# Table of Contents

What Is AWS CloudHSM? .......................................................................................................................... 1

Use Cases .................................................................................................................................................. 1

- Offload the SSL/TLS Processing for Web Servers ........................................................................ 1
- Protect the Private Keys for an Issuing Certificate Authority (CA) ................................................ 2
- Enable Transparent Data Encryption (TDE) for Oracle Databases .................................................. 2

Clusters ................................................................................................................................................... 2

- Cluster Architecture .......................................................................................................................... 3
- Cluster Synchronization ...................................................................................................................... 4
- Cluster High Availability and Load Balancing .................................................................................... 5

Backups ................................................................................................................................................... 6

- Overview of Backups ........................................................................................................................ 6
- Security of Backups ........................................................................................................................... 7
- Durability of Backups .......................................................................................................................... 8
- Frequency of Backups ......................................................................................................................... 8

Client Tools and Libraries ....................................................................................................................... 8

- AWS CloudHSM Client .................................................................................................................. 9
- AWS CloudHSM Command Line Tools ......................................................................................... 10
- AWS CloudHSM Software Libraries ............................................................................................... 10

HSM Users ............................................................................................................................................ 10

- Precrypto Officer (PRECO) ............................................................................................................ 11
- Crypto Officer (CO) ........................................................................................................................ 11
- Crypto User (CU) ............................................................................................................................ 11
- Appliance User (AU) ....................................................................................................................... 11
- HSM User Permissions Table ......................................................................................................... 11

Pricing .................................................................................................................................................... 12

Regions .................................................................................................................................................. 12

Quotas ................................................................................................................................................... 13

System Resources .................................................................................................................................. 13

Getting Started .................................................................................................................................... 15

Create IAM Administrators .................................................................................................................. 15

- Create an IAM User and Administrator Group ............................................................................... 15

Create a VPC ......................................................................................................................................... 16

Create a Private Subnet .......................................................................................................................... 17

Create a Cluster ...................................................................................................................................... 17

Review Cluster Security Group ............................................................................................................ 19

Launch an EC2 Client ............................................................................................................................... 19

Connect EC2 Instance to Cluster ......................................................................................................... 20

- Modify the Default Security Group ............................................................................................... 21
- Connect the Amazon EC2 Instance to the AWS CloudHSM Cluster .............................................. 21

Create an HSM ....................................................................................................................................... 22

Verify HSM Identity (Optional) ............................................................................................................ 23

- Overview .......................................................................................................................................... 23
- Get Certificates from the HSM ......................................................................................................... 25
- Get the Root Certificates .................................................................................................................. 27
- Verify Certificate Chains ................................................................................................................... 27
- Extract and Compare Public Keys ...................................................................................................... 28
- AWS CloudHSM Root Certificate ..................................................................................................... 29

Initialize the Cluster .............................................................................................................................. 30

- Get the Cluster CSR ........................................................................................................................ 30
- Sign the CSR ...................................................................................................................................... 32
- Initialize the Cluster .......................................................................................................................... 33

Install the Client (Linux) ......................................................................................................................... 34

- Install the AWS CloudHSM Client and Command Line Tools ....................................................... 34
- Edit the AWS CloudHSM Client and Command Line Tools ............................................................ 36
Using the Software Libraries ............................................................................................................ 214
Command Line Tools ........................................................................................................................ 75
Managing HSM Users and Keys .......................................................................................................... 52
Managing Clusters ............................................................................................................................ 42
Adding or Removing HSMs .................................................................................................................. 42
Adding an HSM ............................................................................................................................. 42
Removing an HSM .......................................................................................................................... 44
Copying a Backup Across Regions ................................................................................................. 45
Creating a Cluster From a Backup .................................................................................................. 46
Deleting and Restoring a Backup .................................................................................................... 47
Deleting a Cluster ............................................................................................................................ 48
Tagging Resources ............................................................................................................................ 49
Adding or Updating Tags .................................................................................................................. 50
Listing Tags .................................................................................................................................... 50
Removing Tags ............................................................................................................................... 51
Managing HSM Users ....................................................................................................................... 52
Create Users .................................................................................................................................... 52
List Users ....................................................................................................................................... 53
Change a User’s Password ............................................................................................................. 54
Delete Users ................................................................................................................................... 54
Managing Keys ................................................................................................................................... 55
Generate Keys .................................................................................................................................... 55
Import Keys ...................................................................................................................................... 56
Export Keys ...................................................................................................................................... 58
Delete Keys ....................................................................................................................................... 59
Share and Unshare Keys ................................................................................................................... 59
Quorum Authentication (M of N) ...................................................................................................... 60
Overview of Quorum Authentication ............................................................................................... 60
Additional Details about Quorum Authentication ......................................................................... 61
First Time Setup for Crypto Officers ............................................................................................. 61
Quorum Authentication for Crypto Officers ................................................................................... 65
Change the Quorum Value for Crypto Officers .............................................................................. 71
Using Trusted Keys to Control Key Unwraps .................................................................................. 72
Background ..................................................................................................................................... 73
Process ........................................................................................................................................... 73
Sample: Generate a Key with an Unwrap Template in PKCS #11 ................................................... 74
Command Line Tools ....................................................................................................................... 75
cloudhsm_mgmt_util ....................................................................................................................... 75
Getting Started ............................................................................................................................... 76
Reference ....................................................................................................................................... 82
key_mgmt_util .............................................................................................................................. 122
Getting Started ............................................................................................................................. 122
Reference ....................................................................................................................................... 125
Configure Tool ................................................................................................................................. 207
Syntax ............................................................................................................................................ 208
Examples ....................................................................................................................................... 208
Parameters ...................................................................................................................................... 211
Related Topics ............................................................................................................................... 213
Using the Software Libraries .......................................................................................................... 214
PKCS #11 Library ............................................................................................................................ 214
Installing the PKCS #11 Library ..................................................................................................... 214
Authenticating to PKCS #11 ........................................................................................................... 220
Supported PKCS #11 Key Types ..................................................................................................... 220
Supported PKCS #11 Mechanisms ................................................................................................. 220
Security ......................................................................................................................................... 331
Getting Metrics .............................................................................................................................. 330
Monitoring Logs ............................................................................................................................. 310
Integrating Third-Party Applications ................................................................................................. 270
Data Protection ...................................................................................................................... 331
Getting CloudWatch Metrics ..................................................................................................... 330
Monitoring Audit Logs ............................................................................................................ 313
Logging AWS CloudHSM API Calls with AWS CloudTrail ............................................................... 311
Encryption at Rest .......................................................................................................... 332
Understanding AWS CloudHSM Log File Entries ......................................................................... 312
Audit Log Reference .............................................................................................................. 327
Getting Client Logs ................................................................................................................. 310
Getting AWS CloudHSM API Calls with AWS CloudTrail ............................................................... 311
Understanding AWS CloudHSM Log File Entries ......................................................................... 312
Monitoring Audit Logs .............................................................................................................. 313
How Audit Logging Works ........................................................................................................... 314
Interpreting HSM Audit Logs ........................................................................................................ 317
Getting Metrics ........................................................................................................................... 330
Getting CloudWatch Metrics ..................................................................................................... 330
Security ......................................................................................................................................... 331
Data Protection ...................................................................................................................... 331
Encryption at Rest .......................................................................................................... 332
Encryption in Transit ............................................................................................................. 332
Identity and Access Management ............................................................................................... 332
Grant Permissions Using IAM Policies .......................................................................................... 332
API Actions for AWS CloudHSM ............................................................................................. 333
Condition Keys for AWS CloudHSM ......................................................................................... 333
Predefined AWS Managed Policies for AWS CloudHSM ............................................................. 334
Customer Managed Policies for AWS CloudHSM ......................................................................... 334
Service-Linked Roles ............................................................................................................... 336
FIPS Validation ............................................................................................................................ 337
What Is AWS CloudHSM?

AWS CloudHSM provides hardware security modules in the AWS Cloud. A hardware security module (HSM) is a computing device that processes cryptographic operations and provides secure storage for cryptographic keys.

When you use an HSM from AWS CloudHSM, you can perform a variety of cryptographic tasks:

- Generate, store, import, export, and manage cryptographic keys, including symmetric keys and asymmetric key pairs.
- Use symmetric and asymmetric algorithms to encrypt and decrypt data.
- Use cryptographic hash functions to compute message digests and hash-based message authentication codes (HMACs).
- Cryptographically sign data (including code signing) and verify signatures.
- Generate cryptographically secure random data.

If you want a managed service for creating and controlling your encryption keys, but you don't want or need to operate your own HSM, consider using AWS Key Management Service.

To learn more about what you can do with AWS CloudHSM, see the following topics. When you are ready to get started with AWS CloudHSM, see Getting Started (p. 15).

Contents
- AWS CloudHSM Use Cases (p. 1)
- AWS CloudHSM Clusters (p. 2)
- AWS CloudHSM Cluster Backups (p. 6)
- AWS CloudHSM Client Tools and Software Libraries (p. 8)
- HSM Users (p. 10)
- Pricing (p. 12)
- Regions (p. 12)
- AWS CloudHSM Quotas (p. 13)

AWS CloudHSM Use Cases

A hardware security module (HSM) in AWS CloudHSM can help you accomplish a variety of goals.

Topics
- Offload the SSL/TLS Processing for Web Servers (p. 1)
- Protect the Private Keys for an Issuing Certificate Authority (CA) (p. 2)
- Enable Transparent Data Encryption (TDE) for Oracle Databases (p. 2)

Offload the SSL/TLS Processing for Web Servers

Web servers and their clients (web browsers) can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS). These protocols confirm the identity of the web server and establish a secure connection...
to send and receive webpages or other data over the internet. This is commonly known as HTTPS. The web server uses a public–private key pair and an SSL/TLS public key certificate to establish an HTTPS session with each client. This process involves a lot of computation for the web server, but you can offload some of this to the HSMs in your AWS CloudHSM cluster. This is sometimes known as SSL acceleration. Offloading reduces the computational burden on your web server and provides extra security by storing the server’s private key in the HSMs.

For information about setting up SSL/TLS offload with AWS CloudHSM, see SSL/TLS Offload (p. 270).

Protect the Private Keys for an Issuing Certificate Authority (CA)

In a public key infrastructure (PKI), a certificate authority (CA) is a trusted entity that issues digital certificates. These digital certificates bind a public key to an identity (a person or organization) by means of public key cryptography and digital signatures. To operate a CA, you must maintain trust by protecting the private key that signs the certificates issued by your CA. You can store the private key in the HSM in your AWS CloudHSM cluster, and use the HSM to perform the cryptographic signing operations.

Enable Transparent Data Encryption (TDE) for Oracle Databases

Some versions of Oracle's database software offer a feature called Transparent Data Encryption (TDE). With TDE, the database software encrypts data before storing it on disk. The data in the database's table columns or tablespaces is encrypted with a table key or tablespace key. These keys are encrypted with the TDE master encryption key. You can store the TDE master encryption key in the HSMs in your AWS CloudHSM cluster, which provides additional security.

For information about setting up Oracle TDE with AWS CloudHSM, see Oracle Database Encryption (p. 302).

AWS CloudHSM Clusters

AWS CloudHSM provides hardware security modules (HSMs) in a cluster. A cluster is a collection of individual HSMs that AWS CloudHSM keeps in sync. You can think of a cluster as one logical HSM. When you perform a task or operation on one HSM in a cluster, the other HSMs in that cluster are automatically kept up to date.

You can create a cluster that has from 1 to 28 HSMs (the default limit (p. 13) is 6 HSMs per AWS account per AWS Region). You can place the HSMs in different Availability Zones in an AWS Region. Adding more HSMs to a cluster provides higher performance. Spreading clusters across Availability Zones provides redundancy and high availability.

Making individual HSMs work together in a synchronized, redundant, highly available cluster can be difficult, but AWS CloudHSM does some of the undifferentiated heavy lifting for you. You can add and remove HSMs in a cluster and let AWS CloudHSM keep the HSMs connected and in sync for you.

To create a cluster, see Getting Started (p. 15).

For more information about clusters, see the following topics.

Topics
- Cluster Architecture (p. 3)
Cluster Architecture

When you create a cluster, you specify an Amazon Virtual Private Cloud (VPC) in your AWS account and one or more subnets in that VPC. We recommend that you create one subnet in each Availability Zone (AZ) in your chosen AWS Region. To learn how, see Create a Private Subnet (p. 17).

Each time you create an HSM, you specify the cluster and Availability Zone for the HSM. By putting the HSMs in different Availability Zones, you achieve redundancy and high availability in case one Availability Zone is unavailable.

When you create an HSM, AWS CloudHSM puts an elastic network interface (ENI) in the specified subnet in your AWS account. The elastic network interface is the interface for interacting with the HSM. The HSM resides in a separate VPC in an AWS account that is owned by AWS CloudHSM. The HSM and its corresponding network interface are in the same Availability Zone.

To interact with the HSMs in a cluster, you need the AWS CloudHSM client software. Typically you install the client on Amazon EC2 instances, known as client instances, that reside in the same VPC as the HSM ENIs, as shown in the following figure. That's not technically required though; you can install the client on any compatible computer, as long as it can connect to the HSM ENIs. The client communicates with the individual HSMs in your cluster through their ENIs.

