のユーザーまたはポリシーは同期しません。cloudhsm_mgmt_util を使用して HSM ユーザーを管理する と、これらのユーザーの変更は、cloudhsm_mgmt_util 設定ファイルに含まれているクラスターの一部の HSM にのみ影響を及ぼします。これにより、 がクラスター内の HSMs 間でキーを AWS CloudHSM 同期する際に問題が発生する可能性があります。これは、キーを所有するユーザーがクラスター内のすべての HSMs に存在するわけではない可能性があるためです。

このような問題を回避するには、ユーザーを管理する前に、cloudhsm_mgmt_util 設定ファイルを編集します。詳細については、「???」を参照してください。

クラスターに対する接続の消失

AWS CloudHSM クライアント を設定したときに、クラスター内の最初の HSM の IP アドレスを指定しました。この IP アドレスは、 AWS CloudHSM クライアントの設定ファイルに保存されます。クライアントが起動すると、この IP アドレスへの接続を試みます。接続できない場合 (HSM で障害が発生した、HSM を削除した場合など) は、次のようなエラーが表示されることがあります。

```
LIQUIDSECURITY: Daemon socket connection error
```

```
LIQUIDSECURITY: Invalid Operation
```

このようなエラーを解決するには、クラスター内のアクティブで到達可能な HSM の IP アドレスを指定して、設定ファイルを更新します。

AWS CloudHSM クライアントの設定ファイルを更新するには

1. 次のいずれかの方法を使用して、クラスター内のアクティブな HSM の IP アドレスを見つけます。
   - 「 コンソール」のクラスター詳細ページに [AWS CloudHSM HSM] を表示します。
   - AWS Command Line Interface （ AWS CLI) を使用して describe-clusters コマンドを発行します。

   この IP アドレスは、後の手順で必要になります。
2. クライアントを停止するには、次のコマンドを使用します。

   Amazon Linux

   ```
   $ sudo stop cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client stop
   ```

CentOS 7

```
$ sudo service cloudhsm-client stop
```

CentOS 8

```
$ sudo service cloudhsm-client stop
```

RHEL 7

```
$ sudo service cloudhsm-client stop
```

RHEL 8

```
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client stop
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

  AWS CloudHSM クライアントを起動したコマンドウィンドウで Ctrl +C を使用します。

3. クライアントの設定ファイルを更新するには、前のステップで検出した IP アドレスを指定して、次のコマンドを使用します。

```
$ sudo /opt/cloudhsm/bin/configure -a <IP address>
```

4. クライアントを起動するには、次のコマンドを使用します。

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- Windows クライアント 1.1.2+ の場合:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- Windows クライアント 1.1.1 以前の場合。

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

に AWS CloudHSM 監査ログがない CloudWatch

2018 年 1 月 20 日以前にクラスターを作成した場合は、このクラスターの監査ログの配信を有効にするために、サービスにリンクされたロールを手動で設定する必要があります。HSM クラスターでサービスリンクされたロールを有効にする方法については、サービスリンクされたロールを理解する、および IAM ユーザーガイド の サービスリンクされたロールを作成する を参照してください。

AES Key Wrap 用非準拠長 カスタム IV

このトラブルシューティングトピックは、アプリケーションが回復不可能なラップされたキーを生成するかの判断に役立ちます。この問題の影響を受けている場合、このトピックを活用して問題に対処してください。

トピック

- コードが回復不可能なラップされたキーを生成する可能性の判断
- コードが回復不可能なラップされたキーを生成する場合に実行が必要なアクション

コードが回復不可能なラップされたキーを生成する可能性の判断

以下の すべて の条件に当てはまる場合、影響を受けます。

条件	確認方法
PKCS #11 ライブラリをアプリケーションで使用します	PKCS #11 ライブラリは、`libpkcs11.so` フォルダ内の `/opt/cloudhsm/lib` ファイルとしてインストールされます。C 言語で書かれたアプリケーションは、通常 PKCS #11 ライブラリを直接使用しますが、Java で書かれ

条件	確認方法
	たアプリケーションは Java 抽象化レイヤーを介して間接的にライブラリを使用する場合があります。Windows を使用している場合、PKCS #11 ライブラリは現在 Windows では利用できないため、影響を受けません。
アプリケーションでは PKCS #11 ライブラリのバージョン 3.0.0 を特に使用します	AWS CloudHSM チームから E メールを受け取った場合、PKCS #11 ライブラリのバージョン 3.0.0 を使用している可能性があります。 アプリケーションインスタンスのソフトウェアバージョンを確認するには、次のコマンドを使用します。 `rpm -qa | grep ^cloudhsm`
AES キーラッピングを使用してキーをラップします	AES キーラップとは、AES キーを使用して他のキーをラップすることを意味します。対応するメカニズム名は CKM_AES_KEY_WRAP です。これは、関数 C_WrapKey とともに使用されます。CKM_AES_GCM や CKM_CLOUDHSM_AES_GCM などの初期化ベクトル (IV) を使用する他の AES ベースのラッピングメカニズムはこの問題の影響を受けません。関数とメカニズムの詳細情報

条件	確認方法
AES キーラッピングを呼び出すときにカスタム IV を指定し、この IV の長さは 8 より短くなります。	AES キーラップは通常、CK_MECHANISM の構造を使用して以下の通り初期化されます。 `CK_MECHANISM mech = {CKM_AES_KEY_WRAP, IV_POINTER, IV_LENGTH};` これは、次の場合にのみ適用されます。 • IV_POINTER は NULL ではありません • IV_LENGTH が 8 バイト未満です

上記の条件をすべて満たさない場合は、今すぐ読み取りを停止することができます。ラップされたキーは適切にアンラップでき、この問題は影響しません。それ以外の場合は、「the section called "コードが回復不可能なラップされたキーを生成する場合に実行が必要なアクション"」を参照してください。

コードが回復不可能なラップされたキーを生成する場合に実行が必要なアクション

次の 3 つの手順を実行する必要があります。

1. PKCS #11 ライブラリを新しいバージョンにすぐにアップグレードします
   - Amazon Linux、CentOS 6、RHEL 6 用最新 PKCS #11 ライブラリ
   - Amazon Linux 2、CentOS 7、RHEL 7 用最新 PKCS #11 ライブラリ
   - Ubuntu 16.04 LTS 用の最新の PKCS #11 ライブラリ
2. 標準に準拠した IV を使用するためソフトウェアを更新します

   サンプルコードに従い、 NULL IV のみを指定することを強くお勧めします。これにより、HSM は標準準拠のデフォルト IV を使用します。または、IV を 0xA6A6A6A6A6A6A6A6 の対応する IV 長さを持つ 8 として明示的に指定することもできます。AES キーラッピングに他の IV を使用することはお勧めしません。将来のバージョンの PKCS #11 ライブラリでは AES キーラッピングのカスタム IV を明示的に無効にします。

   IV を適切に指定するサンプルコードは、 の aes_wrapping.c に表示されます GitHub。

3. 既存のラップされたキーを特定して復元します

   PKCS #11 ライブラリのバージョン 3.0.0 を使用してラップしたキーを特定し、キーを回復する場合、サポート (https://aws.amazon.com/support) へお問い合わせください。

> **Important**
>
> この問題は、PKCS #11 ライブラリのバージョン 3.0.0 でラップされたキーにのみ影響します。PKCS #11 ライブラリの以前のバージョン (2.0.4 および番号の小さいパッケージ) またはそれ以降のバージョン (3.0.1 以上の番号のパッケージ) を使用してキーをラップできます。

クラスター作成エラーの解決

クラスターを作成すると、ロールがまだ存在しない場合、 は AWSServiceRoleForCloudHSM サービスにリンクされたロール AWS CloudHSM を作成します。がサービスにリンクされたロールを作成 AWS CloudHSM できない場合、クラスターの作成が失敗する可能性があります。

このトピックでは、クラスターを正常に作成できるように、代表的な問題の解決方法を示します。このロールは 1 回だけ作成する必要があります。サービスにリンクされたロールをアカウントに作成すると、サポートされている任意の方法を使用して、追加のクラスターを作成および管理できます。

以下のセクションでは、サービスにリンクされたロールに関連するクラスター作成エラーのトラブルシューティング対策を示します。これらの対策を試してもクラスターを作成できない場合は、AWS Support までお問い合わせください。 AWSServiceRoleForCloudHSM サービスにリンクされたロールの詳細については、「」を参照してくださいのサービスにリンクされたロール AWS CloudHSM。

トピック

- 不足しているアクセス権限の追加
- サービスにリンクされたロールを手動で作成する
- 非フェデレーティッドユーザーの使用

不足しているアクセス権限の追加

サービスにリンクされたロールを作成するには、`iam:CreateServiceLinkedRole` アクセス権限が必要です。クラスターを作成している IAM ユーザーにこのアクセス許可がない場合、 AWS アカ

ウントでサービスにリンクされたロールを作成しようとすると、クラスターの作成プロセスは失敗します。

アクセス権限が不足しているためにエラーが発生すると、以下のエラーメッセージが表示されます。

```
This operation requires that the caller have permission to call
 iam:CreateServiceLinkedRole to create the CloudHSM Service Linked Role.
```

このエラーを解決するには、クラスターを作成する IAM ユーザーに AdministratorAccess アクセス権限を付与するか、ユーザーの IAM ポリシーに iam:CreateServiceLinkedRole アクセス権限を追加します。手順については、「新しいユーザーまたは既存のユーザーへのアクセス権限の追加」を参照してください。

その後で、もう一度スタックを作成してみてください。

サービスにリンクされたロールを手動で作成する

IAM コンソール、CLI、または API を使用して、サービスにリンクされたロールを作成できます AWSServiceRoleForCloudHSM。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの作成」を参照してください。

非フェデレーティッドユーザーの使用

認証情報が の外部から送信されるフェデレーティッドユーザーは AWS、フェデレーティッド以外のユーザーのタスクの多くを実行できます。ただし、AWS は、サービスにリンクされたロールをフェデレーティッドエンドポイントから作成するための API コールをユーザーに許可していません。

この問題を解決するには、非フェデレーティッドユーザーを作成して iam:CreateServiceLinkedRole アクセス権限を付与するか、既存の非フェデレーティッドユーザーに iam:CreateServiceLinkedRole アクセス権限を付与します。次に、そのユーザーに から クラスターを作成 AWS CLIしてもらいます。これにより、サービスにリンクされたロールがアカウントに作成されます。

サービスにリンクされたロールが作成されたら、必要に応じて、非フェデレーティッドユーザーが作成したクラスターを削除できます。クラスターを削除しても、ロールには影響しません。その後、フェデレーティッドユーザーを含む、必要なアクセス許可を持つすべてのユーザーが、アカウントに AWS CloudHSM クラスターを作成できます。

ロールが作成されたことを検証するには、https://console.aws.amazon.com/iam/ で IAM コンソールを開き、ロールを選択します。または、 AWS CLIの IAM get-role コマンドを使用します。