The following figure represents an AWS CloudHSM cluster with three HSMs, each in a different Availability Zone in the VPC.
Cluster Synchronization

In an AWS CloudHSM cluster, AWS CloudHSM keeps the keys on the individual HSMs in sync. You don’t need to do anything to synchronize the keys on your HSMs. To keep the users and policies on each HSM in sync, update the AWS CloudHSM client configuration file before you manage HSM users (p. 52). For more information, see Keep HSM Users In Sync (p. 348).
When you add a new HSM to a cluster, AWS CloudHSM makes a backup of all keys, users, and policies on an existing HSM. It then restores that backup onto the new HSM. This keeps the two HSMs in sync.

If the HSMs in a cluster fall out of synchronization, AWS CloudHSM automatically resynchronizes them. To enable this, AWS CloudHSM uses the credentials of the appliance user (p. 10). This user exists on all HSMs provided by AWS CloudHSM and has limited permissions. It can get a hash of objects on the HSM and can extract and insert masked (encrypted) objects. AWS cannot view or modify your users or keys and cannot perform any cryptographic operations using those keys.

**Cluster High Availability and Load Balancing**

When you create an AWS CloudHSM cluster with more than one HSM, you automatically get load balancing. Load balancing means that the AWS CloudHSM client (p. 8) distributes cryptographic operations across all HSMs in the cluster based on each HSM's capacity for additional processing.

When you create the HSMs in different AWS Availability Zones, you automatically get high availability. High availability means that you get higher reliability because no individual HSM is a single point of failure. We recommend that you have a minimum of two HSMs in each cluster, with each HSM in different Availability Zones within an AWS Region.

For example, the following figure shows an Oracle database application that is distributed to two different Availability Zones. The database instances store their master keys in a cluster that includes an HSM in each Availability Zone. AWS CloudHSM automatically synchronizes the keys to both HSMs so that they are immediately accessible and redundant.
AWS CloudHSM makes periodic backups of your cluster. You can't instruct AWS CloudHSM to make backups anytime that you want, but you can take certain actions that result in AWS CloudHSM making a backup. For more information, see the following topics.

When you add an HSM to a cluster that previously contained one or more active HSMs, AWS CloudHSM restores the most recent backup onto the new HSM. This means that you can use AWS CloudHSM to manage an HSM that you use infrequently. When you don't need to use the HSM, you can delete it, which triggers a backup. Later, when you need to use the HSM again, you can create a new HSM in the same cluster, effectively restoring your previous HSM.

You can also create a new cluster from an existing backup of a different cluster. You must create the new cluster in the same AWS Region that contains the existing backup.

Topics

- Overview of Backups (p. 6)
- Security of Backups (p. 7)
- Durability of Backups (p. 8)
- Frequency of Backups (p. 8)

Overview of Backups

Each backup contains encrypted copies of the following data:

- All users (COs, CUs, and AUs) (p. 10) on the HSM.
- All key material and certificates on the HSM.
- The HSM's configuration and policies.

AWS CloudHSM stores the backups in a service-controlled Amazon Simple Storage Service (Amazon S3) bucket in the same AWS Region as your cluster.
Security of Backups

When AWS CloudHSM makes a backup from the HSM, the HSM encrypts all of its data before sending it to AWS CloudHSM. The data never leaves the HSM in plaintext form.

To encrypt its data, the HSM uses a unique, ephemeral encryption key known as the ephemeral backup key (EBK). The EBK is an AES 256-bit encryption key generated inside the HSM when AWS CloudHSM makes a backup. The HSM generates the EBK, then uses it to encrypt the HSM's data with a FIPS-approved AES key wrapping method that complies with NIST special publication 800-38F. Then the HSM gives the encrypted data to AWS CloudHSM. The encrypted data includes an encrypted copy of the EBK.

To encrypt the EBK, the HSM uses another encryption key known as the persistent backup key (PBK). The PBK is also an AES 256-bit encryption key. To generate the PBK, the HSM uses a FIPS-approved key derivation function (KDF) in counter mode that complies with NIST special publication 800-108. The inputs to this KDF include the following:

- A manufacturer key backup key (MKBK), permanently embedded in the HSM hardware by the hardware manufacturer.
- An AWS key backup key (AKBK), securely installed in the HSM when it's initially configured by AWS CloudHSM.

The encryption processes are summarized in the following figure. The backup encryption key represents the persistent backup key (PBK) and the ephemeral backup key (EBK).
AWS CloudHSM can restore backups onto only AWS-owned HSMs made by the same manufacturer. Because each backup contains all users, keys, and configuration from the original HSM, the restored HSM contains the same protections and access controls as the original. The restored data overwrites all other data that might have been on the HSM prior to restoration.

A backup consists of only encrypted data. Before each backup is stored in Amazon S3, it’s encrypted again under an AWS Key Management Service (AWS KMS) customer master key (CMK).

**Durability of Backups**

AWS CloudHSM stores cluster backups in an Amazon S3 bucket in an AWS account that AWS CloudHSM controls. The durability of backups is the same as any object stored in Amazon S3. Amazon S3 is designed to deliver 99.999999999% durability.

**Frequency of Backups**

AWS CloudHSM makes a cluster backup at least once per 24 hours. In addition to recurring daily backups, AWS CloudHSM makes a backup when you perform any of the following actions:

- Initialize the cluster (p. 30).
- Add an HSM to an initialized cluster (p. 42).
- Remove an HSM from a cluster (p. 44).

**AWS CloudHSM Client Tools and Software Libraries**

To manage and use the HSMs in your cluster, you use the AWS CloudHSM client software. The client software includes several components, as described in the following topics.

**Topics**

- AWS CloudHSM Client (p. 9)
- AWS CloudHSM Command Line Tools (p. 10)
- AWS CloudHSM Software Libraries (p. 10)
AWS CloudHSM Client

The AWS CloudHSM client is a daemon that you install and run on your application hosts. The client establishes and maintains a secure, end-to-end encrypted connection with the HSMs in your AWS CloudHSM cluster. The client provides the fundamental connection between your application hosts and your HSMs. Most of the other AWS CloudHSM client software components rely on the client to communicate with your HSMs. To get started with the AWS CloudHSM client if you are using Linux, see Install the Client (Linux) (p. 34). If you are using Windows, see Install the Client (Windows) (p. 36).

AWS CloudHSM Client End-to-End Encryption

Communication between the AWS CloudHSM client and the HSMs in your cluster is encrypted from end to end. Only your client and your HSMs can decrypt the communication.

The following process explains how the client establishes end-to-end encrypted communication with an HSM.

1. Your client establishes a Transport Layer Security (TLS) connection with the server that hosts your HSM hardware. Your cluster's security group allows inbound traffic to the server only from client instances in the security group. The client also checks the server's certificate to ensure that it's a trusted server.

2. Next, the client establishes an encrypted connection with the HSM hardware. The HSM has the cluster certificate that you signed with your own certificate authority (CA), and the client has the CA's root certificate. Before the client–HSM encrypted connection is established, the client verifies the HSM's cluster certificate against its root certificate. The connection is established only when the client successfully verifies that the HSM is trusted. The client–HSM encrypted connection goes through the client–server connection established previously.
AWS CloudHSM Command Line Tools

The AWS CloudHSM client software includes two command line tools. You use the command line tools to manage the users and keys on the HSMs. For example, you can create HSM users, change user passwords, create keys, and more. For information about these tools, see Command Line Tools (p. 75).

AWS CloudHSM Software Libraries

You can use the AWS CloudHSM software libraries to integrate your applications with the HSMs in your cluster and use them for cryptoprocessing. For more information about installing and using the different libraries, see Using the Software Libraries (p. 214).

HSM Users

Most operations that you perform on the HSM require the credentials of an HSM user. The HSM authenticates each HSM user by means of a user name and password.

Each HSM user has a type that determines which operations the user is allowed to perform on the HSM. The following topics explain the types of HSM users.

Topics

- Precrypto Officer (PRECO) (p. 11)
- Crypto Officer (CO) (p. 11)
- Crypto User (CU) (p. 11)
- Appliance User (AU) (p. 11)
- HSM User Permissions Table (p. 11)
**Precrypto Officer (PRECO)**

The precrypto officer (PRECO) is a temporary user that exists only on the first HSM in an AWS CloudHSM cluster. The first HSM in a new cluster contains a PRECO user with a default user name and password. To activate a cluster (p. 38), you log in to the HSM and change the PRECO user’s password. When you change the password, the PRECO user becomes a crypto officer (CO). The PRECO user can only change its own password and perform read-only operations on the HSM.

**Crypto Officer (CO)**

A crypto officer (CO) can perform user management operations. For example, a CO can create and delete users and change user passwords. For more information, see the HSM User Permissions Table (p. 11). When you activate a new cluster (p. 38), the user changes from a Precrypto Officer (p. 11) (PRECO) to a crypto officer (CO).

**Crypto User (CU)**

A crypto user (CU) can perform the following key management and cryptographic operations.

- **Key management** – Create, delete, share, import, and export cryptographic keys.
- **Cryptographic operations** – Use cryptographic keys for encryption, decryption, signing, verifying, and more.

For more information, see the HSM User Permissions Table (p. 11).

**Appliance User (AU)**

The appliance user (AU) can perform cloning and synchronization operations. AWS CloudHSM uses the AU to synchronize the HSMs in an AWS CloudHSM cluster. The AU exists on all HSMs provided by AWS CloudHSM, and has limited permissions. For more information, see the HSM User Permissions Table (p. 11).

AWS uses the AU to perform cloning and synchronization operations on your cluster’s HSMs. AWS cannot perform any operations on your HSMs except those granted to the AU and unauthenticated users. AWS cannot view or modify your users or keys and cannot perform any cryptographic operations using those keys.

**HSM User Permissions Table**

The following table lists HSM operations and whether each type of HSM user can perform them.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Crypto Officer (CO)</th>
<th>Crypto User (CU)</th>
<th>Appliance User (AU)</th>
<th>Unauthenticated User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get basic cluster info¹</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Zeroize an HSM²</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change own password</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Change any user’s password</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Pricing

With AWS CloudHSM, you pay by the hour with no long-term commitments or upfront payments. For more information, see [AWS CloudHSM Pricing](https://aws.amazon.com/cloudhsm/pricing/) on the AWS website.

### Regions

For information about the supported Regions for AWS CloudHSM, see [AWS CloudHSM Regions and Endpoints](https://docs.aws.amazon.com/cloudhsm/latest/userguide/regions-availability.html) in the *AWS General Reference*, or the [Region Table](https://aws.amazon.com/about-aws/global-infrastructure/region-info/). Like most AWS resources, clusters and HSMs are regional resources. To create HSMs in multiple Regions, you must first create a cluster in each Region. You cannot reuse or extend a cluster across Regions. You must perform all the required steps listed in [Getting Started with AWS CloudHSM](https://docs.aws.amazon.com/cloudhsm/latest/userguide/getting-started.html) to create a cluster in a new Region.

AWS CloudHSM might not be available in all Availability Zones in a given Region. However, this should not affect performance, as AWS CloudHSM automatically load balances across all HSMs in a cluster.

---

<table>
<thead>
<tr>
<th></th>
<th>Crypto Officer (CO)</th>
<th>Crypto User (CU)</th>
<th>Appliance User (AU)</th>
<th>Unauthenticated User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, remove users</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Get sync status³</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Extract, insert masked objects⁴</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Key management functions⁵</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Encrypt, decrypt</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sign, verify</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Generate digests and HMACs</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

¹Basic cluster information includes the number of HSMs in the cluster and each HSM’s IP address, model, serial number, device ID, firmware ID, etc.

²When an HSM is zeroized, all keys, certificates, and other data on the HSM is destroyed. You can use your cluster's security group to prevent an unauthenticated user from zeroizing your HSM. For more information, see [Create a Cluster](https://docs.aws.amazon.com/cloudhsm/latest/userguide/create-cluster.html) (p. 17).

³The user can get a set of digests (hashes) that correspond to the keys on the HSM. An application can compare these sets of digests to understand the synchronization status of HSMs in a cluster.

⁴Masked objects are keys that are encrypted before they leave the HSM. They cannot be decrypted outside of the HSM. They are only decrypted after they are inserted into an HSM that is in the same cluster as the HSM from which they were extracted. An application can extract and insert masked objects to synchronize the HSMs in a cluster.

⁵Key management functions include creating, deleting, wrapping, unwrapping, and modifying the attributes of keys.
AWS CloudHSM Quotas

Quotas, formerly known as limits, are the assigned values for AWS resources. The following quotas apply to your AWS CloudHSM resources per AWS Region and AWS account. The default quota is the initial value applied by AWS, and these values are listed in the table below. An adjustable quota can be increased above the default quota.

### Service Quotas

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default Quota</th>
<th>Adjustable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusters</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>HSMs</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>HSMs per cluster</td>
<td>28</td>
<td>No</td>
</tr>
</tbody>
</table>

The quotas in the following System Quotas table are not adjustable.

### System Quotas

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys per cluster</td>
<td>3,300</td>
</tr>
<tr>
<td>Number of users per cluster</td>
<td>1,024</td>
</tr>
<tr>
<td>Maximum length of a user name</td>
<td>31 characters</td>
</tr>
<tr>
<td>Required password length</td>
<td>7 to 32 characters</td>
</tr>
<tr>
<td>Maximum number of concurrent clients</td>
<td>900</td>
</tr>
</tbody>
</table>

The recommended way of requesting a quota increase is to open the Service Quotas console. In the console, choose your service and quota, and submit your request. For more information, see the Service Quotas documentation.

### System Resources

System resource quotas are quotas on what the AWS CloudHSM client is allowed to use when it runs.

File descriptors are an operating system's mechanism to identify and manage open files on a per-process basis.

The CloudHSM client daemon utilizes file descriptors to manage connections between applications and the client, as well as between the client and the server.

By default, the CloudHSM client configuration will allocate 3000 file descriptors. This default value is designed to yield an optimal session and threading capacity between the client daemon and your HSMs.

In rare circumstances, if you are running your client in a restricted-resource environment, it may become necessary to alter these default values.

**Note**

By changing these values, your CloudHSM client performance may suffer and/or your application may become inoperable.
1. Edit the `/etc/security/limits.d/cloudhsm.conf` file.

```bash
# DO NOT EDIT THIS FILE
#
hsmuser soft nofile 3000
hsmuser hard nofile 3000
```

2. Edit the numeric values, as needed.

   **Note**
   The `soft` quota must be less than or equal to the `hard` quota.

3. Restart your CloudHSM client daemon process.

   **Note**
   This configuration option is not available on Microsoft Windows platforms.
Getting Started with AWS CloudHSM

The following helps you create, initialize, and activate an AWS CloudHSM cluster. After you complete these procedures, you'll be ready to manage users, manage clusters, and use the included software libraries to perform cryptographic operations.

Contents

• Create IAM Administrative Groups (p. 15)
• Create a Virtual Private Cloud (VPC) (p. 16)
• Create a Private Subnet (p. 17)
• Create a Cluster (p. 17)
• Review Cluster Security Group (p. 19)
• Launch an Amazon EC2 Client Instance (p. 19)
• Connect Amazon EC2 Instance to AWS CloudHSM Cluster (p. 20)
• Create an HSM (p. 22)
• Verify the Identity and Authenticity of Your Cluster’s HSM (Optional) (p. 23)
• Initialize the Cluster (p. 30)
• Install and Configure the AWS CloudHSM Client (Linux) (p. 34)
• Install and Configure the AWS CloudHSM Client (Windows) (p. 36)
• Activate the Cluster (p. 38)
• Reconfigure SSL with a New Certificate and Private Key (Optional) (p. 39)
• Best Practices for AWS CloudHSM (p. 41)

Create IAM Administrative Groups

As a best practice, don't use your AWS account root user to interact with AWS, including AWS CloudHSM. Instead, use AWS Identity and Access Management (IAM) to create an IAM user, IAM role, or federated user. Follow the steps in the Create an IAM User and Administrator Group (p. 15) section to create an administrator group and attach the AdministratorAccess policy to it. Then create a new administrator user and add the user to the group. Add additional users to the group as needed. Each user you add inherits the AdministratorAccess policy from the group.

Another best practice is to create an AWS CloudHSM administrator group that has only the permissions required to run AWS CloudHSM. Add individual users to this group as needed. Each user inherits the limited permissions that are attached to the group rather than full AWS access. The Customer Managed Policies for AWS CloudHSM (p. 334) section that follows contains the policy that you should attach to your AWS CloudHSM administrator group.

AWS CloudHSM defines an service–linked role for your AWS account. The service–linked role currently defines permissions that allow your account to log AWS CloudHSM events. The role can be created automatically by AWS CloudHSM or manually by you. You cannot edit the role, but you can delete it. For more information, see Service-Linked Roles for AWS CloudHSM (p. 336).

Create an IAM User and Administrator Group

Start by creating an IAM user along with an administrator group for that user.
To create an administrator user for yourself and add the user to an administrators group (console)

1. Use your AWS account email address and password to sign in as the AWS account root user to the IAM console at https://console.aws.amazon.com/iam/.

   **Note**
   We strongly recommend that you adhere to the best practice of using the Administrator IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

2. In the navigation pane, choose **Users** and then choose **Add user**.

3. For **User name**, enter **Administrator**.

4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.

5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.

6. Choose **Next: Permissions**.

7. Under **Set permissions**, choose **Add user to group**.

8. Choose **Create group**.

9. In the **Create group** dialog box, for **Group name** enter **Administrators**.

10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.

11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

   **Note**
   You must activate IAM user and role access to Billing before you can use the AdministratorAccess permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.

12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.

13. Choose **Next: Tags**.

14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the IAM User Guide.

15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access Management and Example Policies.

For example policies for AWS CloudHSM that you can attach to your IAM user group, see Identity and Access Management for AWS CloudHSM (p. 332).

Create a Virtual Private Cloud (VPC)

If you don't already have a virtual private cloud (VPC), create one now.

**To create a VPC**

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the navigation bar, use the region selector to choose one of the AWS Regions where AWS CloudHSM is currently supported.
3. Choose **Start VPC Wizard**.
4. Choose the first option, **VPC with a Single Public Subnet**. Then choose **Select**.
5. For **VPC name**, type an identifiable name such as `CloudHSM`. For **Subnet name**, type an identifiable name such as `CloudHSM public subnet`. Leave all other options set to their defaults. Then choose **Create VPC**. After the VPC is created, choose **OK**.

---

**Create a Private Subnet**

Create a private subnet (a subnet with no internet gateway attached) for each Availability Zone where you want to create an HSM. Private subnets are available across all AWS Availability Zones. Even if AWS CloudHSM is not supported in a certain Availability Zone, the HSM cluster still performs as expected if support is added later. Creating a private subnet in each Availability Zone provides the most robust configuration for high availability. Visit [AWS Regions and Endpoints](https://aws.amazon.com/regions) in the [AWS General Reference](https://docs.aws.amazon.com/gn-ref/latest/en_US/contents/index.html) or the [AWS Region Table](https://aws.amazon.com/regions) to see the regional and zone availability for AWS CloudHSM.

**To create the private subnets in your VPC**

1. Open the Amazon VPC console at [https://console.aws.amazon.com/vpc/](https://console.aws.amazon.com/vpc/).
2. In the navigation pane, choose **Subnets**. Then choose **Create Subnet**.
3. In the **Create Subnet** dialog box, do the following:
   a. For **Name tag**, type an identifiable name such as `CloudHSM private subnet`.
   b. For **VPC**, choose the VPC that you created previously.
   c. For **Availability Zone**, choose the first Availability Zone in the list.
   d. For **CIDR block**, type the CIDR block to use for the subnet. If you used the default values for the VPC in the previous procedure, then type `10.0.1.0/28`.
   
   Choose **Yes, Create**.
4. Repeat steps 2 and 3 to create subnets for each remaining Availability Zone in the region. For the subnet CIDR blocks, you can use `10.0.2.0/28`, `10.0.3.0/28`, and so on.

---

**Create a Cluster**

A cluster is a collection of individual HSMs. AWS CloudHSM synchronizes the HSMs in each cluster so that they function as a logical unit.

**Important**

When you create a cluster, AWS CloudHSM creates a service-linked role named AWSSTServiceRoleForCloudHSM. If AWS CloudHSM cannot create the role or the role does not already exist, you may not be able to create a cluster. For more information, see [Resolving Cluster Creation Failures](https://docs.aws.amazon.com/cloudhsm/latest/APIReference/API_CreateCluster.html). For more information about service-linked roles, see [Service-Linked Roles for AWS CloudHSM](https://docs.aws.amazon.com/cloudhsm/latest/APIReference/API_CreateCluster.html).

When you create a cluster, AWS CloudHSM creates a security group for the cluster on your behalf. This security group controls network access to the HSMs in the cluster. It allows inbound connections only from Amazon Elastic Compute Cloud (Amazon EC2) instances that are in the security group. By default, the security group doesn't contain any instances. Later, you launch a client instance (p. 19) and configure the cluster's security group (p. 19) to allow communication and connections with the HSM.

You can create a cluster from the [AWS CloudHSM console](https://aws.amazon.com/cloudhsm/), the [AWS Command Line Interface (AWS CLI)](https://docs.aws.amazon.com/cli/latest/userguide/), or the [AWS CloudHSM API](https://docs.aws.amazon.com/cloudhsm/latest/APIReference/).
To create a cluster (console)

2. On the navigation bar, use the region selector to choose one of the AWS Regions where AWS CloudHSM is currently supported.
3. Choose Create cluster.
4. In the Cluster configuration section, do the following:
   a. For VPC, select the VPC that you created.
   b. For AZ(s), next to each Availability Zone, choose the private subnet that you created.
   
   Note
   Even if AWS CloudHSM is not supported in a given Availability Zone, performance should not be affected, as AWS CloudHSM automatically load balances across all HSMs in a cluster. See AWS CloudHSM Regions and Endpoints in the AWS General Reference to see Availability Zone support for AWS CloudHSM.
5. Choose Next: Review.
6. Review your cluster configuration, and then choose Create cluster.

To create a cluster (AWS CLI)

- At a command prompt, run the create-cluster command. Specify the HSM instance type and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you created. Specify only one subnet per Availability Zone.

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm1.medium --subnet-ids <subnet ID 1> <subnet ID 2> <subnet ID N>
```