```
$  aws iam get-role --role-name AWSServiceRoleForCloudHSM
{
    "Role": {
        "Description": "Role for CloudHSM service operations",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "cloudhsm.amazonaws.com"
                    }
                }
            ]
        },
        "RoleId": "AROAJ4I6WN5QVGG5G7CBY",
        "CreateDate": "2017-12-19T20:53:12Z",
        "RoleName": "AWSServiceRoleForCloudHSM",
        "Path": "/aws-service-role/cloudhsm.amazonaws.com/",
        "Arn": "arn:aws:iam::111122223333:role/aws-service-role/cloudhsm.amazonaws.com/
AWSServiceRoleForCloudHSM"
    }
}
```

クライアント設定ログの取得

AWS CloudHSM は、 クライアント SDK 3 と クライアント SDK 5 が AWS 、 サポートが問題のトラブルシューティングを行うための環境に関する情報を収集するためのツールを提供します。

トピック

- (Client SDK 5 対応ツール)
- (Client SDK 3 対応ツール)

(Client SDK 5 対応ツール)

スクリプトは、次の情報を抽出します。

- Client SDK 5 コンポーネントの設定ファイル
- 使用可能なログファイル

- オペレーティングシステムの現行バージョン
- パッケージの情報:

Client SDK 5 の情報ツールの実行

Client SDK 5 には、各コンポーネントのクライアントサポートツールが含まれていますが、すべてのツールは同じ機能を果たします。収集されたすべての情報を含む出力ファイルを作成するツールを実行します。

ツールは次のような構文を使用します。

```
[ pkcs11 | dyn | jce ]_info
```

たとえば、PKCS #11 ライブラリを実行する Linux ホストから対応情報を収集し、システムがデフォルトディレクトリに書き込むようにする場合、次のコマンドを実行します。

```
/opt/cloudhsm/bin/pkcs11_info
```

ツールは /tmp ディレクトリ内に出力ファイルを作成します。

PKCS #11 library

Linux で PKCS #11 ライブラリの対応データの収集

- 対応ツールを使用してデータを収集します。

  ```
  /opt/cloudhsm/bin/pkcs11_info
  ```

Windows で PKCS #11 ライブラリの対応データの収集

- 対応ツールを使用してデータを収集します。

  ```
  C:\Program Files\Amazon\CloudHSM\bin\pkcs11_info.exe
  ```

OpenSSL Dynamic Engine

Linux で OpenSSL Dynamic Engine の対応データの収集

- 対応ツールを使用してデータを収集します。

```
/opt/cloudhsm/bin/dyn_info
```

JCE provider

Linux で JCE プロバイダーの対応データの収集

- 対応ツールを使用してデータを収集します。

```
/opt/cloudhsm/bin/jce_info
```

Windows で JCE プロバイダの対応データの収集

- 対応ツールを使用してデータを収集します。

```
C:\Program Files\Amazon\CloudHSM\bin\jce_info.exe
```

サーバーレス環境からのログの取得

Fargate や Lambda などのサーバーレス環境に を設定するには、 AWS CloudHSM ログタイプを に設定することをお勧めしますterm。に設定するとterm、サーバーレス環境は に出力できるようになります CloudWatch。

からクライアントログを取得するには CloudWatch、「Amazon CloudWatch Logs ユーザーガイド」の「ロググループとログストリームの操作」を参照してください。

(Client SDK 3 対応ツール)

スクリプトは、次の情報を抽出します。

- オペレーティングシステムとその現在のバージョン
- cloudhsm_client.cfg、cloudhsm_mgmt_util.cfg、application.cfg ファイルからのクライアント設定情報

- プラットフォームに固有の場所からのクライアントログ
- cloudhsm_mgmt_util を使用してのクラスターと HSM 情報
- OpenSSL の情報
- 現在のクライアントとビルドのバージョン
- インストーラのバージョン

Client SDK 3 の情報ツールの実行

このスクリプトは、収集されたすべての情報を含む出力ファイルを作成します。スクリプトは `/tmp` ディレクトリ内に出力ファイルを作成します。

Linux: `/opt/cloudhsm/bin/client_info`

Windows: `C:\Program Files\Amazon\CloudHSM\client_info`

> **Warning**
>
> このスクリプトには、Client SDK 3 バージョン 3.1.0 ~ 3.3.1 に関する既知の問題があります。この問題の修正を含むバージョン 3.3.2 にアップグレードすることを強くお勧めします。このツールを使用する前の詳細情報は、既知の問題 ページを参照してください。

AWS CloudHSM クォータ

クォータは、以前は制限と呼ばれていましたが、 AWS リソースに割り当てられた値です。 AWS リージョンおよび AWS アカウントあたりの AWS CloudHSM リソースには、次のクォータが適用されます。デフォルトのクォータは、 によって適用される初期値であり AWS、これらの値を以下の表に示します。調整可能なクォータは、デフォルトのクォータよりも大きくすることができます。

Service Quotas

リソース	デフォルトのクォータ	引き上げ可能?
クラスター	4	あり
HSM	6	あり
クラスターあたりの HSM	28	なし

クォータの増加をリクエストする方法として推奨されるのは、Service Quotas コンソールを開く方法です。このコンソールで、サービスとクォータを選択し、リクエストを送信します。詳細については、Service Quotas ドキュメントを参照してください。

次のシステムクォータ表のクォータは調整できません。

システムクォータ

リソース	hsm1.medium のクォータ	hsm2m.medium のクォータ
クラスターあたりの最大キー	3,300	合計 16,666 個のキー、非対称キーの最大数は 3,333 個
クラスターあたりの最大ユーザー数	1,024	1,024
ユーザー名の最大長	31 文字	31 文字
必要なパスワードの長さ	8 ~ 32 文字	8 ~ 32 文字
クラスターあたりの同時クライアント接続の最大数[1]	900	900

リソース	hsm1.medium のクォータ	hsm2m.medium のクォータ
アプリケーションあたりの PKCS #11 セッションの最大数	1,024	1,024

[1] Client SDK 3 のクライアント接続はクライアントデーモンです。Client SDK 5 では、クライアント接続はアプリケーションです。

詳細については、「システムリソース」を参照してください。

システムリソース

システムリソースクォータは、実行時に AWS CloudHSM クライアントが使用できるクォータです。

ファイル記述子は、プロセスごとに開いているファイルを識別および管理するための、オペレーティングシステムのメカニズムです。

CloudHSM クライアントデーモンは、ファイル記述子を使用して、アプリケーションとクライアント間、およびクライアントとサーバー間の接続を管理します。

デフォルトでは、CloudHSM クライアント設定によって 3,000 個のファイル記述子が割り当てられます。このデフォルト値は、クライアントデーモンと HSM 間の最適なセッションとスレッド処理能力をもたらすように設計されています。

まれに、リソースが制限された環境でクライアントを実行している場合、これらのデフォルト値の変更が必要になることがあります。

> **Note**
>
> これらの値を変更すると、CloudHSM クライアントのパフォーマンスが低下したり、アプリケーションが動作不能になったりする可能性があります。

1. `/etc/security/limits.d/cloudhsm.conf` ファイルを編集します。

```
#
```

```
# DO NOT EDIT THIS FILE
#
hsmuser soft nofile 3000
hsmuser hard nofile 3000
```

2. 必要に応じて数値を編集します。

> **Note**
>
> `soft` クォータは、`hard` クォータ以下である必要があります。

3. CloudHSM クライアントデーモンプロセスを再起動します。

> **Note**
>
> この設定オプションは Microsoft Windows プラットフォームでは使用できません。

Client SDK AWS CloudHSM のダウンロード

ダウンロード

2021 年 3 月に、 クライアント SDK バージョン 5.0.0 が AWS CloudHSM リリースされました。これにより、さまざまな要件、機能、プラットフォームサポートを備えたまったく新しいクライアント SDK が導入されました。

Client SDK 5 は本番環境で完全にサポートされており、CNG および KSP プロバイダーのサポートを除き、Client SDK 3 と同じコンポーネントとレベルのサポートを提供します。詳細については、「Client SDK コンポーネントの比較」を参照してください。

> **Note**
>
> 各 Client SDK でサポートされているプラットフォームについては、Client SDK 5 がサポートするプラットフォーム「」および「」を参照してくださいClient SDK 3 がサポートするプラットフォーム。

最新リリース

このセクションには、最新バージョンの Client SDK が含まれています。

Client SDK 5 リリース: バージョン 5.12.0

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 383baed4a861391eb0923c0d9cf451851c6dd02d7d6a9e9cc3638c60bf300ef2)
- OpenSSL Dynamic Engine (SHA256 checksum f7aba68787a4c975f3e9f4ead28c2c28adc787ca0babebc070a928d226ff330a)
- JCE プロバイダー(SHA256 checksum 1f75f1a5d428b18ce2dc6ce8e17923009895c2545e2d04d76dafd6da914c0b4e)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs

- CloudHSM CLI (SHA256 checksum 4c27fae1ef5fd1642c04514ec84ad4cab78f59a32eb3fce59b51805c44b25295)

ARM64 アーキテクチャの Amazon Linux 2 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum c28a1f27e23e6ab1550dab6a353c6c9338a391a84d57f4ac99a1a3a9810c753f)
- OpenSSL Dynamic Engine (SHA256 checksum 7d2e864c31c13f55443c1b1d04589fbdd4558fe103954de4384691e2c429a872)
- JCE プロバイダー(SHA256 checksum e9a35eb87b2f257c47fb083d286deb835da45858b2d89759ca7d5bb4ef747b4b)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 28b6f918912b5c63bf10018824b642a805b309c21947a1d0ebbdc44647e80554)

Amazon Linux 2023

x86_64 アーキテクチャの Amazon Linux 2023 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 02801365cba449c5238a4e5ad3df1ddf7edd00ade976f47e956e885286503f3f)
- OpenSSL Dynamic Engine (SHA256 checksum 0abed69a7c6acaafdaabdcc5fab7d56611ffd94f5480cade6f8beace9aeae056)
- JCE プロバイダー(SHA256 checksum 3d5d9a903d3a216eca40f92dbb0b4030b7a86ad7ceee8d62241c97a6e1881e25)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum f96671d882b862033bba0b3633448dc6a26e45a25063e29b79a5cd4b7fc4945c)

ARM64 アーキテクチャの Amazon Linux 2023 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 53d05006b46bda8e9c1dd76e8307a780bfe0a67b10a9a87723c97f94e29f5b8e)
- OpenSSL Dynamic Engine (SHA256 checksum ec1cca8e01b3303ff9473eeef6b33dc85b6affac7a47387b098905f9f2fc85ba)

- JCE プロバイダー(SHA256 checksum c828ae56f46233215b9f35798b5859ebdac962af442acbc457081c3baaa44f11)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum ddd5dcd68d01f4fafaf13dc0b4ddcf98e3731ed51bdd51f85535b29353644a9f)

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 383baed4a861391eb0923c0d9cf451851c6dd02d7d6a9e9cc3638c60bf300ef2)
- OpenSSL Dynamic Engine (SHA256 checksum f7aba68787a4c975f3e9f4ead28c2c28adc787ca0babebc070a928d226ff330a)
- JCE プロバイダー(SHA256 checksum 1f75f1a5d428b18ce2dc6ce8e17923009895c2545e2d04d76dafd6da914c0b4e)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 4c27fae1ef5fd1642c04514ec84ad4cab78f59a32eb3fce59b51805c44b25295)

RHEL 7 (7.8+)

x86_64 アーキテクチャの RHEL 7 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 383baed4a861391eb0923c0d9cf451851c6dd02d7d6a9e9cc3638c60bf300ef2)
- OpenSSL Dynamic Engine (SHA256 checksum f7aba68787a4c975f3e9f4ead28c2c28adc787ca0babebc070a928d226ff330a)
- JCE プロバイダー(SHA256 checksum 1f75f1a5d428b18ce2dc6ce8e17923009895c2545e2d04d76dafd6da914c0b4e)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 4c27fae1ef5fd1642c04514ec84ad4cab78f59a32eb3fce59b51805c44b25295)

RHEL 8 (8.3+)

x86_64 アーキテクチャの RHEL 8 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 6e51e95122fd0991278888287f0c408808b26fb5f1196c46168477b9090fc478)
- OpenSSL Dynamic Engine (SHA256 checksum 1f1d52ff7af6c537d8cfeb5973c691a9d90a518accd685ff9b66cd78daf98928)
- JCE プロバイダー(SHA256 checksum 156944607de987d6b39bd8a2d21ccd294c01377a9e35f9f15f8b0f4c8bb90033)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 351e802f79dd2d0b5f7d23bb74c146be05e5169b603c9aace24189094a45a35d)

RHEL 9 (9.2+)

x86_64 アーキテクチャの RHEL 9 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum d1b2f4ac7e6e0c18e788512e7726bc68b571d99a1442ce2f2e80f4b0f9956266)
- OpenSSL Dynamic Engine (SHA256 checksum cf86a3f17cd6c51969d4ce80c1e3ea6513b995611be7e2e72e5e5233c71d6add)
- JCE プロバイダー(SHA256 checksum ae89e256eb89ec6b4fa0f001e7a4e1d8f1c08530423e81aa74d69a17b25d9a99)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum dfe6fe5d890c33b2f5d38f906ade113b06c8c05f3427a327744c454e7302f1a5)

ARM64 アーキテクチャの RHEL 9 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum cad72a6ab2232b4c38b90d7c62147520b975d646773dd90d7be897fa0a537d2d)
- OpenSSL Dynamic Engine (SHA256 checksum ad751f756530a2317c3c64380ea3a07865b13e1874fab0e61ac530b21487c7fb)
- JCE プロバイダー(SHA256 checksum d204e69acfbb90996fb08ae3573607b65630b1124fb379e078c002d55ac07766)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum c0f412cc59bafd235e046cdc1a0c5d330f2d72f7d6434672e9522f86bc945090)

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum d37b1f872eb2b1ab34303d5b8b803daa925902b645c57c6e15a28bb6321e0f42)
- OpenSSL Dynamic Engine (SHA256 checksum cdc6e737652556b57d26d8816b2bc9820128cb3919360660b6f7fe65f9d39e3f)
- JCE プロバイダー(SHA256 checksum f567a08344414a4776e1c5a9715657476925ca32695c4c2dd84a4f3fc5dc1615)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum f2ee5ad01c5018fc3670f602228fd71087228cd3923bf5b9bc73e4d7084dac6c)

Ubuntu 22.04 LTS

x86_64 アーキテクチャの Ubuntu 22.04 LTS 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 0e78928acd7a1662e4b07b15d5c3ccb88714ff89e47b991c8ab6e4c2229ee5aa)
- OpenSSL Dynamic Engine (SHA256 checksum 4f3168745edc5592234891a7b1d82b179a4947e87c72fade1be3bad58b7ed1a3)
- JCE プロバイダー(SHA256 checksum d4c3655cdc2b00d1ab5ceafac94dfbc5c5244ed20e10fdd9db9f4e741e013733)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum d00bbacb6f2e57bd92d832a2bd11cadede972f8e82cc402ec0684b9c6b23123c)

ARM64 アーキテクチャの Ubuntu 22.04 LTS 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 0c1121535c523acb864215338292bab32acee438357878b5fc0b6d268713b86f)
- OpenSSL Dynamic Engine (SHA256 checksum dc7a219302021570bc8c36674d2bd33165557bb2f9a0af8fdf114f1b85a70d84)

- JCE プロバイダー(SHA256 checksum af3834a10081f1e4e7894275c8b9c7b7649b8de3b6f0aeb0781a3358183a9046)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum baa253ac62c2fbcc5712561e0fb0feb25461efc3ce68cf86d4c7bf0af0f14a34)

Windows Server 2016

x86_64 アーキテクチャの Windows Server 2016 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 11c3255fcc90b47810cfe4b2f71d56a006d295efccdd90f0d3f2dec5d2bab893)
- JCE プロバイダー(SHA256 checksum 09001458196590f54352c0c8986f442003bfc2db71bac6392ce512899d386806)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum b446ad1387fe406dcc0a12b6de86fa98e9db4a18f9829b745efb87750c6e31ea)

Windows Server 2019

x86_64 アーキテクチャの Windows Server 2019 用のバージョン 5.12.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 11c3255fcc90b47810cfe4b2f71d56a006d295efccdd90f0d3f2dec5d2bab893)
- JCE プロバイダー(SHA256 checksum 09001458196590f54352c0c8986f442003bfc2db71bac6392ce512899d386806)
  - AWS CloudHSM (SHA256 checksum 7158bc80e3b5b0915d83c39d4c060060a43a79cc407b1f783383b9e20bc5ff43) 用の Javadocs
- CloudHSM CLI (SHA256 checksum b446ad1387fe406dcc0a12b6de86fa98e9db4a18f9829b745efb87750c6e31ea)

Client SDK 5.12.0 では、複数のプラットフォームに ARM サポートが追加され、すべての SDKs。CloudHSM CLI および JCE プロバイダーに新機能が追加されました。

プラットフォームのサポート

- すべての SDK の ARM64 アーキテクチャでの Amazon Linux 2023 のサポートが追加されました。 SDKs
- すべての SDK の ARM64 アーキテクチャで Red Hat Enterprise Linux 9 (9.2 以降) のサポートが追加されました。 SDKs

- すべての SDK の ARM64 アーキテクチャでの Ubuntu 22.04 LTS のサポートが追加されました。 SDKs

CloudHSM CLI

- 次のコマンドを追加しました。
  - キーレプリケート
- 複数のクラスターへの接続のサポートが追加されました。詳細については、「CloudHSM CLI を使用した複数のクラスターへの接続」を参照してください。

JCE プロバイダー

- を使用してキーを取得KeyReferenceSpecするために を追加しましたKeyStoreWithAttributes。
- を使用して複数のキーを一度に取得getKeysするための を追加しましたKeyStoreWithAttributes。

フォーマンスの向上

- すべての SDK の AES CBC NoPadding オペレーションのパフォーマンスが向上しました。 SDKs

以前の Client SDK リリース

このセクションでは、以前の Client SDK リリースを一覧表示します。

バージョン 5.11.0

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 9fc0cd7cf003a7cb7e42dbd19671d58a97fc3b3d871d284dc6ae7fd226598772)
- OpenSSL Dynamic Engine (SHA256 checksum 1df6669c971440d446890b0fbeb74125a423df7b14e7ac4577347be7ef176572)
- JCE プロバイダー(SHA256 checksum 148a3f1de55a68e3bb525fb2994645333a52c2e9e46946dd8d90fcbc90ab64fd)

    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum a68f4a56d4c539cfcc8a1e56e19b5ff385bb24936ea5f349255b4e9bfbee9aab)