```
{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "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",
        "CreateTimestamp": 1502423370.069
    }
}
```

To create a cluster (AWS CloudHSM API)

- Send a CreateCluster request. Specify the HSM instance type and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you created. Specify only one subnet per Availability Zone.

If your attempts to create a cluster fail, it might be related to problems with the AWS CloudHSM service-linked roles. For help on resolving the failure, see Resolving Cluster Creation Failures (p. 351).
Review Cluster Security Group

When you create a cluster, AWS CloudHSM creates a security group with the name `cloudhsm-cluster-clusterID-sg`. This security group contains a preconfigured TCP rule that allows inbound and outbound communication within the cluster security group over ports 2223-2225. This rule allows HSMs in your cluster to communicate with each other.

**Warning**

Note the following:

- Do not delete or modify the preconfigured TCP rule, which is populated in the cluster security group. This rule can prevent connectivity issues and unauthorized access to your HSMs.
- The cluster security group prevents unauthorized access to your HSMs. Anyone that can access instances in the security group can access your HSMs. Most operations require a user to log in to the HSM. However, it's possible to zeroize HSMs without authentication, which destroys the key material, certificates, and other data. If this happens, data created or modified after the most recent backup is lost and unrecoverable. To prevent the unauthorized access, ensure that only trusted administrators can modify or access the instances in the default security group.

In the next step, you can launch an Amazon EC2 instance (p. 19) and connect it to your HSMs by attaching the cluster security group (p. 20) to it.

Launch an Amazon EC2 Client Instance

To interact with and manage your AWS CloudHSM cluster and HSM instances, you must be able to communicate with the elastic network interfaces of your HSMs. The easiest way to do this is to use an EC2 instance in the same VPC as your cluster. You can also use the following AWS resources to connect to your cluster:

- Amazon VPC Peering
- AWS Direct Connect
- VPN Connections

The AWS CloudHSM documentation typically assumes that you are using an EC2 instance in the same VPC and Availability Zone (AZ) in which you create your cluster.

**To create an EC2 instance**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the **EC2 Dashboard**, choose **Launch Instance**.
3. Choose **Select** for an Amazon Machine Image (AMI). Choose a Linux AMI or a Windows Server AMI.
4. Choose an instance type and then choose **Next: Configure Instance Details**.
5. For **Network**, choose the VPC that you previously created for your cluster.
6. For **Subnet**, choose the public subnet that you created for the VPC.
7. For **Auto-assign Public IP**, choose **Enable**.
8. Choose **Next: Add Storage** and configure your storage.
9. Choose **Next: Add Tags** and add any name–value pairs that you want to associate with the instance. We recommend that you at least add a name. Choose **Add Tag** and type a name for the **Key** and up to 255 characters for the **Value**.
10. Choose **Next: Configure Security Group**
11. For **Assign a security group**, choose **Select an existing security group**.
12. Choose the default Amazon VPC security group from the list.
13. Choose **Review and Launch**.

On the **Review Instance Launch** page, choose **Launch**.

14. When prompted for a key pair, choose **Create a new key pair**, enter a name for the key pair, and then choose **Download Key Pair**. This is the only chance for you to save the private key file, so download it and store it in a safe place. You must provide the name of your key pair when you launch an instance. In addition, you must provide the corresponding private key each time that you connect to the instance. Then choose the key pair that you created when getting set up.

Alternatively, you can use an existing key pair. Choose **Choose an existing key pair**, and then choose the desired key pair.

**Warning**
Don’t choose **Proceed without a key pair**. If you launch your instance without a key pair, you won’t be able to connect to it.

When you are ready, select the acknowledgement check box, and then choose **Launch Instances**.

For more information about creating a Linux Amazon EC2 client, see **Getting Started with Amazon EC2 Linux Instances**. For information about connecting to the running client, see the following topics:

- **Connecting to Your Linux Instance Using SSH**
- **Connecting to Your Linux Instance from Windows Using PuTTY**

The Amazon EC2 user guide contains detailed instructions for setting up and using your Amazon EC2 instances. The following list provides an overview of available documentation for Linux and Windows Amazon EC2 clients:

- **To create a Linux Amazon EC2 client**, see **Getting Started with Amazon EC2 Linux Instances**.

  For information about connecting to the running client, see the following topics:

  - **Connecting to your Linux Instance Using SSH**
  - **Connecting to Your Linux Instance from Windows Using PuTTY**

- **To create a Windows Amazon EC2 client**, see **Getting Started with Amazon EC2 Windows Instances**. For more information about connecting to your Windows client, see **Connect to Your Windows Instance**.

**Note**

Your EC2 instance can run all of the AWS CLI commands contained in this guide. If the AWS CLI is not installed, you can download it from **AWS Command Line Interface**. If you are using Windows, you can download and run a 64-bit or 32-bit Windows installer. If you are using Linux or macOS, you can install the CLI using pip.

---

**Connect Amazon EC2 Instance to AWS CloudHSM Cluster**

When you launched an Amazon EC2 instance, you associated it with a default Amazon VPC security group. This topic explains how to associate the cluster security group with the EC2 instance. This association allows the AWS CloudHSM client running on your EC2 instance to communicate with your
HSMs. To connect your EC2 instance to your AWS CloudHSM cluster, you must properly configure the VPC default security group and associate the cluster security group with the instance.

**Modify the Default Security Group**

You need to modify the default security group to permit the SSH or RDP connection so that you can download and install client software, and interact with your HSM.

**To modify the default security group**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the Amazon EC2 dashboard, select the check box for the EC2 instance on which you want to install the AWS CloudHSM client.
3. Under the **Description** tab, choose the security group named **Default**.
4. At the top of the page, choose **Actions**, and then **Edit Inbound Rules**.
5. Select **Add Rule**.
6. For **Type**, do one of the following:
   - For a Windows Server Amazon EC2 instance, choose **RDP**. The port range **3389** is automatically populated.
   - For a Linux Amazon EC2 instance, choose **SSH**. The port range **22** is automatically populated.
7. For either option, set **Source** to **My IP** to allow the client to communicate with the AWS CloudHSM cluster.

   **Important**
   Do not specify 0.0.0.0/0 as the port range to avoid allowing anyone to access your instance.
8. Choose **Save**.

**Connect the Amazon EC2 Instance to the AWS CloudHSM Cluster**

You must attach the cluster security group to the EC2 instance so that the EC2 instance can communicate with HSMs in your cluster. The cluster security group contains a preconfigured rule that allows inbound communication over ports 2223-2225.

**To connect the EC2 instance to the AWS CloudHSM cluster**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the Amazon EC2 dashboard, select the check box for the EC2 instance on which you want to install the AWS CloudHSM client.
3. At the top of the page, choose **Actions**, **Networking**, and then **Change Security Groups**.
4. Select the security group with the group name that matches your cluster ID, such as `cloudhsm-cluster-clusterID-sg`.
5. Choose **Assign Security Groups**.

**Note**
You can assign a maximum of five security groups to an Amazon EC2 instance. If you have reached the maximum limit, you must modify the default security group of the Amazon EC2 instance and the cluster security group:

In the default security group, do the following:
Create an HSM

- Add an outbound rule to permit traffic on all ports to 0.0.0.0/0.
- Add an inbound rule to permit traffic using the TCP protocol over ports 2223–2225 from the cluster security group.

In the cluster security group, do the following:

- Add an outbound rule to permit traffic on all ports to 0.0.0.0/0.
- Add an inbound rule to permit traffic using the TCP protocol over ports 2223–2225 from the default security group.

Create an HSM

After you create a cluster, you can create an HSM. However, before you can create an HSM in your cluster, the cluster must be in the uninitialized state. To determine the cluster's state, view the clusters page in the AWS CloudHSM console, use the AWS CLI to run the describe-clusters command, or send a DescribeClusters request in the AWS CloudHSM API. You can create an HSM from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

**To create an HSM (console)**

2. Choose Initialize next to the cluster that you created previously.
3. Choose an Availability Zone (AZ) for the HSM that you are creating. Then choose Create.

**To create an HSM (AWS CLI)**

- At a command prompt, run the create-hsm command. Specify the cluster ID of the cluster that you created previously and an Availability Zone for the HSM. Specify the Availability Zone in the form of us-west-2a, us-west-2b, etc.

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

```json
{
    "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"
    }
}
```

**To create an HSM (AWS CloudHSM API)**

- Send a CreateHsm request. Specify the cluster ID of the cluster that you created previously and an Availability Zone for the HSM.

After you create a cluster and HSM, you can optionally verify the identity of the HSM (p. 23), or proceed directly to Initialize the Cluster (p. 30).
Verify the Identity and Authenticity of Your Cluster's HSM (Optional)

To initialize your cluster, you sign a certificate signing request (CSR) generated by the cluster's first HSM. Before you do this, you might want to verify the identity and authenticity of the HSM.

Note
This process is optional. However, it works only until a cluster is initialized. After the cluster is initialized, you cannot use this process to get the certificates or verify the HSMs.

Topics
- Overview (p. 23)
- Get Certificates from the HSM (p. 25)
- Get the Root Certificates (p. 27)
- Verify Certificate Chains (p. 27)
- Extract and Compare Public Keys (p. 28)
- AWS CloudHSM Root Certificate (p. 29)

Overview

To verify the identity of your cluster's first HSM, complete the following steps:

1. Get the certificates and CSR (p. 25) – In this step, you get three certificates and a CSR from the HSM. You also get two root certificates, one from AWS CloudHSM and one from the HSM hardware manufacturer.

2. Verify the certificate chains (p. 27) – In this step, you construct two certificate chains, one to the AWS CloudHSM root certificate and one to the manufacturer root certificate. Then you verify the HSM certificate with these certificate chains to determine that AWS CloudHSM and the hardware manufacturer both attest to the identity and authenticity of the HSM.

3. Compare public keys (p. 28) – In this step, you extract and compare the public keys in the HSM certificate and the cluster CSR, to ensure that they are the same. This should give you confidence that the CSR was generated by an authentic, trusted HSM.

The following diagram shows the CSR, the certificates, and their relationship to each other. The subsequent list defines each certificate.
**AWS Root Certificate**

This is AWS CloudHSM's root certificate. You can view and download this certificate at [https://docs.aws.amazon.com/cloudhsm/latest/userguide/root-certificate.html](https://docs.aws.amazon.com/cloudhsm/latest/userguide/root-certificate.html) (p. 29).

**Manufacturer Root Certificate**

This is the hardware manufacturer's root certificate. You can view and download this certificate at [https://www.cavium.com/LS/TAmanuCert/](https://www.cavium.com/LS/TAmanuCert/).

**AWS Hardware Certificate**

AWS CloudHSM created this certificate when the HSM hardware was added to the fleet. This certificate asserts that AWS CloudHSM owns the hardware.

**Manufacturer Hardware Certificate**

The HSM hardware manufacturer created this certificate when it manufactured the HSM hardware. This certificate asserts that the manufacturer created the hardware.
HSM Certificate

The HSM certificate is generated by the FIPS-validated hardware when you create the first HSM in the cluster. This certificate asserts that the HSM hardware created the HSM.

Cluster CSR

The first HSM creates the cluster CSR. When you sign the cluster CSR (p. 32), you claim the cluster. Then, you can use the signed CSR to initialize the cluster (p. 33).

Get Certificates from the HSM

To verify the identity and authenticity of your HSM, start by getting a CSR and five certificates. You get three of the certificates from the HSM, which you can do with the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To get the CSR and HSM certificates (console)

2. Choose Initialize next to the cluster that you created previously.
3. When the certificates and CSR are ready, you see links to download them.
Certificate signing request

To initialize the cluster, you must download a certificate signing request (CSR) and then sign it.

To get the CSR and HSM certificates (AWS CLI)

- At a command prompt, run the `describe-clusters` command four times, extracting the CSR and different certificates each time and saving them to files.
  
  a. Issue the following command to extract the cluster CSR. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```bash
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> --output text --query 'Clusters[].Certificates.ClusterCsr' > <cluster ID>_ClusterCsr.csr
```
b. Issue the following command to extract the HSM certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

c. Issue the following command to extract the AWS hardware certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

d. Issue the following command to extract the manufacturer hardware certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

To get the CSR and HSM certificates (AWS CloudHSM API)

- Send a `DescribeClusters` request, then extract and save the CSR and certificates from the response.

Get the Root Certificates

Follow these steps to get the root certificates for AWS CloudHSM and the manufacturer. Save the root certificate files to the directory that contains the CSR and HSM certificate files.

To get the AWS CloudHSM and manufacturer root certificates

1. Go to https://docs.aws.amazon.com/cloudhsm/latest/userguide/root-certificate.html (p. 29), and then choose `AWS_CloudHSM_ROOT-G1.zip`. After you download the file, extract (unzip) its contents.
2. Go to https://www.cavium.com/LS/TAmanuCert/, and then choose `Download Certificate`. You might need to right-click the `Download Certificate` link and then choose `Save Link As...` to save the certificate file.

Verify Certificate Chains

In this step, you construct two certificate chains, one to the AWS CloudHSM root certificate and one to the manufacturer root certificate. Then use OpenSSL to verify the HSM certificate with each certificate chain.

To create the certificate chains, open a Linux shell. You need OpenSSL, which is available in most Linux shells, and you need the root certificate (p. 27) and HSM certificate files (p. 25) that you downloaded. However, you do not need the AWS CLI for this step, and the shell does not need to be associated with your AWS account.
Extract and Compare Public Keys

To verify the HSM certificate with the AWS CloudHSM root certificate

1. Navigate to the directory where you saved the root certificate (p. 27) and HSM certificate files (p. 25) that you downloaded. The following commands assume that all of the certificates are in the current directory and use the default file names.

Use the following command to create a certificate chain that includes the AWS hardware certificate and the AWS CloudHSM root certificate, in that order. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

2. Use the following OpenSSL command to verify the HSM certificate with the AWS certificate chain. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

To verify the HSM certificate with the manufacturer root certificate

1. Use the following command to create a certificate chain that includes the manufacturer hardware certificate and the manufacturer root certificate, in that order. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```bash
$ cat <cluster ID>_ManufacturerHardwareCertificate.crt cavium_cert.crt > <cluster ID>_manufacturer_chain.crt
```

2. Use the following OpenSSL command to verify the HSM certificate with the manufacturer certificate chain. Replace `<cluster ID>` with the ID of the cluster that you created previously.

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

Extract and Compare Public Keys

Use OpenSSL to extract and compare the public keys in the HSM certificate and the cluster CSR, to ensure that they are the same.

To compare the public keys, use your Linux shell. You need OpenSSL, which is available in most Linux shells, but you do not need the AWS CLI for this step. The shell does not need to be associated with your AWS account.

To extract and compare the public keys

1. Use the following command to extract the public key from the HSM certificate.
2. Use the following command to extract the public key from the cluster CSR.

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

3. Use the following command to compare the public keys. If the public keys are identical, the following command produces no output.

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

After you verify the identity and authenticity of the HSM, proceed to Initialize the Cluster (p. 30).

**AWS CloudHSM Root Certificate**

Download the AWS CloudHSM root certificate: `AWS_CloudHSM_Root-G1.zip`.

Certificate:

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 3 (0x2)</td>
</tr>
<tr>
<td>Serial Number: 17952736724058547791 (0xf924eeeecf9ea64f)</td>
</tr>
<tr>
<td>Signature Algorithm: sha256WithRSAEncryption</td>
</tr>
<tr>
<td>Issuer: C=US, ST=Virginia, L=Herndon,</td>
</tr>
<tr>
<td>O=Amazon Web Services INC., OU=CloudHSM,</td>
</tr>
<tr>
<td>CN=AWS CloudHSM Root G1</td>
</tr>
<tr>
<td>Validity</td>
</tr>
<tr>
<td>Not Before: Apr 28 08:37:46 2017 GMT</td>
</tr>
<tr>
<td>Not After : Apr 26 08:37:46 2027 GMT</td>
</tr>
<tr>
<td>Subject: C=US, ST=Virginia, L=Herndon,</td>
</tr>
<tr>
<td>O=Amazon Web Services INC., OU=CloudHSM,</td>
</tr>
<tr>
<td>CN=AWS CloudHSM Root G1</td>
</tr>
<tr>
<td>Subject Public Key Info:</td>
</tr>
<tr>
<td>Public Key Algorithm: rsaEncryption</td>
</tr>
<tr>
<td>Public-Key: (2048 bit)</td>
</tr>
<tr>
<td>Modulus:</td>
</tr>
<tr>
<td>00:c8:e3:f6:2a:e0:1f:1e:66:73:00:1e:57:dc:3e:</td>
</tr>
<tr>
<td>84:bb:00:dc:57:d8:48:94:5c:13:7a:ff:3b:37:52:</td>
</tr>
<tr>
<td>3f:ce:1c:2f:b5:1e:0f:4f:15:60:27:00:23:67:d5:</td>
</tr>
</tbody>
</table>
Initialize the Cluster

Complete the steps in the following topics to initialize your AWS CloudHSM cluster.

Note
Before you initialize the cluster, review the process by which you can verify the identity and authenticity of the HSMs (p. 23). This process is optional and works only until a cluster is initialized. After the cluster is initialized, you cannot use this process to get your certificates or verify the HSMs.

Topics
- Get the Cluster CSR (p. 30)
- Sign the CSR (p. 32)
- Initialize the Cluster (p. 33)

Get the Cluster CSR

Before you can initialize the cluster, you must download and sign a certificate signing request (CSR) that is generated by the cluster's first HSM. If you followed the steps to verify the identity of your cluster's HSM (p. 23), you already have the CSR and you can sign it. Otherwise, get the CSR now by using the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To get the CSR (console)

2. Choose Initialize next to the cluster that you created previously (p. 17).
3. When the CSR is ready, you see a link to download it.
Certificate signing request

To initialize the cluster, you must download a certificate signing request (CSR) and then sign it.

Choose Cluster CSR to download and save the CSR.

Cluster verification certificate

Optionally, you may wish to download the HSM certificate below, which generated this Cluster CSR and verify its authenticity.

Choose Cluster CSR to download and save the CSR.

To get the CSR (AWS CLI)

- At a command prompt, run the following `describe-clusters` command, which extracts the CSR and saves it to a file. Replace `<cluster ID>` with the ID of the cluster that you created previously (p. 17).