ARM64 アーキテクチャの Amazon Linux 2 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 5ac16449ec149c9b5e7776865803245ab17d0f1ad56df80173840c5e8d257b19)
- OpenSSL Dynamic Engine (SHA256 checksum 28c2eb7f3f60172b0186e5c25f71bb7341537058a71f288673936766048083c1)
- JCE プロバイダー(SHA256 checksum 06c9d9d281c12b1d2bd9a7b601d6317e46cedf175706bbfa3e4dcaed6ba05448)
    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 218982bb17aa751969a7866b0a9ff27e7aa5007a07817627d9cc1f7d60a78160)

Amazon Linux 2023

x86_64 アーキテクチャの Amazon Linux 2023 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 55310ab333d18bcfabdc4b74115b040386b4508934bdff93e1d054c4c4a6f9ea)
- OpenSSL Dynamic Engine (SHA256 checksum f3d4934dc872a9b5212a180b9814ca2af3eca01ee228a8725563f1770add0dce)
- JCE プロバイダー(SHA256 checksum 757d3abb515aeb08f4b1c83970ee0979399efee00ee78c9a9dbec05f4ed9768d)
    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 22af8f0501ff9a45a9e0683a408a63771c2c06c66abf5478d310d6d32e013555)

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 9fc0cd7cf003a7cb7e42dbd19671d58a97fc3b3d871d284dc6ae7fd226598772)
- OpenSSL Dynamic Engine (SHA256 checksum 1df6669c971440d446890b0fbeb74125a423df7b14e7ac4577347be7ef176572)
- JCE プロバイダー(SHA256 checksum 148a3f1de55a68e3bb525fb2994645333a52c2e9e46946dd8d90fcbc90ab64fd)
  - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum a68f4a56d4c539cfcc8a1e56e19b5ff385bb24936ea5f349255b4e9bfbee9aab)

RHEL 7 (7.8+)

x86_64 アーキテクチャの RHEL 7 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 9fc0cd7cf003a7cb7e42dbd19671d58a97fc3b3d871d284dc6ae7fd226598772)
- OpenSSL Dynamic Engine (SHA256 checksum 1df6669c971440d446890b0fbeb74125a423df7b14e7ac4577347be7ef176572)
- JCE プロバイダー(SHA256 checksum 148a3f1de55a68e3bb525fb2994645333a52c2e9e46946dd8d90fcbc90ab64fd)
  - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum a68f4a56d4c539cfcc8a1e56e19b5ff385bb24936ea5f349255b4e9bfbee9aab)

RHEL 8 (8.3+)

x86_64 アーキテクチャの RHEL 8 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum b95b9f588656fb14fd08bb66ce0e0da807b96daa38348dec07a508c9bef7403a)
- OpenSSL Dynamic Engine (SHA256 checksum 7bb437b91a52e863b2b00ff7f427ce22522026daf757be873ee031ec6ffffd88)
- JCE プロバイダー(SHA256 checksum e0db887e05eb535314f4d99f21da12d87d35ebb8baf9726f4ce8f01d9df0ea01)
  - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 8485b5a6d679767ca9b4f611718159a643cf3e85090a8e4d20fe53c3707e25c3)

RHEL 9 (9.2+)

x86_64 アーキテクチャの RHEL 9 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 87b56a20accf67df53a203b7f115655b2acfaec4516682d4976d9475b10bec8e)
- OpenSSL Dynamic Engine (SHA256 checksum 83a6b58572e985df937beede4b10e867b0ac6050ace8010dc8d535be365d2747)
- JCE プロバイダー(SHA256 checksum ee95213d02d913250478d0793d6dd578e5c54d765e635c7468a49bdff4c2a6f3)
  - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 7e168ed3bef8e9c5110645e9960680e9a57f7b94e16aec71422e3c67ebc58fb5)

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum abc3a339d1fe5850db65620804e9a910f8b4f913624ef9b7189f2f0df1825c01)
- OpenSSL Dynamic Engine (SHA256 checksum 075fc3f9974d552f27ad67fa92c8abff31b756b9add875b8cd4957e6801583a4)
- JCE プロバイダー(SHA256 checksum 5de45c519133a0dae8da3ac01809db7974be25c14c15eb773fc5c972c0178c13)
  - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 83e0e4505a063792c19feb3d4cfd032b9089091916168d92b0f51a967a007734)

Ubuntu 22.04 LTS

x86_64 アーキテクチャの Ubuntu 22.04 LTS 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum b8f20be125c8530b2a7bd945956e9c04296fba5634af408b40be4e03bdbad72a)
- OpenSSL Dynamic Engine (SHA256 checksum d728c156eb4ee5c67159e57d6b092785800baa5fb61c14d64f460a8b8f53a778)
- JCE プロバイダー(SHA256 checksum 44e943b8cd1176ad666e249342687744a280c6222df58b5a9f084c932f628284)

    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 8ccf5389d459611be813e42d7f9d040090f94f3fe88f9d110bcfb25e9619e4a7)

Windows Server 2016

x86_64 アーキテクチャの Windows Server 2016 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum aa4bce5be15bbe0978b7205c619bb91c55a8e0f1f4636be311f24878f7709e07)
- JCE プロバイダー(SHA256 checksum 004cdb9ecb4a4d72458084997de7f562fb76a4e2f0567009f1dfafa7b2bded47)
    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 679795db759fda4823232142297a281e21a7d6f32cb5ddd6ac4c479866fa33b7)

Windows Server 2019

x86_64 アーキテクチャの Windows Server 2019 用のバージョン 5.11.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum aa4bce5be15bbe0978b7205c619bb91c55a8e0f1f4636be311f24878f7709e07)
- JCE プロバイダー(SHA256 checksum 004cdb9ecb4a4d72458084997de7f562fb76a4e2f0567009f1dfafa7b2bded47)
    - AWS CloudHSM (SHA256 checksum fb469ae53b516338f3326b402b15b7b84912801a8c25a28cd31a5da0631cd3c5) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 679795db759fda4823232142297a281e21a7d6f32cb5ddd6ac4c479866fa33b7)

Client SDK 5.11.0 では、すべての SDKs。

プラットフォームのサポート

- すべての SDKs。
- Ubuntu 18.04 LTS のサポートは、最近サポートが終了したため削除されました。
- 最近のサポート終了により、Amazon Linux のサポートが削除されました。

CloudHSM CLI

- 次のコマンドを追加しました。
  - 暗号記号
  - 暗号検証
  - キーインポート pem
  - キーアンラップ
  - キーラップ
- key generate-file でパブリックキーのエクスポートがサポートされるようになりました。

OpenSSL Dynamic Engine

- AWS CloudHSM OpenSSL Dynamic Engine は、OpenSSL ライブラリバージョン OpenSSL 3.x にインストールされているプラットフォームでサポートされるようになりました。これには、Amazon Linux 2023、RHEL 9 (9.2+)、Ubuntu 22.04 が含まれます。

JCE

- JDK 17 と JDK 21 のサポートが追加されました。
- HMAC オペレーションに使用される AES キーのサポートが追加されました。
- 新しいキー属性 を追加しましたID。
- キーを使い果たすための新しいDataExceptionCauseバリアント を導入しましたDataExceptionCause.KEY_EXHAUSTED。

バグ修正/機能向上

- label 属性の最大長を 126 文字から 127 文字に増やしました。
- RsaOaep メカニズムを使用して EC キーをアンラップできないバグを修正しました。
- JCE プロバイダーの getKey オペレーションの既知の問題を解決しました。詳細については、「問題: getKey オペレーションによるクライアント SDK 5 メモリリーク」を参照してください。
- FIPS 140-2 に従って、暗号化ブロックの最大制限に達したトリプル DES キーのすべての SDKs のログ記録が改善されました。
- OpenSSL Dynamic Engine の既知の問題を追加しました。詳細については、「OpenSSL Dynamic Engine SDK の既知の問題」を参照してください。

バージョン 5.10.0

Amazon Linux

x86_64 アーキテクチャの Amazon Linux 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum d63adf3e96c19c2d894b2defcbadd916dbb0398993050b1358bd93a36aa5acab)
- OpenSSL Dynamic Engine (SHA256 checksum 4daa3e591ffd5f7ce8ef3759c41deaa38867f5e5d21f15927aea83afb1678ac5)
- JCE プロバイダー(SHA256 checksum 6c1ac94d3080f1c609d9dafbcb14480911beef3a488c4ed6f2b11b377da9b477)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum c12617fcd7990ba53e96f477979b410e3a5f17842ca7a912861b8b820809b5b5)

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum fc47e705e57a0bfd433f7b46c9477a70df5c442a8ad9c2969bcef38e328e4933)
- OpenSSL Dynamic Engine (SHA256 checksum 0aca262df6780995c9b884fcb8765bbd64acaf21b2286ec4d05a9a90edb3d4cb)
- JCE プロバイダー(SHA256 checksum b5be7f73c4bcffc5da6f89f324e6b3db5b091610464c8bd38dbddfff0484b2c2)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum e8cf09966890b88a61e695dc034874a445093300359d5d6a86b5a546803920bb)

ARM64 アーキテクチャの Amazon Linux 2 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 5d8dfd835f1ed5a7f5a4fcc8ecf81cfa29883aca7e2985de69b5db723ab663db)

- OpenSSL Dynamic Engine (SHA256 checksum 91fb8efe2646bf0dbd9087554baa09554714e9d56e9bfd5c0dc3023a9f485574)
- JCE プロバイダー(SHA256 checksum 99f6e55c37fdf00085a816d46835aeff54470797b3b71f4d28a70dc79c9caf44)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 4a88ba9b4cf0dd5573f3dd88ab9dc257e4c486069cb529c5d554979ee2dd83af)

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum fc47e705e57a0bfd433f7b46c9477a70df5c442a8ad9c2969bcef38e328e4933)
- OpenSSL Dynamic Engine (SHA256 checksum 0aca262df6780995c9b884fcb8765bbd64acaf21b2286ec4d05a9a90edb3d4cb)
- JCE プロバイダー(SHA256 checksum b5be7f73c4bcffc5da6f89f324e6b3db5b091610464c8bd38dbddfff0484b2c2)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum e8cf09966890b88a61e695dc034874a445093300359d5d6a86b5a546803920bb)

RHEL 7 (7.8+)

x86_64 アーキテクチャーの RHEL 7 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum fc47e705e57a0bfd433f7b46c9477a70df5c442a8ad9c2969bcef38e328e4933)
- OpenSSL Dynamic Engine (SHA256 checksum 0aca262df6780995c9b884fcb8765bbd64acaf21b2286ec4d05a9a90edb3d4cb)
- JCE プロバイダー(SHA256 checksum b5be7f73c4bcffc5da6f89f324e6b3db5b091610464c8bd38dbddfff0484b2c2)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum e8cf09966890b88a61e695dc034874a445093300359d5d6a86b5a546803920bb)

RHEL 8 (8.3+)

x86_64 アーキテクチャーの RHEL 8 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 96afb7042a148ddc7a60ab6235b49e176d0460d1c2957bd76ca3d8406ac1cb03)
- OpenSSL Dynamic Engine (SHA256 checksum 2caad2bffea8aef73c91ad422d09772ef830fe7f80a7be19020e6a107eadfbe8)
- JCE プロバイダー(SHA256 checksum 3543551f08fbe3900821ea2d4ea148b4e86e2334bc94d7ffef6f3b831457cd71)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 812eccaadfc490f13bcd0b0a835ef58f3a3d4344ad7e0a237de476dd24509525)

Ubuntu 18.04 LTS

x86_64 アーキテクチャの Ubuntu 18.04 LTS 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum be4c61766b8b46e1f6c14c3dcf90aaab9f38240fcd9c68b4009704276c5f6f4a)
- OpenSSL Dynamic Engine (SHA256 checksum 64bd8af827b6dc3786e8ad28858cbc4ef6a0fd42164a0945f427eddcf5f02858)
- JCE プロバイダー(SHA256 checksum 9fcbdf08e93641468588b608173f26f18781bbc029ed95b2e086da29a968cc00)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 13808bddddb7eedeb2b8486d23a9976c7fa8d9220149a6b9400626bcaff3b513)

> **Note**
>
> Ubuntu 18.04 LTS の最近のサポート終了により、 は次のリリースでこのプラットフォームをサポート AWS CloudHSM できなくなります。

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 99ae96504580ff85ed4958a582903a847f666bdaafafbe887a5a76db58f24500)
- OpenSSL Dynamic Engine (SHA256 checksum 13e3f6fe086acf9617b163f66e3941f973daa583fb9322d16c396aa29fc3611d)
- JCE プロバイダー(SHA256 checksum 44562cebd9af1aa965840cd9bcb237e518d24c715b3c8bca1405c9c1871835e2)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum ab71b4ec531c5e6d05c91539c7edc1c07e6c748052ebf6200f148cb6812538c5)

Ubuntu 22.04 LTS

x86_64 アーキテクチャの Ubuntu 22.04 LTS 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum ee331a44fbe4936ec98a3ae55d58e67ed38e8bbff0a4f4ce8b1bd8239b75877b)
- OpenSSL Dynamic Engine のサポートは、このプラットフォームではまだ利用できません。
- JCE プロバイダー(SHA256 checksum 9e44d14dd33624f6fe36711633013e47e4a93f4d4635e08900546113ded56e3d)
  - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 2df361546848cd3f8965b1007dca42a0c959eb10d9e3f4995e8e1c852406751d)

Windows Server 2016

x86_64 アーキテクチャのWindows Server 2016 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 7aae9bfd99a6dd0f4d376c227c206c01847f83a9efd774d1063d76cc6fdaa89f)
- JCE プロバイダー(SHA256 checksum 1c58fd651e51be2ba59051a87aceca0452990b29837b8a7efabcd510ccbf8c1f)

    - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum f745a2236c9eb9f6f128313eddc35795bd5e47fdf67332bedeb2554201b61a24)

Windows Server 2019

x86_64 アーキテクチャのWindows Server 2019 用のバージョン 5.10.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 7aae9bfd99a6dd0f4d376c227c206c01847f83a9efd774d1063d76cc6fdaa89f)
- JCE プロバイダー(SHA256 checksum 1c58fd651e51be2ba59051a87aceca0452990b29837b8a7efabcd510ccbf8c1f)
    - AWS CloudHSM (SHA256 checksum dcbb870c6bd58c6770ba7a2b616c6103a5efb3bdeab831ce8f9c82cc09a9870f) 用の Javadocs
- CloudHSM CLI (SHA256 checksum f745a2236c9eb9f6f128313eddc35795bd5e47fdf67332bedeb2554201b61a24)

Client SDK 5.10.0 では安定性が向上し、すべての SDK のバグ修正が含まれています。

CloudHSM CLI

- 顧客が CloudHSM CLI を使用してキーを管理できるようにする次のような新しいコマンドを追加しました。
    - 対称キーと非対称キーペアを作成
    - 共有と共有解除キー
    - キー属性を使用してキーを一覧表示およびフィルタリングします
    - キー属性の設定
    - キーリファレンスファイルの生成
    - キーの削除
- エラーログ記録を改良しました。
- インタラクティブモードでの複数行の Unicode コマンドのサポートが追加されました。

バグ修正/機能向上

- すべての SDK のセッションキーのインポート、アンラップ、派生、作成のパフォーマンスが向上しました。

- 終了時に一時ファイルが削除されないという JCE プロバイダのバグを修正しました。
- クラスター内の HSM が置き換えられた後、特定の条件下で接続エラーが発生するバグを修正しました。
- 大きなマイナーバージョン番号を処理し、パッチ番号を含むように JCE `getVersion` 出力形式を変更しました。

プラットフォームのサポート

- JCE、PKCS #11、および CloudHSM CLI を搭載した Ubuntu 22.04 のサポートを追加しました (OpenSSL ダイナミックエンジンのサポートはまだありません)。

バージョン 5.9.0

Amazon Linux

x86_64 アーキテクチャの Amazon Linux 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 4f368be41f006b751ac41b14e1435c27841f60bbde0f032ec02a359fea637dcf)
- OpenSSL Dynamic Engine (SHA256 checksum 81af0d34683825cd6ff844ccacf9c8f4842a4ba76e3875a89121d09a286b4490)
- JCE プロバイダー(SHA256 checksum e8e5bc09d8e0b3cb24f30ab420fe08902a19073012335ac94382ec55fcc45abd)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 17284144b45043204ce012fe8b62b1973f10068950abedbd9c2c6172ed0979c6)

Amazon Linux 2

x86_64 アーキテクチャの Amazon Linux 2 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum e5affca37abc4ff76369237649830feb32fccd3fa05199cc2021230137093c56)
- OpenSSL Dynamic Engine (SHA256 checksum 848a2e31550bbc2b0223468877baa2a8cda3131ef8537856b31db226d55c4170)
- JCE プロバイダー(SHA256 checksum 884f483ef3e9c7def92e3ff01b226e5cbf276d96dcb2f6f56009516f19d41dc0)

    - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 2e62d5a27cff46d9fb47d656afeccd9dbfb5413bfd2267dd3c8fb7960fef7f26)

ARM64 アーキテクチャの Amazon Linux 2 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 4337dca5a08c5194b1118fa197bb4a4f7988df4e1b961e6f2e367295ba99d61d)
- OpenSSL Dynamic Engine (SHA256 checksum 4f08689934e877662a7ce64554fb04eb4b2c213b936018609ff187d100e34a85)
- JCE プロバイダー(SHA256 checksum b337b80271a2d308949d5911971fe6ad35df4e34876a481fcac347f1d897fe39)
    - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum a4d466e6b5f74dcd283ba32c9dd87441941d5e5a05936b7c2b4cc7ef85eb1071)

CentOS 7 (7.8+)

x86_64 アーキテクチャの CentOS 7 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum e5affca37abc4ff76369237649830feb32fccd3fa05199cc2021230137093c56)
- OpenSSL Dynamic Engine (SHA256 checksum 848a2e31550bbc2b0223468877baa2a8cda3131ef8537856b31db226d55c4170)
- JCE プロバイダー(SHA256 checksum 884f483ef3e9c7def92e3ff01b226e5cbf276d96dcb2f6f56009516f19d41dc0)
    - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 2e62d5a27cff46d9fb47d656afeccd9dbfb5413bfd2267dd3c8fb7960fef7f26)

RHEL 7 (7.8+)

x86_64 アーキテクチャーの RHEL 7 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum e5affca37abc4ff76369237649830feb32fccd3fa05199cc2021230137093c56)
- OpenSSL Dynamic Engine (SHA256 checksum 848a2e31550bbc2b0223468877baa2a8cda3131ef8537856b31db226d55c4170)

- JCE プロバイダー(SHA256 checksum 884f483ef3e9c7def92e3ff01b226e5cbf276d96dcb2f6f56009516f19d41dc0)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 2e62d5a27cff46d9fb47d656afeccd9dbfb5413bfd2267dd3c8fb7960fef7f26)

RHEL 8 (8.3+)

x86_64 アーキテクチャーの RHEL 8 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 081887f6ea1d9df9d1e409b2b5bde83e965c42229acbeb1f950c8fe478361edc)
- OpenSSL Dynamic Engine (SHA256 checksum 6b0500a42fd57c39f076f14e5079f80145b6ebd2c441395761eb04600c07bda5)
- JCE プロバイダー(SHA256 checksum 2bc7ac26b259af92a65fbd5a30d5eb2a92ce0e70efe41feb53bf82f168aa90bb)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 79ecbe9b4c5316ccf447d8c59b76b5ac2cc854bd79cd50c1f29197aa8cb080db)

Ubuntu 18.04 LTS

x86_64 アーキテクチャの Ubuntu 18.04 LTS 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum bc6d2227edd7b5a83fed32741fbacbb1756d5df89ebb3435d96f0609a180db65)
- OpenSSL Dynamic Engine (SHA256 checksum 2d6a26434fa6faf337f1dfb42de033220fa405a82d4540e279639a03b3ee6e9d)
- JCE プロバイダー(SHA256 checksum e12aef122f490e9026452ce31c25625b1accb9a5866b3d470488f10f047f1873)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum f0bcabe594db3e8ff86cc0f65c2a10858d34452eb6b9fc33d7aac05c0f5f4f30)

Ubuntu 20.04 LTS

x86_64 アーキテクチャの Ubuntu 20.04 LTS 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum 15dde8182f432de9e7d369b05e384e1f2d80dcca85db3b16ecc26cdef1a34bb9)
- OpenSSL Dynamic Engine (SHA256 checksum c8ba94a999038af87d4905b7c1feb4cc87e20d1776a32ef6f6d11ee000b5a896)
- JCE プロバイダー(SHA256 checksum de33cd3e8130a06d9da5207079533aac8276a1319ac435a3737b4f65bd8fb972)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum cfa31535ad9a99a5113496c06fbace38e9593491aca9bb031a18b51075973e68)

Windows Server 2016

x86_64 アーキテクチャの Windows Server 2016 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum ab5380805b0e17dd89dbbefd3fbda8b54da3c140f82e9f3d021850c31837bbe3)
- JCE プロバイダー(SHA256 checksum f0941d7a20193818133de8a742d3b848ea19abaf25f5a71ac65949ce5a37c533)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 131530ffe5caff963d483f440d06dcfb41dc11b0f8d78f1dd07bb07f76aeb6d2)

Windows Server 2019

x86_64 アーキテクチャの Windows Server 2019 用のバージョン 5.9.0 ソフトウェアをダウンロードします。

- PKCS #11 ライブラリ (SHA256 checksum ab5380805b0e17dd89dbbefd3fbda8b54da3c140f82e9f3d021850c31837bbe3)
- JCE プロバイダー(SHA256 checksum f0941d7a20193818133de8a742d3b848ea19abaf25f5a71ac65949ce5a37c533)
  - AWS CloudHSM (SHA256 checksum 6343427177180c8f61eec0341e827fbba29420ed2033c0e4b4803d49a3df7763) 用の Javadocs
- CloudHSM CLI (SHA256 checksum 131530ffe5caff963d483f440d06dcfb41dc11b0f8d78f1dd07bb07f76aeb6d2)

Client SDK 5.9.0 では安定性が向上し、すべての SDK のバグ修正が含まれています。すべての SDK が最適化され、HSM が使用できないと判断されるとすぐにアプリケーションにオペレーションの

失敗を通知できるようになりました。このリリースには JCE のパフォーマンス強化が含まれています。

JCE プロバイダー

- パフォーマンスの向上
- セッションプールの枯渇に関する既知の問題が修正されました

バージョン 3.4.4

クライアント SDK 3 を Linux プラットフォームでアップグレードするには、クライアントデーモンとすべてのライブラリを同時にアップグレードするバッチコマンドを使用する必要があります。アップデートの詳細については、「クライアント SDK 3 のアップグレード」を参照してください。

> **Note**
>
> Client SDK 3 および関連するコマンドラインツール (キー管理ユーティリティおよび CloudHSM 管理ユーティリティ) は、HSM タイプ hsm1.medium でのみ使用できます。詳細については、「AWS CloudHSM クラスターモードと HSM タイプ」を参照してください。

ソフトウェアをダウンロードするには、任意のオペレーティングシステムのタブを選択後、各ソフトウェアパッケージへのリンクを選択します。

Amazon Linux