```bash
# aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
   --output text \ 
   --query 'Clusters[].Certificates.ClusterCsr' \ 
   > <cluster ID>_ClusterCsr.csr
```
To get the CSR (AWS CloudHSM API)
1. Send a DescribeClusters request.
2. Extract and save the CSR from the response.

Sign the CSR

Currently, you must create a self-signed signing certificate and use it to sign the CSR for your cluster. You do not need the AWS CLI for this step, and the shell does not need to be associated with your AWS account. To sign the CSR, you must do the following:
1. Get the CSR (see Get the Cluster CSR (p. 30)).
2. Create a private key.
3. Use the private key to create a signing certificate.
4. Sign your cluster CSR.

Create a private key

Use the following command to create a private key. For a production cluster, the key should be created in a secure manner using a trusted source of randomness. We recommend that you use a secured offsite and offline HSM or the equivalent. Store the key safely. If you can demonstrate that you own the key, you can also demonstrate that you own the cluster and the data it contains.

During development and test, you can use any convenient tool (such as OpenSSL) to create and sign the cluster certificate. The following example shows you how to create a key. After you have used the key to create a self-signed certificate (see below), you should store it in a safe manner. To sign into your AWS CloudHSM instance, the certificate must be present, but the private key does not. You use the key only for specific purposes such as restoring from a backup.

```
$ 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:
```

Use the private key to create a self-signed certificate

The trusted hardware that you use to create the private key for your production cluster should also provide a software tool to generate a self-signed certificate using that key. The following example uses OpenSSL and the private key that you created in the previous step to create a signing certificate. The certificate is valid for 10 years (3652 days). Read the on-screen instructions and follow the prompts.

```
$ 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 []:

This command creates a certificate file named customerCA.crt. Put this certificate on every host from which you will connect to your AWS CloudHSM cluster. If you give the file a different name or store it in a path other than the root of your host, you should edit your client configuration file accordingly. Use the certificate and the private key you just created to sign the cluster certificate signing request (CSR) in the next step.

**Sign the Cluster CSR**

The trusted hardware that you use to create your private key for your production cluster should also provide a tool to sign the CSR using that key. The following example uses OpenSSL to sign the cluster's CSR. The example uses your private key and the self-signed certificate that you created in the previous step.

```bash
$ 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 identifier>:PARTN:<partition number>, for FIPS mode
Getting CA Private Key
Enter pass phrase for customerCA.key:

This command creates a file named <cluster ID>_CustomerHsmCertificate.crt. Use this file as the signed certificate when you initialize the cluster.

**Initialize the Cluster**

Use your signed HSM certificate and your signing certificate to initialize your cluster. You can use the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

**To initialize a cluster (console)**

2. Choose Initialize next to the cluster that you created previously.
3. On the Download certificate signing request page, choose Next. If Next is not available, first choose one of the CSR or certificate links. Then choose Next.
4. On the Sign certificate signing request (CSR) page, choose Next.
5. On the Upload the certificates page, do the following:
   a. Next to Cluster certificate, choose Upload file. Then locate and select the HSM certificate that you signed previously. If you completed the steps in the previous section, select the file named <cluster ID>_CustomerHsmCertificate.crt.
   b. Next to Issuing certificate, choose Upload file. Then select your signing certificate. If you completed the steps in the previous section, select the file named customerCA.crt.
   c. Choose Upload and initialize.
To initialize a cluster (AWS CLI)

- At a command prompt, run the **initialize-cluster** command. Provide the following:
  - The ID of the cluster that you created previously.
  - The HSM certificate that you signed previously. If you completed the steps in the previous section, it's saved in a file named `<cluster ID>_CustomerHsmCertificate.crt`.
  - Your signing certificate. If you completed the steps in the previous section, the signing certificate is saved in a file named `customerCA.crt`.

```
$ aws cloudhsmv2 initialize-cluster --cluster-id <cluster ID> \
   --signed-cert file://<cluster ID>_CustomerHsmCertificate.crt \
   --trust-anchor file://customerCA.crt
```

To initialize a cluster (AWS CloudHSM API)

- Send an **InitializeCluster** request with the following:
  - The ID of the cluster that you created previously.
  - The HSM certificate that you signed previously.
  - Your signing certificate.

Install and Configure the AWS CloudHSM Client (Linux)

To interact with the HSM in your AWS CloudHSM cluster, you need the AWS CloudHSM client software for Linux. You should install it on the Linux EC2 client instance that you created previously. You can also install a client if you are using Windows. For more information, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

**Tasks**

- Install the AWS CloudHSM Client and Command Line Tools (p. 34)
- Edit the Client Configuration (p. 36)

**Install the AWS CloudHSM Client and Command Line Tools**

Connect to your client instance and run the following commands to download and install the AWS CloudHSM client and command line tools.
Install the AWS CloudHSM Client and Command Line Tools

Amazon Linux

```bash
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm

sudo yum install -y ./cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

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

sudo yum install -y ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 6

```bash
sudo yum install wget

wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm

sudo yum install -y ./cloudhsm-client-latest.el6.x86_64.rpm
```

CentOS 7

```bash
sudo yum install wget

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

sudo yum install -y ./cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 6

```bash
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm

sudo yum install -y ./cloudhsm-client-latest.el6.x86_64.rpm
```

RHEL 7

```bash
sudo yum install wget

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

sudo yum install -y ./cloudhsm-client-latest.el7.x86_64.rpm
```
Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb

sudo dpkg -i cloudhsm-client_latest_amd64.deb
```

## Edit the Client Configuration

Before you can use the AWS CloudHSM client to connect to your cluster, you must edit the client configuration.

**To edit the client configuration**

1. Copy your issuing certificate—the one that you used to sign the cluster's certificate (p. 32)—to the following location on the client instance: `/opt/cloudhsm/etc/customerCA.crt`. You need instance root user permissions on the client instance to copy your certificate to this location.

2. Use the following `configure` (p. 207) command to update the configuration files for the AWS CloudHSM client and command line tools, specifying the IP address of the HSM in your cluster. To get the HSM's IP address, view your cluster in the AWS CloudHSM console, or run the `describe-clusters` AWS CLI command. In the command's output, the HSM's IP address is the value of the `EniIp` field. If you have more than one HSM, choose the IP address for any of the HSMs; it doesn't matter which one.

   ```
   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. Go to Activate the Cluster (p. 38).

## Install and Configure the AWS CloudHSM Client (Windows)

To interact with an HSM in your AWS CloudHSM cluster, you need the AWS CloudHSM client software for Windows. You should install it on the Windows Server instance that you created previously. You can also install a client if you are using Linux. For more information, see Install and Configure the AWS CloudHSM Client (Linux) (p. 34).

**To install (or update) the latest client and command line tools**

1. Connect to your Windows Server instance.

2. Download the `AWSCloudHSMClient-latest.msi` installer.

   **Note**
   For versions 1.1.2+, the AWS CloudHSM client software for Windows runs as a Windows service. For more information, see Client History (p. 355).

3. Go to your download location and run the `AWSCloudHSMClient-latest.msi` installer. Follow the installer instructions.
Important
You must run the installer with administrative privileges.

The installer automatically registers the Cryptography API: Next Generation (CNG) and Key Storage Providers (KSPs) for AWS CloudHSM. To uninstall the AWS CloudHSM client software for Windows, run the installer again and follow the uninstall instructions.

4. Choose Close after the installer has finished.

The installer copies the following executable files into the C:\Program Files\Amazon\CloudHSM folder:

- cloudhsm_client.exe
- cloudhsm_mgmt_util.exe
- cng_config.exe
- configure.exe
- import_key.exe

Note
This file is installed from versions 1.1.2 and later.

- key_mgmt_util.exe
- ksp_config.exe
- pkpspeed_blocking.exe
- pkpspeed_blocking.exe
- set_cloudhsm_credentials.exe

Note
This file is installed from versions 2.0.4 and later.

The installer copies the following certificate and key files into the C:\ProgramData\Amazon\CloudHSM folder:

- client.crt
- client.key

The installer copies the following configuration files into the C:\ProgramData\Amazon\CloudHSM\data folder:

- application.cfg
- cloudhsm_client.cfg
- cloudhsm_mgmt_util.cfg

Note
If you are updating the client, existing configuration files from previous installations will not be overwritten.

5. Copy your self-signed issuing certificate—the one that you used to sign the cluster certificate (p. 32)—to the C:\ProgramData\Amazon\CloudHSM folder.

6. Run the following command to update your configuration files. Be sure to stop and start the client during reconfiguration if you are updating it:

   c:\Program Files\Amazon\CloudHSM>configure.exe -a <HSM IP address>

7. Go to Activate the Cluster (p. 38).
Activate the Cluster

When you activate an AWS CloudHSM cluster, the cluster's state changes from initialized to active. You can then manage the HSM's users (p. 52) and use the HSM (p. 214).

To activate the cluster, log in to the HSM with the credentials of the precrypto officer (PRECO) (p. 10). This a temporary user that exists only on the first HSM in an AWS CloudHSM cluster. The first HSM in a new cluster contains a PRECO user with a default user name and password. When you change the password, the PRECO user becomes a crypto officer (CO).

**To activate a cluster**

1. Connect to the client instance that you launched in previously. For more information, see Launch an Amazon EC2 Client Instance (p. 19). You can launch a Linux instance or a Windows Server.
2. Use the following command to start the `cloudhsm_mgmt_util` command line utility.

   **Note**
   If you are using an AMI that uses Amazon Linux 2, see Known Issues for Amazon EC2 Instances Running Amazon Linux 2 (p. 342).

   Amazon Linux
   
   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Ubuntu
   
   ```
   $ /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. Use the `enable_e2e` command to enable end-to-end encryption.

   ```
   aws-cloudhsm>enable_e2e
   E2E enabled on server 0(server1)
   ```

4. (Optional) Use the `listUsers` command to display the existing users.

   ```
   aws-cloudhsm>listUsers
   Users on server 0(server1):
   Number of users found:2
   
<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>LoginFailureCnt</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRECO</td>
<td>admin</td>
<td>0</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>0</td>
<td>NO</td>
</tr>
</tbody>
</table>
   ```

5. Use the `loginHSM` command to log in to the HSM as the PRECO user. This is a temporary user that exists on the first HSM in your cluster.

   ```
   aws-cloudhsm>loginHSM PRECO admin password
   ```
6. Use the `changePswd` command to change the password for the PRECO user. When you change the password, the PRECO user becomes a crypto officer (CO).

```plaintext
aws-cloudhsm>changePswd PRECO admin <NewPassword>

*************************CAUTION*****************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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 admin(PRECO) on 1 nodes
```

We recommend that you write down the new password on a password worksheet. Do not lose the worksheet. We recommend that you print a copy of the password worksheet, use it to record your critical HSM passwords, and then store it in a secure place. We also recommended that you store a copy of this worksheet in secure off-site storage.

7. (Optional) Use the `listUsers` command to verify that the user’s type changed to crypto officer (CO) (p. 11).

```plaintext
aws-cloudhsm>listUsers
Users on server 0(server1):
Number of users found:2

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

8. Use the `quit` command to stop the cloudhsm_mgmt_util tool.

```plaintext
aws-cloudhsm>quit
```

Reconfigure SSL with a New Certificate and Private Key (Optional)

AWS CloudHSM uses an SSL certificate to establish a connection to an HSM. A default key and SSL certificate are included when you install the client. You can, however, create and use your own. Note that you will need the self–signed certificate (`customerCA.crt`) that you created when you initialized (p. 32) your cluster.

To reconfigure SSL with a new certificate and private key

1. Create a private key using the following OpenSSL command:

```plaintext
openssl genrsa -out ssl-client.key 2048
Generating RSA private key, 2048 bit long modulus
```
2. Use the following OpenSSL command to create a certificate signing request (CSR). You will be asked a series of questions for your certificate.

```
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. Sign the CSR with the `customerCA.crt` certificate that you created when you initialized your cluster.

```
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
```

4. Copy your key and certificate to the appropriate directory. In Linux, use the following commands. The configure `--ssl` option became available with version 1.0.14 of the AWS CloudHSM client.