Amazon Linux 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 900de424d70f41e661aa636f256a6a79cc43bea6b0fe6eb95c2aaa63e5289505)
- PKCS #11 ライブラリ (SHA256 checksum a3f93f084d59fee5d7c859292bc02cb7e7f15fb06e971171ebf9b52bbd229c30)
- OpenSSL Dynamic Engine (SHA256 checksum 8db07b9843d49016b0b6fec46d39881d94e426fcaae1cee2747be14af9313bb0)
- JCE プロバイダー(SHA256 checksum 360617c55bf4caa8e6e78ede079ca68cf9ef11473e7918154c22ba908a219843)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum c9961ffe38921131bd6f3702e10d73588e68b8ab10fbb241723e676f4fa8c4fa)

Amazon Linux 2

Amazon Linux 2 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- PKCS #11 ライブラリ (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- OpenSSL Dynamic Engine (SHA256 checksum 6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- JCE プロバイダー(SHA256 checksum 70e3cdce143c45a76e155ffb5969841e0153e011f59eb9f2c6e6be0707030abf)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

CentOS 6

AWS CloudHSM は、クライアント SDK バージョン 3.4.4 の CentOS 6 をサポートしていません。

CentOS 6 の場合、the section called “バージョン 3.2.1” を使用するか、またはサポートされているプラットフォームを選択します。

CentOS 7 (7.8+)

CentOS 7 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- PKCS #11 ライブラリ (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- OpenSSL Dynamic Engine (SHA256 checksum 6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- JCE プロバイダー(SHA256 checksum 70e3cdce143c45a76e155ffb5969841e0153e011f59eb9f2c6e6be0707030abf)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

CentOS 8

CentOS 8 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 81639c9ec83e501709c4117ba9d98b23dea7838a206ed244c9c6cc0d65130f8c)
- PKCS #11 ライブラリ (SHA256 checksum 9a15daa87b8616cf03a6bf6b375f53451ef448dbc54bf2c27fbc2be7823fc633)
- JCE プロバイダー(SHA256 checksum 2b1c4208992903cf7bcc669c1392c59a64fbfc82e010c626ffa58d0cb8e9126b)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 3adbecc802e0854c23aa4b8d80540d1748903c8dba93b6c8042fb7885051c360)

> **Note**
>
> 最近の CentOS 8 のサポート終了により、次のリリースではこのプラットフォームをサポートできなくなります。

RHEL 6

AWS CloudHSM は、クライアント SDK バージョン 3.4.4 の RedHat Enterprise Linux 6 をサポートしていません。

the section called “バージョン 3.2.1” for RedHat Enterprise Linux 6 を使用するか、サポートされているプラットフォームを選択します。

RHEL 7 (7.8+)

RedHat Enterprise Linux 7 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- PKCS #11 ライブラリ (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- OpenSSL Dynamic Engine (SHA256 checksum 6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- JCE プロバイダー(SHA256 checksum 70e3cdce143c45a76e155ffb5969841e0153e011f59eb9f2c6e6be0707030abf)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

RHEL 8 (8.3+)

RedHat Enterprise Linux 8 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 81639c9ec83e501709c4117ba9d98b23dea7838a206ed244c9c6cc0d65130f8c)
- PKCS #11 ライブラリ (SHA256 checksum 9a15daa87b8616cf03a6bf6b375f53451ef448dbc54bf2c27fbc2be7823fc633)
- JCE プロバイダー(SHA256 checksum 2b1c4208992903cf7bcc669c1392c59a64fbfc82e010c626ffa58d0cb8e9126b)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 3adbecc802e0854c23aa4b8d80540d1748903c8dba93b6c8042fb7885051c360)

Ubuntu 16.04 LTS

Ubuntu 16.04 LTS 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum 317c92c2e0b5d60afab1beb947f053d13ddaacb994cccc2c2b898e997ece29b9)
- PKCS #11 ライブラリ (SHA256 checksum 91451c420c51488a022569fd32f052a3b988a2883ea4c2ac952acb61a2fea37c)
- OpenSSL Dynamic Engine (SHA256 checksum 4098771ad0e38df9bf14d50520ca49b9395f819f0387e2bc3b0e61abb5888e66)
- JCE プロバイダー(SHA256 checksum e136ff183271c2f9590a9fccb8261a7eb809506686b070e3854df1b8686c6641)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum cbf24a4032f393a913a9898b1b27036392104e8e05d911cab84049b2bcca2541)

> **Note**
>
> Ubuntu 16.04 のサポート終了が迫っているため、次のリリースでこのプラットフォームのサポートを停止する予定です。

Ubuntu 18.04 LTS

Ubuntu 18.04 LTS 用のバージョン 3.4.4 ソフトウェアをダウンロードします。

- AWS CloudHSM クライアント (SHA256 checksum cf57d5e0e95efbf032aac8887aebd59ac8cc80e97c69e7c39fdad40873374fe8)

- PKCS #11 ライブラリ (SHA256 checksum 428f8bdad7925db5401112f707942ee8f3ca554f4ab53fa92237996e69144d2f)
- JCE プロバイダー(SHA256 checksum 1ff17b8f7688e84f7f0bfc96383564dca598a1cab2f2c52c888d0361682f2b9e)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum afe253046146ed6177c520b681efc680dac1048c4a95b3d8ad0f305e79bbe93e)

Windows Server

AWS CloudHSM は、Windows Server 2012、Windows Server 2012 R2、Windows Server 2016、および Windows Server 2019 の 64 ビットバージョンをサポートしています。Windows Server 用の AWS CloudHSM 3.4.4 クライアントソフトウェアには、必要な CNG および KSP プロバイダーが含まれています。詳細については、AWS CloudHSM 「クライアントのインストールと設定 (Windows)」を参照してください。Windows Server 用の最新バージョン (3.4.4) ソフトウェアをダウンロードします。

- AWS CloudHSM for Windows Server (SHA256 checksum d51a7db588e9121d8f0b0351606bd986e1c4de6547f2c8235200dc8a5ffbe53e)
- AWS CloudHSM 管理ユーティリティ (SHA256 checksum 0c12d7da9086735cdf189535937a8e036163009c5018dcaf2ee9cdbb6bd4c06f)

バージョン 3.4.4 では、JCE プロバイダーに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- log4j をバージョン 2.17.1 に更新します。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

非推奨のリリース

バージョン 5.8.0 以前は廃止されました。本番ワークロードでは、非推奨のリリースを使用することはお勧めしません。非推奨のリリースに下位互換性のあるアップデートを提供したり、ダウンロード用に非推奨のリリースをホストしたりすることはありません。非推奨のリリースの使用中に本番環境への影響が発生した場合は、アップグレードしてソフトウェアフィックスを入手する必要があります。

非推奨の Client SDK 5 リリース

このセクションでは、非推奨の Client SDK 5 リリースを一覧表示します。

バージョン 5.8.0

バージョン5.8.0では、CloudHSM CLI のクォーラム認証、JSSEによるSSL/TLSオフロード、PKCS #11 のマルチスロットサポート、JCEのマルチクラスター/マルチユーザーサポート、JCEによるキー抽出、KeyFactory for JCE、非端末リターンコードの新しい再試行構成が導入され、すべての SDK の安定性の向上とバグ修正が含まれています。

PKCS #11 ライブラリ

- マルチスロット構成のサポートが追加されました。

JCE プロバイダー

- 設定ベースのキー抽出が追加されました。
- マルチクラスター構成とマルチユーザー構成のサポートが追加されました。
- JSSE による SSL と TLS のオフロードのサポートが追加されました。
- AES/CBC/ のアンラップサポートを追加しましたNoPadding。
- 新しいタイプのキーファクトリ: SecretKeyFactory および を追加しました KeyFactory。

CloudHSM CLI

- クォーラム認証のサポートを追加する

バージョン 5.7.0

バージョン 5.7.0 では CloudHSM CLI が導入され、新しい暗号ベースのメッセージ認証コード (CMAC) アルゴリズムが含まれています。このリリースでは、Amazon Linux 2 に ARM アーキテクチャが追加されました。JCE プロバイダー Javadoc が AWS CloudHSMで利用できるようになりました。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。
- Amazon Linux 2 の ARM アーキテクチャでサポートされるようになりました。
- アルゴリズム
  - CKM_AES_CMAC (署名して確認)

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。
- Amazon Linux 2 の ARM アーキテクチャでサポートされるようになりました。

JCE プロバイダー

- 安定性の向上およびバグ修正。
- アルゴリズム
  - AESCMAC

バージョン 5.6.0

バージョン 5.6.0 には、PKCS #11 ライブラリと JCE プロバイダーの新しいメカニズムサポートが含まれています。さらに、バージョン 5.6 は Ubuntu 20.04 をサポートしています。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

- メカニズム
  - CKM_RSA_X_509、暗号化、復号化、署名、検証の各モード用

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 安定性の向上およびバグ修正。
- 暗号
  - RSA/ECB/NoPadding、暗号化モードと復号モード用

サポートされるキー

- 曲線 secp224r1 と secp521r1 の EC

プラットフォームのサポート

- Ubuntu 20.04 に追加されたサポート。

バージョン 5.5.0

バージョン5.5.0では、OpenJDK 11、キーツールとJarsignerの統合、およびJCEプロバイダへの追加メカニズムのサポートが追加されています。クラスがキーサイズパラメータをビット数ではなくバイト数として KeyGenerator 誤って解釈することに関する既知の問題を解決します。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- Keytool ユーティリティと Jarsigner ユーティリティのサポート
- すべてのプラットフォームでの OpenJDK 11 のサポート
- 暗号
  - AES/CBC/NoPadding Encrypt および Decrypt モード
  - AES/ECB/PKCs5Padding (暗号化モードと復号化モード)
  - AES/CTR/NoPadding Encrypt および Decrypt モード
  - AES/GCM/NoPadding ラップモードとアンラップモード
  - desede/ECB/PKCS5Padding (暗号化モードと復号化モード)
  - DESede /CBC/NoPadding Encrypt and Decrypt モード
  - AESWrap /ECB/NoPadding Wrap および Unwrap モード
  - AESWrap/ECB/PKCS5Padding (ラップ/アンラップモード)
  - AESWrap /ECB/ZeroPadding Wrap および Unwrap モード
  - RSA/ECB/PKCS1-Padding (ラップ/アンラップモード)
  - RSA/ECB/OAEPPadding (ラップ/アンラップモード)
  - RSA/ECB/OAEP (SHA-1 と MGF1 パディング、ラップ/アンラップモード)
  - RSA/ECB/OAEP (SHA-224 と MGF1 パディング、ラップ/アンラップモード)
  - RSA/ECB/OAEP (SHA-256 と MGF1 パディング、ラップ/アンラップモード)
  - RSA/ECB/OAEP (SHA-384 と MGF1 パディング、ラップ/アンラップモード)
  - RSA/ECB/OAEP (SHA-512 と MGF1 パディング、ラップ/アンラップモード)
  - RSAAESWrap/ECB/OAEPPadding (ラップ/アンラップモード)
  - RSaaES Wrap/ECB/OAEP (SHA-1 と MGF1 パディング、ラップ/アンラップモード)
  - RSaaES Wrap/ECB/OAEP (SHA-224 と MGF1 パディング、ラップ/アンラップモード)
  - RSaaES Wrap/ECB/OAEP (SHA-256 と MGF1 パディング、ラップ/アンラップモード)
  - RSaaES Wrap/ECB/OAEP (SHA-384 と MGF1 パディング、ラップ/アンラップモード)
  - RSaaES Wrap/ECB/OAEP (SHA-512 と MGF1 パディング、ラップ/アンラップモード)
- KeyFactory および SecretKeyFactory
  - RSA – 2048 ~ 4096 ビットの RSA キー (256 ビットの増分)
  - AES – 128、192、256 ビットの AES キー

  - NIST 曲線 secp256r1 (P-256)、secp384r1 (P-384)、および secp256k1 を対象とした EC キーペア
  - DeSede (3DES)
  - GenericSecret
  - HMAC – SHA1、SHA224、SHA256、SHA384、SHA512 ハッシュをサポート
- 署名/検証
  - RSASSA-PSS
  - SHA1withRSA/PSS
  - SHA224withRSA/PSS
  - SHA256withRSA/PSS
  - SHA384withRSA/PSS
  - SHA512withRSA/PSS
  - SHA1withRSAandMGF1
  - SHA224withRSAandMGF1
  - SHA256withRSAandMGF1
  - SHA384withRSAandMGF1
  - SHA512withRSAandMGF1

バージョン 5.4.2

バージョン 5.4.2 では、すべての SDK の安定性が向上し、バグが修正されています。これは CentOS 8 プラットフォームの最後のリリースでもあります。詳細については、「CentOS のウェブサイト」を参照してください。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 安定性の向上およびバグ修正。

バージョン 5.4.1

バージョン 5.4.1 では、PKCS #11 ライブラリの既知の問題が解決されました。これは CentOS 8 プラットフォームの最後のリリースでもあります。詳細については、「CentOS のウェブサイト」を参照してください。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 安定性の向上およびバグ修正。

バージョン 5.4.0

バージョン 5.4.0 では、すべてのプラットフォームの JCE プロバイダーの初期サポートが追加されています。JCE プロバイダーは OpenJDK 8 と互換性があります。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- キータイプ
  - RSA – 2048 ~ 4096 ビットの RSA キー (256 ビットの増分)。
  - AES – 128、192、256 ビットの AES キー。
  - NIST 曲線 secp256r1 (P-256)、secp384r1 (P-384)、および secp256k1 を対象とした ECC キーペア。
  - DeSede (3DES)
  - HMAC – SHA1、SHA224、SHA256、SHA384、SHA512 ハッシュをサポート。
- 暗号 (暗号化と復号のみ)
  - AES/GCM/NoPadding
  - AES/ECB/NoPadding
  - AES/CBC/PKCS5Padding
  - DESede/ECB/NoPadding
  - スウェーデン/CBC/PKCS5 パディング
  - AES/CTR/NoPadding
  - RSA/ECB/PKCS1Padding
  - RSA/ECB/OAEPPadding
  - RSA/ECB/OAEPWithSHA-1ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-224ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-256ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-384ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-512ANDMGF1Padding
- ダイジェスト
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- 署名/検証
  - NONEwithRSA

- SHA1withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA
- NONEwithECDSA
- SHA1 (ECDSA 搭載)
- SHA224 (ECDSA 搭載)
- SHA256 (ECDSA 搭載)
- SHA384 (ECDSA 搭載)
- SHA512 (ECDSA 搭載)

- Java との統合 KeyStore

バージョン 5.3.0

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 曲線 P-256、P-384、secp256k1 による ECDSA 署名/検証のサポートを追加します。
- プラットフォームのサポートを追加: Amazon Linux、Amazon Linux 2、Centos 7.8+、RHEL 7 (7.8+)。
- OpenSSL バージョン 1.0.2 のサポートが追加されました。
- 安定性の向上およびバグ修正。

JCE プロバイダー

- キータイプ
  - RSA – 2048 ~ 4096 ビットの RSA キー (256 ビットの増分)。
  - AES – 128、192、256 ビットの AES キー。

  - NIST 曲線 secp256r1 (P-256)、secp384r1 (P-384)、および secp256k1 を対象とした EC キーペア。
  - DeSede (3DES)
  - HMAC – SHA1、SHA224、SHA256、SHA384、SHA512 ハッシュをサポート。
- 暗号 (暗号化と復号のみ)
  - AES/GCM/NoPadding
  - AES/ECB/NoPadding
  - AES/CBC/PKCS5Padding
  - DESede/ECB/NoPadding
  - スウェーデン/CBC/PKCS5 パディング
  - AES/CTR/NoPadding
  - RSA/ECB/PKCS1Padding
  - RSA/ECB/OAEPPadding
  - RSA/ECB/OAEPWithSHA-1ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-224ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-256ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-384ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-512ANDMGF1Padding
- ダイジェスト
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- 署名/検証
  - NONEwithRSA
  - SHA1withRSA
  - SHA224withRSA
  - SHA256withRSA
  - SHA384withRSA

- SHA512withRSA
- NONEwithECDSA
- SHA1 (ECDSA 搭載)
- SHA224 (ECDSA 搭載)
- SHA256 (ECDSA 搭載)
- SHA384 (ECDSA 搭載)
- SHA512 (ECDSA 搭載)
- Java との統合 KeyStore

バージョン 5.2.1

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

バージョン 5.2.0

バージョン 5.2.0 では、PKCS #11 ライブラリの追加のキーのタイプとメカニズムのサポートが追加されました。

PKCS #11 ライブラリ

キーのタイプ

- ECDSA– P-224、P-256、P-384、P-521、および楕円曲線暗号 secp256k1
- Triple DES (3DES)

メカニズム

- CKM_EC_KEY_PAIR_GEN
- CKM_DES3_KEY_GEN
- CKM_DES3_CBC

- CKM_DES3_CBC_PAD
- CKM_DES3_ECB
- CKM_ECDSA
- CKM_ECDSA_SHA1
- CKM_ECDSA_SHA224
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384
- CKM_ECDSA_SHA512
- 暗号化および復号用 CKM_RSA_PKCS

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

バージョン 5.1.0

バージョン 5.1.0 では、PKCS #11 ライブラリの追加のメカニズムのサポートが追加されました。

PKCS #11 ライブラリ

メカニズム

- ラップおよびラップ解除用 CKM_RSA_PKCS
- CKM_RSA_PKCS_PSS
- CKM_SHA1_RSA_PKCS_PSS
- CKM_SHA224_RSA_PKCS_PSS
- CKM_SHA256_RSA_PKCS_PSS
- CKM_SHA384_RSA_PKCS_PSS
- CKM_SHA512_RSA_PKCS_PSS
- CKM_AES_ECB
- CKM_AES_CTR
- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_SP800_108_COUNTER_KDF

- CKM_GENERIC_SECRET_KEY_GEN
- CKM_SHA_1_HMAC
- CKM_SHA224_HMAC
- CKM_SHA256_HMAC
- CKM_SHA384_HMAC
- CKM_SHA512_HMAC
- CKM_RSA_PKCS_OAEP ラップおよびラップ解除のみ
- CKM_RSA_AES_KEY_WRAP
- CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD

API オペレーション

- C_CreateObject
- C_DeriveKey
- C_WrapKey
- C_UnWrapKey

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

バージョン 5.0.1

バージョン 5.0.1 では、OpenSSL Dynamic Engine の初期サポートが追加されました。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- OpenSSL Dynamic Engine の初期リリース

- このリリースでは、キーのタイプと OpenSSL API の入門サポートを提供しています。
  - 2048、3072、および 4096 ビットキーの RSA キーの生成
  - OpenSSL API:
    - SHA1/224/256/384/512 と RSA PSS で RSA PKCS を使った RSA サイン
    - RSA キーの生成

  詳細については、「OpenSSL Dynamic Engine」を参照してください。
- 対応プラットフォーム: CentOS 8.3+、Red Hat Enterprise Linux (RHEL) 8.3+、Ubuntu 18.04 LTS
  - 必要事項: OpenSSL 1.1.1

  詳細については、「Supported Platforms (サポートされているプラットフォーム)」を参照してください。
- NGINX 1.19 を含む、CentOS 8.3+、Red Hat Enterprise Linux (RHEL) 8.3、Ubuntu 18.04 LTS での SSL/TLS のオフロードのサポート (一部の暗号スイート向け)。

  詳細については、「Using SSL/TLS Offload on Linux (Linux 上の SSL/TLS オフロード)」を参照してください。

バージョン 5.0.0

バージョン 5.0.0 が最初のリリースです。

PKCS #11 ライブラリ

- これは最初のリリースです。

Client SDK バージョン 5.0.0 での PKCS #11 ライブラリ入門サポート

このセクションでは、Client SDK バージョン 5.0.0 のキーのタイプ、メカニズム、API オペレーション、および属性のサポートについて詳しく説明します。

キーのタイプ

- AES - 128、192、256 ビットの AES キー
- RSA - 2048 ~ 4096 ビットの RSA キー (256 ビットの増分)

メカニズム

- CKM_AES_GCM
- CKM_AES_KEY_GEN
- CKM_CLOUDHSM_AES_GCM
- CKM_RSA_PKCS
- CKM_RSA_X9_31_KEY_PAIR_GEN
- CKM_SHA1
- CKM_SHA1_RSA_PKCS
- CKM_SHA224
- CKM_SHA224_RSA_PKCS
- CKM_SHA256
- CKM_SHA256_RSA_PKCS
- CKM_SHA384
- CKM_SHA384_RSA_PKCS
- CKM_SHA512
- CKM_SHA512_RSA_PKCS

API オペレーション:

- C_CloseAllSessions
- C_CloseSession
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate
- C_Encrypt
- C_EncryptFinal

- C_EncryptInit
- C_EncryptUpdate
- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignUpdate
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyUpdate

属性:

- GenerateKeyPair
  - すべての RSA キー属性
- GenerateKey
  - すべての AES キー属性
- GetAttributeValue
  - すべての RSA キー属性
  - すべての AES キー属性

サンプル

- キーの生成 (AES、RSA、EC)
- キー属性のリスト化
- AES GCM を使用したデータの暗号化および復号
- RSAを使用したデータの署名と検証

非推奨の Client SDK 3 リリース

このセクションでは、非推奨の Client SDK 3 リリースを一覧表示します。

バージョン 3.4.3

バージョン 3.4.3 では、JCE プロバイダーに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- log4j をバージョン 2.17.0 に更新します。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.4.2

バージョン 3.4.2 では、JCE プロバイダーに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- log4j をバージョン 2.16.0 に更新します。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.4.1

バージョン 3.4.1 では、JCE プロバイダーに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- log4j をバージョン 2.15.0 に更新します。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.4.0

バージョン 3.4.0 では、すべてのコンポーネントに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 安定性の向上およびバグ修正。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 安定性の向上およびバグ修正。

Windows (CNG および KSG プロバイダー)

- 安定性の向上およびバグ修正。

バージョン 3.3.2

バージョン 3.3.2 では 問題 解決に client_info スクリプトを使用します。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- 整合性のために更新されたバージョン。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.3.1

バージョン 3.3.1 では、すべてのコンポーネントに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 安定性の向上およびバグ修正。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 安定性の向上およびバグ修正。

Windows (CNG および KSG プロバイダー)

- 安定性の向上およびバグ修正。

バージョン 3.3.0

バージョン 3.3.0 では、2 要素認証 (2FA) の追加などの改良が行われました。

AWS CloudHSM クライアントソフトウェア

- Crypto Officer (CO) の 2FA 認証を追加しました。詳細については、「Managing Two-Factor Authentication for Crypto Officers (Crypto Officer 用の 2 要素認証の管理)」を参照してください。
- RedHat Enterprise Linux 6 および CentOS 6 のプラットフォームサポートを削除しました。詳細については、「Linux サポート」を参照してください。
- クライアント SDK 5 またはクライアント SDK 3 で使用する独立型の CMU を追加しました。これは、バージョン 3.3.0 のクライアントデーモンに含まれている CMU のバージョンと同じで、クライアントデーモンをダウンロードせずに CMU をダウンロードできるようになりました。

PKCS #11 ライブラリ

- 安定性の向上およびバグ修正。
- RedHat Enterprise Linux 6 および CentOS 6 のプラットフォームサポートを削除しました。詳細については、「Linux サポート」を参照してください。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。
- RedHat Enterprise Linux 6 および CentOS 6 のプラットフォームサポートを削除しました。詳細については、「Linux サポート」を参照してください。

JCE プロバイダー

- 安定性の向上およびバグ修正。
- RedHat Enterprise Linux 6 および CentOS 6 のプラットフォームサポートを削除しました。詳細については、「Linux サポート」を参照してください。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.2.1

バージョン 3.2.1 では、PKCS #11 ライブラリの AWS CloudHSM 実装と PKCS #11 標準、新しいプラットフォーム、その他の改善点の間のコンプライアンス分析が追加されました。

AWS CloudHSM クライアントソフトウェア

- CentOS 8、RHEL 8、Ubuntu 18.04 LTS 用のプラットフォームのサポートを追加します。詳細については、「???」を参照してください。

PKCS #11 ライブラリ

- クライアント SDK 3.2.1 用 PKCS #11 ライブラリコンプライアンスレポート
- CentOS 8、RHEL 8、Ubuntu 18.04 LTS 用のプラットフォームのサポートを追加します。詳細については、「???」を参照してください。

OpenSSL Dynamic Engine

- CentOS 8、RHEL 8、Ubuntu 18.04 LTS 用のサポートがありません。詳細については、「???」を参照してください。

JCE プロバイダー

- CentOS 8、RHEL 8、Ubuntu 18.04 LTS 用のプラットフォームのサポートを追加します。詳細については、「???」を参照してください。