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

5. Add the `customerCA.crt` certificate to the trust store. Create a hash of the certificate subject name. This creates an index to allow the certificate to be looked up by that name. Create a file that contains the certificate with the hash name.

```
openssl x509 -in /opt/cloudhsm/etc/customerCA.crt -hash | head -n 1
1234abcd
sudo cp /opt/cloudhsm/etc/customerCA.crt /opt/cloudhsm/etc/certs/1234abcd.0
```
Best Practices for AWS CloudHSM

Learn best practices for working with AWS CloudHSM. As new best practices are identified, we'll update this section.

AWS CloudHSM Basic Operational Guidelines

Use the following guidelines when working with AWS CloudHSM. The AWS CloudHSM Service Level Agreement requires that you follow these guidelines.

Administration: We recommend you create at least two cryptographic officers (COs) to administer your cluster. Before setting quorum (MofN) policy, you must create at least M+1 CO accounts. Delete CO accounts with caution. If you fall below the quorum number of COs, you will no longer be able to administer your cluster.

Administration: If an HSM fails, you can experience unrecoverable data loss. We do not configure fault tolerance for you. You are responsible for configuring fault tolerance for your HSMs.

Cluster Configuration: For production clusters, you should have at least two HSM instances spread across two availability zones in a region. For latency-sensitive workloads, we recommend +1 redundancy. For applications requiring durability of newly generated keys, we recommend at least three HSM instances spread across all availability zones in a region.

Cluster Capacity: You should create enough HSMs in your cluster to handle your workload. You should consider both audit log capacity and cryptographic capacity when deciding how many HSMs to create in a cluster.
Managing AWS CloudHSM Clusters

You can manage your AWS CloudHSM clusters from the AWS CloudHSM console or one of the AWS SDKs or command line tools. For more information, see the following topics.

To create a cluster, see Getting Started (p. 15).

Topics
- Adding or Removing HSMs in an AWS CloudHSM Cluster (p. 42)
- Copying a Backup Across Regions (p. 45)
- Creating an AWS CloudHSM Cluster from a Previous Backup (p. 46)
- Deleting and Restoring an AWS CloudHSM Cluster Backup (p. 47)
- Deleting an AWS CloudHSM Cluster (p. 48)
- Tagging AWS CloudHSM Resources (p. 49)

Adding or Removing HSMs in an AWS CloudHSM Cluster

To scale up or down your AWS CloudHSM cluster, add or remove HSMs by using the AWS CloudHSM console or one of the AWS SDKs or command line tools.

Topics
- Adding an HSM (p. 42)
- Removing an HSM (p. 44)

Adding an HSM

The following figure illustrates the events that occur when you add an HSM to a cluster.
1. You add a new HSM to a cluster. The following procedures explain how to do this from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), and the AWS CloudHSM API.

   This is the only action that you take. The remaining events occur automatically.

2. AWS CloudHSM makes a backup copy of an existing HSM in the cluster. For more information, see Backups (p. 6).

3. AWS CloudHSM restores the backup onto the new HSM. This ensures that the HSM is in sync with the others in the cluster.

4. The existing HSMs in the cluster notify the AWS CloudHSM client that there's a new HSM in the cluster.

5. The client establishes a connection to the new HSM.
To add an HSM (console)
2. Choose a cluster for the HSM that you are adding.
3. On the HSMs tab, choose Create HSM.
4. Choose an Availability Zone (AZ) for the HSM that you are creating. Then choose Create.

To add an HSM (AWS CLI)
- At a command prompt, issue the create-hsm command, specifying a cluster ID and an Availability Zone for the HSM that you are creating. If you don't know the cluster ID of your preferred cluster, issue the describe-clusters command. Specify the Availability Zone in the form of us-east-2a, us-east-2b, etc.

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-zone <Availability Zone>
{
  "Hsm": {
    "State": "CREATE_IN_PROGRESS",
    "ClusterId": "cluster-5a73d5qrdh",
    "HsmId": "hsm-lgavqitns2a",
    "SubnetId": "subnet-0e358c43",
    "AvailabilityZone": "us-east-2c",
    "EniId": "eni-bab18892",
    "EniIp": "10.0.3.10"
  }
}
```

To add an HSM (AWS CloudHSM API)
- Send a CreateHsm request, specifying the cluster ID and an Availability Zone for the HSM that you are creating.

Removing an HSM
You can remove an HSM by using the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To remove an HSM (console)
2. Choose the cluster that contains the HSM that you are removing.
3. On the HSMs tab, choose the HSM that you are removing. Then choose Delete HSM.
4. Confirm that you want to delete the HSM. Then choose Delete.

To remove an HSM (AWS CLI)
- At a command prompt, issue the delete-hsm command. Pass the ID of the cluster that contains the HSM that you are deleting and one of the following HSM identifiers:
  - The HSM ID (--hsm-id)
  - The HSM IP address (--eni-ip)
  - The HSM's elastic network interface ID (--eni-id)
Copying a Backup Across Regions

AWS CloudHSM allows you to copy a backup of a cluster into a different region, where it can then be used to create a new cluster as a clone of the original. AWS CloudHSM Cluster Backups (p. 6) are packages of encrypted data that contain the elements of a particular cluster. In order to clone a cluster into a different region, first copy the cluster backup to the destination region and then create a new cluster from the copied backup. You may want to do this for a number of reasons, including simplification of the disaster recovery process.

You can copy a cluster backup across regions from the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API. Upon issuing the `copy-backup-to-region` command, the copied backup appears in the destination region with a `CREATE_IN_PROGRESS` status. Upon successful completion, the status of the copied backup is `READY`.

In the event that the `copy-backup-to-region` command cannot be successfully completed, the status of the copied backup is `DELETED`. Check your input parameters for errors and ensure that the specified source backup is not in a `DELETED` state before rerunning the operation.

For information on how to create a cluster from a backup, see Creating an AWS CloudHSM Cluster from a Previous Backup (p. 46).

**Important**
Note the following:

- To copy a cluster backup to a destination region, your account must have the proper IAM policy permissions. In order to copy the backup to a different region, your IAM policy must allow access to the source region in which the backup is located. Once copied across regions, your IAM policy must allow access to the destination region in order to interact with the copied backup, which includes using the `CreateCluster` operation. For more information, see Customer Managed Policies for AWS CloudHSM (p. 15).

- The original cluster and the cluster that may be built from a backup in the destination region are not linked. You must manage each of these clusters independently. For more information, see Managing AWS CloudHSM Clusters (p. 42)

- Backups cannot be copied between AWS restricted regions and standard regions. Backups can be copied between the AWS GovCloud (US-East) and AWS GovCloud (US-West) regions.

**To copy a cluster backup to a different region (AWS CLI)**

- At a command prompt, run the `copy-backup-to-region` command. Specify the destination region and either the cluster ID of the source cluster or the backup ID of the source backup. If you specify a backup ID, the associated backup is copied. If you specify a cluster ID, the most recent available
backup of the associated cluster is copied. If you provide both, the backup ID provided is used by default. If you don’t know the cluster ID or backup ID, run the `describe-clusters` or `describe-backups` command respectively.

```
$ aws cloudhsmv2 copy-backup-to-region --destination-region <destination region> \   --backup-id <backup ID>
{
    "DestinationBackup": {
        "CreateTimestamp": 1531742400,
        "SourceBackup": "backup-4kuraxsqetz",
        "SourceCluster": "cluster-kzlczlspnho",
        "SourceRegion": "us-east-1"
    }
}
```

**To copy a cluster backup to a different region (AWS CloudHSM API)**

- Send a `CopyBackupToRegion` request. Specify the destination region and the cluster ID or most recent backup ID of the cluster to be copied.

---

### Creating an AWS CloudHSM Cluster from a Previous Backup

To restore an AWS CloudHSM cluster from a previous backup, you create a new cluster and specify the backup to restore. After you create the cluster, you don’t need to initialize or activate it. You can just add an HSM to the cluster. This HSM contains the same users, key material, certificates, configuration, and policies that were in the backup that you restored. For more information about backups, see [Backups](#).

You can restore a cluster from a backup from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

**To create a cluster from a previous backup (console)**

2. Choose **Create cluster**.
3. In the **Cluster configuration** section, do the following:
   a. For **VPC**, choose a VPC for the cluster that you are creating.
   b. For **AZ(s)**, choose a private subnet for each Availability Zone that you are adding to the cluster.
4. In the **Cluster source** section, do the following:
   a. Choose **Restore cluster from existing backup**.
   b. Choose the backup that you are restoring.
5. Choose **Next: Review**.
6. Review your cluster configuration, then choose **Create cluster**.

**To create a cluster from a previous backup (AWS CLI)**

- At a command prompt, issue the `create-cluster` command. Specify the HSM instance type, the subnet IDs of the subnets where you plan to create HSMs, and the backup ID of the backup that you are restoring. If you don’t know the backup ID, issue the `describe-backups` command.
To create a cluster from a previous backup (AWS CloudHSM API)

- Send a CreateCluster request. Specify the HSM instance type, the subnet IDs of the subnets where you plan to create HSMs, and the backup ID of the backup that you are restoring.

To create an HSM that contains the same users, key material, certificates, configuration, and policies that were in the backup that you restored, add an HSM (p. 42) to the cluster.

Deleting and Restoring an AWS CloudHSM Cluster Backup

AWS CloudHSM Cluster Backups (p. 6) are packages of encrypted data that contain the elements of a particular cluster. Backups are generated once a day, as well as whenever an HSM is added to a cluster.

You may want to remove certain cryptographic materials from your AWS environment, such as an expired key or inactive user. In order to remove these materials, you first delete them from your HSMs. To ensure that deleted information is not restored when initializing a new cluster with an old backup, you must then delete all existing backups of the cluster. To do this, you use the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

A backup must be in the READY state in order to be deleted. You can check the status of a backup by using the describe-backups command from the AWS CLI, or with the DescribeBackups API call.

Upon successful deletion, a backup is in the PENDING_DELETION state for 7 days, during which time it can be restored with the restore-backup command. After the deletion window has passed, the target backup can no longer be restored.
To delete and restore a backup (AWS CLI)

1. At a command prompt, run the delete-backup command, passing the ID of the backup to be deleted. If you don't know the backup ID, run the describe-backups command.

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

2. To restore a backup, issue the restore-backup command, passing the ID of a backup that is in the PENDING_DELETION state. You can check the status of a backup by issuing the describe-backups command with the ID of the target backup. If you would like to see a list of all backups in the PENDING_DELETION state, run the describe-backups command and include states=PENDING_DELETION as a filter.

```bash
$ aws cloudhsmv2 describe-backups --filters states=PENDING_DELETION
{
    "Backups": [
        {
            "BackupId": "backup-ro5c4er4aac",
            "BackupState": "PENDING_DELETION",
            "CreateTimestamp": 1534461854.64,
            "ClusterId": "cluster-dygnwhmscg5",
            "DeleteTimestamp": 1536339805.522
        }
    ]
}
```

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

To delete and restore a backup (AWS CloudHSM API):

1. To delete a backup, send a DeleteBackup request, specifying the ID of the backup to be deleted.
2. To restore a backup, send a RestoreBackup request, specifying the ID of the backup to be restored.

Deleting an AWS CloudHSM Cluster

Before you can delete a cluster, you must remove all HSMs from the cluster. For more information, see Removing an HSM (p. 44).

After you remove all HSMs, you can delete a cluster by using the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.
To delete a cluster (console)

2. Choose the cluster that you are deleting. Then choose Delete cluster.
3. Confirm that you want to delete the cluster, then choose Delete.

To delete a cluster (AWS CLI)

- At a command prompt, issue the delete-cluster command, passing the ID of the cluster that you are deleting. If you don't know the cluster ID, issue the describe-clusters command.

```bash
$ aws cloudhsmv2 delete-cluster --cluster-id <cluster ID> 
{
    "Cluster": {
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "SourceBackupId": "backup-rtg2dwi2gq6",
        "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"
    }
}
```

To delete a cluster (AWS CloudHSM API)

- Send a DeleteCluster request, specifying the ID of the cluster that you are deleting.

Tagging AWS CloudHSM Resources

A tag is a label that you assign to an AWS resource. You can assign tags to your AWS CloudHSM clusters. Each tag consists of a tag key and a tag value, both of which you define. For example, the tag key might be Cost Center and the tag value might be 12345. Tag keys must be unique for each cluster.

You can use tags for a variety of purposes. One common use is to categorize and track your AWS costs. You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this report to view your AWS CloudHSM costs in terms of projects or applications, instead of viewing all AWS CloudHSM costs as a single line item.

For more information about using tags for cost allocation, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

You can use the AWS CloudHSM console or one of the AWS SDKs or command line tools to add, update, list, and remove tags.
Adding or Updating Tags

You can add or update tags from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To add or update tags (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/
2. Choose the cluster that you are tagging.
3. Choose Tags.
4. To add a tag, do the following:
   a. Choose Edit Tag and then choose Add Tag.
   b. For Key, type a key for the tag.
   c. (Optional) For Value, type a value for the tag.
   d. Choose Save.
5. To update a tag, do the following:
   a. Choose Edit Tag.
   b. Type the new tag value.
   c. Choose Save.

To add or update tags (AWS CLI)

1. At a command prompt, issue the `tag-resource` command, specifying the tags and the ID of the cluster that you are tagging. If you don’t know the cluster ID, issue the `describe-clusters` command.
   
   ```
   $ aws cloudhsmv2 tag-resource --resource-id <cluster ID> 
   --tag-list Key="<tag key>",Value="<tag value>"
   ```

2. To update tags, use the same command but specify an existing tag key. When you specify a new tag value for an existing tag, the tag is overwritten with the new value.

To add or update tags (AWS CloudHSM API)

- Send a TagResource request. Specify the tags and the ID of the cluster that you are tagging.

Listing Tags

You can list tags for a cluster from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.
To list tags (console)
2. Choose the cluster whose tags you are listing.
3. Choose Tags.

To list tags (AWS CLI)
• At a command prompt, issue the list-tags command, specifying the ID of the cluster whose tags you are listing. If you don’t know the cluster ID, issue the describe-clusters command.

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

To list tags (AWS CloudHSM API)
• Send a ListTags request, specifying the ID of the cluster whose tags you are listing.

Removing Tags

You can remove tags from a cluster by using the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To remove tags (console)
2. Choose the cluster whose tags you are removing.
3. Choose Tags.
4. Choose Edit Tag and then choose Remove tag for the tag you want to remove.
5. Choose Save.

To remove tags (AWS CLI)
• At a command prompt, issue the untag-resource command, specifying the tag keys of the tags that you are removing and the ID of the cluster whose tags you are removing. When you use the AWS CLI to remove tags, specify only the tag keys, not the tag values.

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

To remove tags (AWS CloudHSM API)
• Send an UntagResource request in the AWS CloudHSM API, specifying the ID of the cluster and the tags that you are removing.
Managing HSM Users and Keys in AWS CloudHSM

Before you can use your AWS CloudHSM cluster for cryptoprocessing, you must create users and keys on the HSMs in your cluster. See the following topics for more information about using the AWS CloudHSM command line tools to manage HSM users and keys. You can also learn how to use quorum authentication (also known as M of N access control).

Topics
- Managing HSM Users in AWS CloudHSM (p. 52)
- Managing Keys in AWS CloudHSM (p. 55)
- Enforcing Quorum Authentication (M of N Access Control) (p. 60)
- Using Trusted Keys to Control Key Unwraps (p. 72)

Managing HSM Users in AWS CloudHSM

To manage users on the HSMs in your AWS CloudHSM cluster, use the AWS CloudHSM command line tool known as cloudhsm_mgmt_util. Before you can manage users, you must start cloudhsm_mgmt_util, enable end-to-end encryption, and log in to the HSMs. For more information, see cloudhsm_mgmt_util (p. 75).

To manage HSM users, log in to the HSM with the user name and password of a cryptographic officer (p. 11) (CO). Only COs can manage other users. The HSM contains a default CO named admin. You set this user’s password when you activated the cluster (p. 38).

Topics
- Create Users (p. 52)
- List Users (p. 53)
- Change a User’s Password (p. 54)
- Delete Users (p. 54)

Create Users

Use the createuser (p. 87) command to create a user on the HSM. The following examples create new CO and CU users, respectively. For information about user types, see HSM Users (p. 10).

aws-cloudhsm>createUser CO example_officer <password>
*******************************************CAUTION*******************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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 example_officer (CO) on 3 nodes

aws-cloudhsm> createUser CU example_user <password>
****************************************************************
CAUTION****************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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 example_user (CU) on 3 nodes

The following shows the syntax for the createUser (p. 87) command. User types and passwords are case-sensitive in cloudhsm_mgmt_util commands, but user names are not.

aws-cloudhsm> createUser <user type> <user name> <password>

**List Users**

Use the listUsers (p. 108) command to list the users on each HSM in the cluster. All HSM user types (p. 10) can use this command; it's not restricted to COs.

The PCO is the first (“primary”) CO created on each HSM. It has the same permissions on the HSM as any other CO.

aws-cloudhsm> listUsers

Users on server 0(10.0.2.9):
Number of users found: 4

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginFailureCnt</td>
<td></td>
<td>2FA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>example_officer</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CU</td>
<td>example_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Users on server 1(10.0.3.11):
Number of users found: 4

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginFailureCnt</td>
<td></td>
<td>2FA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>example_officer</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CU</td>
<td>example_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Users on server 2(10.0.1.12):
Number of users found: 4

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginFailureCnt</td>
<td></td>
<td>2FA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>example_officer</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CU</td>
<td>example_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Change a User's Password

Use the `changePswd` command to change the password for the any user. All HSM user types (p. 10) can issue this command, but only COs can change the password for other users. Crypto users (CUs) and appliance users (AUs) can change only their own password. The following examples change the password for the CO and CU users that were created in the Create Users (p. 52) examples.

```
aws-cloudhsm>changePswd CO example_officer <new password>
CAUTION: This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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 example_officer(CO) on 3 nodes
```

```
aws-cloudhsm>changePswd CU example_user <new password>
CAUTION: This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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 example_user(CU) on 3 nodes
```

The following shows the syntax for the `changePswd` command. User types and passwords are case-sensitive, but user names are not.

```
aws-cloudhsm>changePswd <user type> <user name> <new password>
```

**Warning**
The CO cannot change the password for a user (CO or CU) who is currently logged in.

Delete Users

Use the `deleteUser` command to delete a user. The following examples delete the CO and CU users that were created in the Create Users (p. 52) examples.

```
aws-cloudhsm>deleteUser CO example_officer
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)
```

<table>
<thead>
<tr>
<th></th>
<th>PCO</th>
<th>NO</th>
<th>admin</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AU</td>
<td>NO</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>NO</td>
<td>example_officer</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td>CU</td>
<td>NO</td>
<td>example_user</td>
<td>NO</td>
</tr>
</tbody>
</table>
Managing Keys in AWS CloudHSM

To manage keys on the HSMs in your AWS CloudHSM cluster, use the `key_mgmt_util` (p. 122) command line tool. Before you can manage keys, you must start the AWS CloudHSM client, start `key_mgmt_util`, and log in to the HSMs.

To manage keys, log in to the HSM (p. 124) with the user name and password of a crypto user (CU). Only CUs can create keys. Keys are inherently owned and managed by the CU who created them.

**Topics**
- Generate Keys (p. 55)
- Import Keys (p. 56)
- Export Keys (p. 58)
- Delete Keys (p. 59)
- Share and Unshare Keys (p. 59)

**Generate Keys**

To generate keys on the HSM, use the command that corresponds to the type of key that you want to generate.

**Generate Symmetric Keys**

Use the `genSymKey` (p. 158) command to generate AES, triple DES, and other types of symmetric keys. To see all available options, use the `genSymKey -h` command.

The following example creates a 256-bit AES key.

```bash
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
```
Generate RSA Key Pairs

To generate an RSA key pair, use the `genRSAKeyPair` command. To see all available options, use the `genRSAKeyPair -h` command.

The following example generates an RSA 2048-bit key pair.

```
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
```

Generate ECC (Elliptic Curve Cryptography) Key Pairs

To generate an ECC key pair, use the `genECCKeyPair` command. To see all available options, including a list of the supported elliptic curves, use the `genECCKeyPair -h` command.

The following example generates an ECC key pair using the P-384 elliptic curve defined in NIST FIPS publication 186-4.

```
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
```

Import Keys

To import secret keys—that is, symmetric keys and asymmetric private keys—into the HSM, you must first create a wrapping key on the HSM. You can import public keys directly without a wrapping key.

Import Secret Keys

Complete the following steps to import a secret key. Before you import a secret key, save it to a file. Save symmetric keys as raw bytes, and asymmetric private keys in PEM format.

This example shows how to import a plaintext secret key from a file into the HSM. To import an encrypted key from a file into the HSM, use the `unWrapKey` command.

**To import a secret key**

1. Use the `genSymKey` command to create a wrapping key. The following command creates a 128-bit AES wrapping key that is valid only for the current session. You can use a session key or a persistent key as a wrapping key.

```
Command: genSymKey -t 31 -s 16 -sess -l import-wrapping-key

Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524299
```
2. Use one of the following commands, depending on the type of secret key that you are importing.

- To import a symmetric key, use the `imSymKey` command. The following command imports an AES key from a file named `aes256.key` using the wrapping key created in the previous step. To see all available options, use the `imSymKey -h` command.

  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
  ```

- To import an asymmetric private key, use the `importPrivateKey` command. The following command imports a private key from a file named `rsa2048.key` using the wrapping key created in the previous step. To see all available options, use the `importPrivateKey -h` command.

  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
  ```

Import Public Keys

Use the `importPubKey` command to import a public key. To see all available options, use the `importPubKey -h` command.

The following example imports an RSA public key from a file named `rsa2048.pub`.

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
```
Export Keys

To export secret keys—that is, symmetric keys and asymmetric private keys—from the HSM, you must first create a wrapping key. You can export public keys directly without a wrapping key.

Only the key owner can export a key. Users with whom the key is shared can use the key in cryptographic operations, but they cannot export it. When running this example, be sure to export a key that you created.

Important
The `exSymKey (p. 136)` command writes a plaintext (unencrypted) copy of the secret key to a file. The export process requires a wrapping key, but the key in the file is not a wrapped key. To export a wrapped (encrypted) copy of a key, use the `wrapKey (p. 201)` command.

Export Secret Keys

Complete the following steps to export a secret key.

To export a secret key

1. Use the `genSymKey (p. 158)` command to create a wrapping key. The following command creates a 128-bit AES wrapping key that is valid only for the current session.

   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. Use one of the following commands, depending on the type of secret key that you are exporting.

   - To export a symmetric key, use the `exSymKey (p. 136)` command. The following command exports an AES key to a file named `aes256.key.exp`. To see all available options, use the `exSymKey -h` command.

     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"

   - To export a private key, use the `exportPrivateKey` command. The following command exports a private key to a file named `rsa2048.key.exp`. To see all available options, use the `exportPrivateKey -h` command.

     Command: `exportPrivateKey -k 524296 -out rsa2048.key.exp -w 524304`

     Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

     Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS

Note
The command's output says that a "Wrapped Symmetric Key" is written to the output file. However, the output file contains a plaintext (not wrapped) key. To export a wrapped (encrypted) key to a file, use the `wrapKey (p. 201)` command.
Export Public Keys

Use the `exportPubKey` command to export a public key. To see all available options, use the `exportPubKey -h` command.

The following example exports an RSA public key to a file named `rsa2048.pub.exp`.

Command:
```
exportPubKey -k 524294 -out rsa2048.pub.exp
```

PEM formatted public key is written to `rsa2048.pub.key`

Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS

Delete Keys

Use the `deleteKey` (p. 130) command to delete a key, as in the following example. Only the key owner can delete a key.

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

Share and Unshare Keys

In AWS CloudHSM, the CU who creates the key owns it. The owner manages the key, can export and delete it, and can use the key in cryptographic operations. The owner can also share the key with other CU users. Users with whom the key is shared can use the key in cryptographic operations, but they cannot export or delete the key, or share it with other users.

You can share keys with other CU users when you create the key, such as by using the `-u` parameter of the `genSymKey` (p. 158) or `genRSAKeyPair` (p. 154) commands. To share existing keys with a different HSM user, use the `cloudhsm_mgmt_util` (p. 75) command line tool. This is different from most of the tasks documented in this section, which use the `key_mgmt_util` (p. 122) command line tool.

Before you can share a key, you must start `cloudhsm_mgmt_util`, enable end-to-end encryption, and log in to the HSMs. To share a key, log in to the HSM as the crypto user (CU) that owns the key. Only key owners can share a key.

Use the `shareKey` command to share or unshare a key, specifying the handle of the key and the IDs of the user or users. To share or unshare with more than one user, specify a comma-separated list of user IDs. To share a key, use 1 as the command's last parameter, as in the following example. To unshare, use 0.

```
aws-cloudhsm> shareKey 524295 4 1
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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)

The following shows the syntax for the `shareKey` command.