Windows (CNG および KSG プロバイダー)

- 安定性の向上およびバグ修正。

バージョン 3.2.0

バージョン 3.2.0 では、パスワードのマスキングのサポートやその他の改善点が追加されました。

AWS CloudHSM クライアントソフトウェア

- コマンドラインツールを使用するときにパスワードを非表示にするサポートが追加されました。詳細については、「loginHSM および logoutHSM (CloudHSM _mgmt_util)」と「loginHSM および logoutHSM (key_mgmt_util)」を参照してください。

PKCS #11 ライブラリ

- 以前サポートされていなかった一部の PKCS #11 メカニズムについて、ソフトウェアでラージデータをハッシュするためのサポートが追加されました。詳細については、「Supported Mechanisms (サポートされているメカニズム)」を参照してください。

OpenSSL Dynamic Engine

- 安定性の向上およびバグ修正。

JCE プロバイダー

- 整合性のために更新されたバージョン。

Windows (CNG および KSG プロバイダー)

- 安定性の向上およびバグ修正。

バージョン 3.1.2

バージョン 3.1.2 では、JCE プロバイダーに更新が追加されます。

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- log4j をバージョン 2.13.3 に更新します。

Windows (CNG および KSG プロバイダー)

- 整合性のために更新されたバージョン。

バージョン 3.1.1

AWS CloudHSM クライアントソフトウェア

- 整合性のために更新されたバージョン。

PKCS #11 ライブラリ

- 整合性のために更新されたバージョン。

OpenSSL Dynamic Engine

- 整合性のために更新されたバージョン。

JCE プロバイダー

- パフォーマンス向上とバグ修正が行われています。

Windows (CNG、KSP)

- 整合性のために更新されたバージョン。

バージョン 3.1.0

バージョン 3.1.0 では、標準に準拠した AES キーラップが追加されました。

AWS CloudHSM クライアントソフトウェア

- アップグレードの新しい要件: クライアントのバージョンは、使用しているソフトウェアライブラリのバージョンと一致する必要があります。アップグレードするには、クライアントとすべてのライブラリを同時にアップグレードするバッチコマンドを使用する必要があります。詳細については、「クライアント SDK 3 のアップグレード」を参照してください。
- Key_mgmt_util (KMU) には次の更新が含まれています。
  - 2 つの新しい AES キーラップ方法が追加されました。標準に準拠したゼロパディングを使用する AES キーラップと、パディングなしの AES キーラップです。詳細については、「wrapKey」および「unwrapKey」を参照してください。
  - AES_KEY_WRAP_PAD_PKCS5 を使用してキーをラップするときにカスタム IV を指定する機能が無効になりました。詳細については、「AES キーラップ」を参照してください。

PKCS #11 ライブラリ

- 2 つの新しい AES キーラップ方法が追加されました。標準に準拠したゼロパディングを使用する AES キーラップと、パディングなしの AES キーラップです。詳細については、「AES キーラップ」を参照してください。
- RSA-PSS 署名のソルトの長さを設定できます。この機能の使用方法については、 の「RSA-PSS 署名の設定可能なソルトの長さ」を参照してください GitHub。

OpenSSL Dynamic Engine

- 重要な変更: SHA1 を使用する TLS 1.0 および 1.2 暗号スイートは、OpenSSL Engine 3.1.0 では利用できません。この問題は間もなく解決されます。
- RHEL 6 または CentOS 6 に OpenSSL Dynamic Engine ライブラリをインストールする場合は、これらのオペレーティングシステムにインストールされているデフォルトの OpenSSL バージョンに関する既知の問題を参照してください。
- 安定性の向上およびバグ修正。

JCE プロバイダー

- 重要な変更: Java Cryptography Extension (JCE) 準拠の問題に対処するため、AES ラップとラップ解除で AES アルゴリズムの代わりに AES Wrap アルゴリズムが適切に使用されるようになりました。つまり、AES/ECB および AES/CBC メカニズムでは、`Cipher.WRAP_MODE` および `Cipher.UNWRAP_MODE` は成功しなくなりました。

  クライアントバージョン 3.1.0 にアップグレードするには、コードを更新する必要があります。既存のラップされたキーがある場合は、ラップ解除に使用するメカニズムと IV のデフォルトの変更方法に特に注意する必要があります。クライアントバージョン 3.0.0 以前でキーをラップした場合、3.1.1 では、AESWrap/ECB/PKCS5Padding を使用して既存のキーをラップ解除する必要があります。詳細については、「AES キーラップ」を参照してください。
- JCE プロバイダーから同じラベルを持つ複数のキーを一覧表示できます。使用可能なすべてのキーを反復処理する方法については、「 ですべてのキーを検索する」を参照してください GitHub。
- パブリックキーとプライベートキーに異なるラベルを指定するなど、キーの作成時に属性に対してより制限的な値を設定できます。詳細については、「サポートされている Java 属性」を参照してください。

Windows (CNG、KSP)

- 安定性の向上およびバグ修正。

E nd-of-life リリース

AWS CloudHSM は、 サービスとの互換性がなくなったリリースのサポート終了を発表しました。アプリケーションの安全性を維持するために、リリースからの接続を積極的に拒否する権利は当社が保持します end-of-life 。

- 現在、クライアント SDK のバージョンは end-of-life リリースされていません。

ドキュメント履歴

このトピックでは、AWS CloudHSM ユーザーガイドの重要な更新について説明します。

トピック

最新の更新

以下の表は、このドキュメントの 2018 年 4 月以降の大きな変更点をまとめたものです。ここに表示されている主要な変更に加えて、その内容の説明と例を向上し、ユーザーから寄せられるフィードバックにも応える目的で、このドキュメントは頻繁に更新されます。重要な変更についての通知を受け取るには、右上隅にあるリンクを使用して、RSS フィードをサブスクライブします。

新しいリリースの詳細については、「Client SDK AWS CloudHSM のダウンロード」を参照してください

変更	説明	日付
新しい HSM タイプとクラスターモード	新しい HSM タイプ (hsm2m.medium) と新しいクラスターモード (非 FIPS) を起動しました。	2024 年 6 月 10 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.12.0 をリリースしました。	2024 年 3 月 20 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.11.0 をリリースしました。	2024 年 1 月 17 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.10.0 をリリースしました。	2023 年 7 月 28 日

新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.9.0 をリリースしました。	2023 年 5 月 23 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.8.0 をリリースしました。	2023 年 3 月 16 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.7.0 をリリースしました。	2022 年 11 月 16 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.6.0 をリリースしました。	2022 年 9 月 1 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.5.0 をリリースしました。	2022 年 5 月 13 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.4.2 をリリースしました。	2022 年 3 月 18 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.4.1 をリリースしました。	2022 年 2 月 10 日
新しいリリースの追加	Windows プラットフォーム用の AWS CloudHSM JCE プロバイダーバージョン 5.4.0 をリリースしました。	2022 年 2 月 1 日

新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.4.0 をリリースしました。これにより、すべての Linux プラットフォームの JCE プロバイダーの初期サポートが追加されました。	2022 年 1 月 28 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.3.0 をリリースしました。	2022 年 1 月 3 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.4.4 をリリースしました。	2022 年 1 月 3 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.4.3 をリリースしました。	2021 年 12 月 20 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.4.2 をリリースしました。	2021 年 12 月 15日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.4.1 をリリースしました。	2021 年 12 月 10 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.2.1 をリリースしました。	2021 年 10 月 4 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.4.0 をリリースしました。	2021 年 8 月 25 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.2.0 をリリースしました。	2021 年 8 月 3 日

新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.3.2 をリリースしました。	2021 年 7 月 2 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.1.0 をリリースしました。	2021 年 6 月 1 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.3.1 をリリースしました。	2021 年 4 月 26 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.0.1 をリリースしました。	2021 年 4 月 8 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 5.0.0 をリリースしました。	2021 年 3 月 12 日
新しいコンテンツの追加	インターネット、NAT デバイス、VPN 接続、または 接続を介したアクセスを必要と AWS CloudHSM せずに、VPC と の間にプライベート接続を作成できる AWS 機能であるインターフェイス VPC エンドポイントを追加しました AWS Direct Connect 。	2021 年 2 月 10 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.3.0 をリリースしました。	2021 年 2 月 3 日
新しいコンテンツの追加	古いバックアップを自動的に削除する機能である、マネージドバックアップ保持機能が追加されました。	2020 年 11 月 18 日

新しいコンテンツの追加	PKCS #11 標準を使用した PKCS #11 ライブラリの AWS CloudHSM Client SDK 3.2.1 実装を分析するコンプライアンスレポートを追加しました。	2020 年 10 月 29 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.2.1 をリリースしました。	2020 年 10 月 8 日
新しいコンテンツの追加	AWS CloudHSMのキーの同期設定について説明したドキュメントを追加しました。	2020 年 9 月 1 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.2.0 をリリースしました。	2020 年 8 月 31 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.1.2 をリリースしました。	2020 年 7 月 30 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.1.1 をリリースしました。	2020 年 6 月 3 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.1.0 をリリースしました。	2020 年 5 月 21 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 3.0.1 をリリースしました。	2020 年 4 月 20 日

新しいリリースの追加	Windows Server プラットフォーム用の AWS CloudHSM クライアントバージョン 3.0.0 をリリースしました。	2019 年 10 月 30 日
新しいリリースの追加	Windows を除くすべてのプラットフォームで AWS CloudHSM クライアントバージョン 3.0.0 がリリースされました。	2019 年 10 月 22 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 2.0.4 をリリースしました。	2019 年 8 月 26 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 2.0.3 をリリースしました。	2019 年 5 月 13 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 2.0.1 をリリースしました。	2019 年 3 月 21 日
新しいリリースの追加	AWS CloudHSM クライアントバージョン 2.0.0 をリリースしました。	2019 年 2 月 6 日
リージョンサポートの追加	EU (ストックホルム) および AWS GovCloud （米国東部) リージョン AWS CloudHSM のサポートを追加しました。	2018 年 12 月 19 日
新しいリリースの追加	Windows 用の AWS CloudHSM クライアントバージョン 1.1.2 をリリースしました。	2018 年 11 月 20 日

更新された既知の問題	トラブルシューティングガイドに新しいコンテンツが追加されました。	2018 年 11 月 8 日
新しいリリースの追加	Linux プラットフォーム用の AWS CloudHSM クライアントバージョン 1.1.2 をリリースしました。	2018 年 11 月 8 日
リージョンサポートの追加	欧州 (パリ) およびアジアパシフィック (ソウル) リージョン AWS CloudHSM のサポートを追加しました。	2018 年 10 月 24 日
新しいコンテンツの追加	AWS CloudHSM バックアップを削除および復元する機能を追加しました。	2018 年 9 月 10 日
新しいコンテンツの追加	Amazon CloudWatch Logs への自動監査ログ配信を追加しました。	2018 年 8 月 13 日
新しいコンテンツの追加	リージョン間で AWS CloudHSM クラスターバックアップをコピーする機能を追加しました。	2018 年 7 月 30 日
リージョンサポートの追加	EU (ロンドン) リージョン AWS CloudHSM のサポートを追加しました。	2018 年 13 月 6 日

新しいコンテンツの追加	Amazon Linux 2、Red Hat Enterprise Linux (RHEL) 6、Red Hat Enterprise Linux (RHEL) 7、CentOS 6、CentOS 7、Ubuntu 16.04 LTS の AWS CloudHSM クライアントとライブラリのサポートが追加されました。	2018 年 5 月 10 日
新しいリリースの追加	Windows AWS CloudHSM クライアントを追加しました。	2018 年 30 月 4 日

以前の更新

次の表は、2018 AWS CloudHSM 年以前の に対する重要な変更点を示しています。

変更	説明	日付
新しいコンテンツ	Crypto User (CO) 用のクォーラム認証 (M of N アクセスコントロール) を追加しました。詳細については、「CloudHSM 管理ユーティリティ (CMU) を使用したクォーラム認証の管理 (M of N アクセスコントロール)」を参照してください。	2017 年 11 月 9 日
更新	key_mgmt_util コマンドラインツールの使用方法についてのドキュメントを追加しました。詳細については、「key_mgmt_util コマンドリファレンス」を参照してください。	2017 年 11 月 9 日

変更	説明	日付
新しいコンテンツ	Oracle Transparent Data Encryption を追加しました。詳細については、「Oracle Database 暗号化」を参照してください。	2017 年 10 月 25 日
新しいコンテンツ	SSL オフロードを追加しました。詳細については、「SSL/TLS のオフロード」を参照してください。	2017 年 10 月 12 日
新規ガイド	このリリースでは、 AWS CloudHSM	2017 年 8 月 14 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。