```
aws-cloudhsm>shareKey <key handle> <user ID> <Boolean: 1 for share, 0 for unshare>
```

**Enforcing Quorum Authentication (M of N Access Control)**

The HSMs in your AWS CloudHSM cluster support quorum authentication, which is also known as M of N access control. With quorum authentication, no single user on the HSM can do quorum-controlled operations on the HSM. Instead, a minimum number of HSM users (at least 2) must cooperate to do these operations. With quorum authentication, you can add an extra layer of protection by requiring approvals from more than one HSM user.

Quorum authentication can control the following operations:

- HSM user management by crypto officers (COs) (p. 11) – Creating and deleting HSM users, and changing a different HSM user's password. For more information, see Using Quorum Authentication for Crypto Officers (p. 65).

The following topics provide more information about quorum authentication in AWS CloudHSM.

**Topics**

- Overview of Quorum Authentication (p. 60)
- Additional Details about Quorum Authentication (p. 61)
- Using Quorum Authentication for Crypto Officers: First Time Setup (p. 61)
- Using Quorum Authentication for Crypto Officers (p. 65)
- Change the Quorum Minimum Value for Crypto Officers (p. 71)

**Overview of Quorum Authentication**

The following steps summarize the quorum authentication processes. For the specific steps and tools, see Using Quorum Authentication for Crypto Officers (p. 65).

1. Each HSM user creates an asymmetric key for signing. He or she does this outside of the HSM, taking care to protect the key appropriately.
2. Each HSM user logs in to the HSM and registers the public part of his or her signing key (the public key) with the HSM.
3. When an HSM user wants to do a quorum-controlled operation, he or she logs in to the HSM and gets a quorum token.
4. The HSM user gives the quorum token to one or more other HSM users and asks for their approval.
5. The other HSM users approve by using their keys to cryptographically sign the quorum token. This occurs outside the HSM.
6. When the HSM user has the required number of approvals, he or she logs in to the HSM and gives the quorum token and approvals (signatures) to the HSM.
7. The HSM uses the registered public keys of each signer to verify the signatures. If the signatures are valid, the HSM approves the token.
8. The HSM user can now do a quorum-controlled operation.

Additional Details about Quorum Authentication

Note the following additional information about using quorum authentication in AWS CloudHSM.

- An HSM user can sign his or her own quorum token—that is, the requesting user can provide one of the required approvals for quorum authentication.
- You choose the minimum number of quorum approvers for quorum-controlled operations. The smallest number you can choose is two (2). For HSM user management operations by COs, the largest number you can choose is twenty (20).
- The HSM can store up to 1024 quorum tokens. If the HSM already has 1024 tokens when you try to create a new one, the HSM purges one of the expired tokens. By default, tokens expire ten minutes after their creation.

Using Quorum Authentication for Crypto Officers: First Time Setup

The following topics describe the steps that you must complete to configure your HSM so that crypto officers (COs) can use quorum authentication. You need to do these steps only once when you first configure quorum authentication for COs. After you complete these steps, see Using Quorum Authentication for Crypto Officers (p. 65).

Topics
- Prerequisites (p. 61)
- Create and Register a Key for Signing (p. 62)
- Set the Quorum Minimum Value on the HSM (p. 64)

Prerequisites

To understand this example, you should be familiar with the cloudhsm_mgmt_util command line tool (p. 75). In this example, the AWS CloudHSM cluster has two HSMs, each with the same COs, as shown in the following output from the listUsers command. For more information about creating users, see Managing HSM Users (p. 52).

```
aws-cloudhsm>listUsers
Users on server 0(10.0.2.14):
Number of users found:7

<table>
<thead>
<tr>
<th>User Id</th>
<th>LoginFailureCnt</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>CO</td>
<td>officer1</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>CO</td>
<td>officer2</td>
<td>NO</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>CO</td>
<td>officer3</td>
<td>NO</td>
</tr>
</tbody>
</table>
```

Create and Register a Key for Signing

To use quorum authentication, each CO must create an asymmetric key for signing (a signing key). This is done outside of the HSM.

There are many different ways to create and protect a personal signing key. The following example shows how to do it with OpenSSL.

**Example – Create a personal signing key with OpenSSL**

The following example demonstrates how to use OpenSSL to create a 2048-bit RSA key that is protected by a pass phrase. To use this example, replace `officer1.key` with the name of the file where you want to store the key.

```
$ 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:
```

Each CO should create his or her own key.

After creating a key, the CO must register the public part of the key (the public key) with the HSM.

**To register a public key with the HSM**

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. Use the `enable_e2e` command to establish end-to-end encrypted communication.

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 81).
4. Use the `registerMofnPubKey` command to register the public key. For more information, see the following example or use the `help registerMofnPubKey` command.

**Example – Register a public key with the HSM**

The following example shows how to use the `registerMofnPubKey` command in the `cloudhsm_mgmt_util` command line tool to register a CO's public key with the HSM. To use this command, the CO must be logged in to the HSM. Replace these values with your own:

- `key_match_string` – An arbitrary string that is used to match the public and private keys. You can use any string for this value. The `cloudhsm_mgmt_util` command line tool encrypts this string with the private key, and then sends the encrypted blob and the plaintext string to the HSM. The HSM uses the public key to decrypt the encrypted blob, and then compares the decrypted string to the plaintext string. If the strings match, the HSM registers the public key; otherwise it doesn't.
- `officer1` – The user name of the CO who is registering the public key. This must be the same CO who is logged in to the HSM and is running this command.
- `officer1.key` – The name of the file that contains the CO's key. This file must contain the complete key (not just the public part) because the `cloudhsm_mgmt_util` command line tool uses the private key to encrypt the `key match string`.

When prompted, type the pass phrase that protects the CO's key.

```
aws-cloudhsm> registerMofnPubKey CO key_match_string officer1 officer1.key
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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
Enter PEM pass phrase: 
registerMofnPubKey success on server 0(10.0.2.14)
registerMofnPubKey success on server 1(10.0.1.4)
```

Each CO must register his or her public key with the HSM. After all COs register their public keys, the output from the `listUsers` command shows this in the `MofnPubKey` column, as shown in the following example.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.2.14):
Number of users found: 7

<table>
<thead>
<tr>
<th>User Id</th>
<th>LoginFailureCnt</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>CO</td>
<td>officer1</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>CO</td>
<td>officer2</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>CO</td>
<td>officer3</td>
<td>YES</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>CO</td>
<td>officer4</td>
<td>YES</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>CO</td>
<td>officer5</td>
<td>YES</td>
</tr>
</tbody>
</table>
```


Users on server 1(10.0.1.4):
Number of users found: 7

<table>
<thead>
<tr>
<th>User Id</th>
<th>LoginFailureCnt</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>CO</td>
<td>officer1</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>CO</td>
<td>officer2</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>CO</td>
<td>officer3</td>
<td>YES</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>CO</td>
<td>officer4</td>
<td>YES</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>CO</td>
<td>officer5</td>
<td>YES</td>
</tr>
</tbody>
</table>

Set the Quorum Minimum Value on the HSM

To use quorum authentication for COs, a CO must log in to the HSM and then set the *quorum minimum value*, also known as the *m value*. This is the minimum number of CO approvals that are required to perform HSM user management operations. Any CO on the HSM can set the quorum minimum value, including COs that have not registered a key for signing. You can change the quorum minimum value at any time; for more information, see Change the Quorum Value for Crypto Officers (p. 71).

**To set the quorum minimum value on the HSM**

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

   ```bash
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

2. Use the `enable_e2e` command to establish end-to-end encrypted communication.

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 81).

4. Use the `setMValue` command to set the quorum minimum value. For more information, see the following example or use the `help setMValue` command.

**Example – Set the quorum minimum value on the HSM**

This example uses a quorum minimum value of two. You can choose any value from two to twenty, up to the total number of COs on the HSM. In this example, the HSM has six COs (the PCO user (p. 11) is the same as a CO), so the maximum possible value is six.

To use the following example command, replace the final number (2) with the preferred quorum minimum value.

```
aws-cloudhsm> setMValue 3 2
****************************CAUTION****************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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
```
In the preceding example, the first number (3) identifies the HSM service whose quorum minimum value you are setting.

The following table lists the HSM service identifiers along with their names, descriptions, and the commands that are included in the service.

<table>
<thead>
<tr>
<th>Service Identifier</th>
<th>Service Name</th>
<th>Service Description</th>
<th>HSM Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>USER_MGMT</td>
<td>HSM user management</td>
<td>• createUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• deleteUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• changePswd (applies only when changing the password of a different HSM user)</td>
</tr>
<tr>
<td>4</td>
<td>MISC_CO</td>
<td>Miscellaneous CO service</td>
<td>• setMValue</td>
</tr>
</tbody>
</table>

To get the quorum minimum value for a service, use the `getMValue` command, as in the following example.

```
aws-cloudhsm>getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

The output from the preceding `getMValue` command shows that the quorum minimum value for HSM user management operations (service 3) is now two.

After you complete these steps, see Using Quorum Authentication for Crypto Officers (p. 65).

Using Quorum Authentication for Crypto Officers

A crypto officer (CO) (p. 11) on the HSM can configure quorum authentication for the following operations on the HSM:

- Creating HSM users
- Deleting HSM users
- Changing another HSM user's password

After the HSM is configured for quorum authentication, COs cannot perform HSM user management operations on their own. The following example shows the output when a CO attempts to create a new user on the HSM. The command fails with a `RET_MXN_AUTH_FAILED` error, which indicates that quorum authentication failed.

```
aws-cloudhsm>createUser CU user1 password
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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

To perform an HSM user management operation, a CO must complete the following tasks:

1. Get a quorum token (p. 66).
2. Get approvals (signatures) from other COs (p. 67).
3. Approve the token on the HSM (p. 67).
4. Perform the HSM user management operation (p. 69).

If you have not yet configured the HSM for quorum authentication for COs, do that now. For more information, see First Time Setup for Crypto Officers (p. 61).

Get a Quorum Token

First the CO must use the cloudhsm_mgmt_util command line tool to request a quorum token.

To get a quorum token

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. Use the `enable_e2e` command to establish end-to-end encrypted communication.

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 81).

4. Use the `getToken` command to get a quorum token. For more information, see the following example or use the `help getToken` command.

Example – Get a quorum token

This example gets a quorum token for the CO with user name officer1 and saves the token to a file named officer1.token. To use the example command, replace these values with your own:

- `officer1` – The name of the CO who is getting the token. This must be the same CO who is logged in to the HSM and is running this command.
- `officer1.token` – The name of the file to use for storing the quorum token.

In the following command, 3 identifies the service for which you can use the token that you are getting. In this case, the token is for HSM user management operations (service 3). For more information, see Set the Quorum Minimum Value on the HSM (p. 64).

```
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:
```
Get Signatures from Approving COs

A CO who has a quorum token must get the token approved by other COs. To give their approval, the other COs use their signing key to cryptographically sign the token. They do this outside the HSM.

There are many different ways to sign the token. The following example shows how to do it with OpenSSL. To use a different signing tool, make sure that the tool uses the CO’s private key (signing key) to sign a SHA-256 digest of the token.

Example – Get signatures from approving COs

In this example, the CO that has the token (officer1) needs at least two approvals. The following example commands show how two COs can use OpenSSL to cryptographically sign the token.

In the first command, officer1 signs his or her own token. To use the following example commands, replace these values with your own:

- officer1.key and officer2.key – The name of the file that contains the CO's signing key.
- officer1.token.sig1 and officer1.token.sig2 – The name of the file to use for storing the signature. Make sure to save each signature in a different file.
- officer1.token – The name of the file that contains the token that the CO is signing.

```
$ openssl dgst -sha256 -sign officer1.key -out officer1.token.sig1 officer1.token
Enter pass phrase for officer1.key:
```

In the following command, officer2 signs the same token.

```
$ openssl dgst -sha256 -sign officer2.key -out officer1.token.sig2 officer1.token
Enter pass phrase for officer2.key:
```

Approve the Signed Token on the HSM

After a CO gets the minimum number of approvals (signatures) from other COs, he or she must approve the signed token on the HSM.

To approve the signed token on the HSM

1. Create a token approval file. For more information, see the following example.
2. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

3. Use the enable_e2e command to establish end-to-end encrypted communication.
4. Use the loginHSM command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 81).
5. Use the approveToken command to approve the signed token, passing the token approval file. For more information, see the following example.
Example – Create a token approval file and approve the signed token on the HSM

The token approval file is a text file in a particular format that the HSM requires. The file contains information about the token, its approvers, and the approvers' signatures. The following shows an example token approval file.

```
# 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;
```

After creating the token approval file, the CO uses the cloudhsm_mgmt_util command line tool to log in to the HSM. The CO then uses the `approveToken` command to approve the token, as shown in the following example. Replace `approval.txt` with the name of the token approval file.

```
aws-cloudhsm>approveToken approval.txt
approveToken success on server 0(10.0.2.14)
approveToken success on server 1(10.0.1.4)
```

When this command succeeds, the HSM has approved the quorum token. To check the status of a token, use the `listTokens` command, as shown in the following example. The command's output shows that the token has the required number of approvals.

```
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
```

The token validity time indicates how long the token is guaranteed to persist on the HSM. Even after the token validity time elapses (zero seconds), you can still use the token.
Use the Token for User Management Operations

After a CO has a token with the required number of approvals, as shown in the previous section, the CO can perform one of the following HSM user management operations:

- Create an HSM user with the `createUser` command
- Delete an HSM user with the `deleteUser` command
- Change a different HSM user's password with the `changePswd` command

For more information about using these commands, see Managing HSM Users.

The CO can use the token for only one operation. When that operation succeeds, the token is no longer valid. To do another HSM user management operation, the CO must get a new quorum token, get new signatures from approvers, and approve the new token on the HSM.

In the following example command, the CO creates a new user on the HSM.

```bash
createUser CU user1 password
```

This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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

After the previous command succeeds, a subsequent `listUsers` command shows the new user.

```bash
listUsers
```

Users on server 0(10.0.2.14):

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>LoginFailureCnt</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>
If the CO tries to perform another HSM user management operation, it fails with a quorum authentication error, as shown in the following example.

```
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)
```

The **listTokens** command shows that the CO has no approved tokens, as shown in the following example. To perform another HSM user management operation, the CO must get a new quorum token, get new signatures from approvers, and approve the new token on the HSM.

```
aws-cloudhsm> listTokens

=======================
Server 0(10.0.2.14)

Num of tokens = 0

=======================
Server 1(10.0.1.4)
```
Change the Quorum Minimum Value for Crypto Officers

After you set the quorum minimum value (p. 64) so that crypto officers (COs) (p. 11) can use quorum authentication, you might want to change the quorum minimum value. The HSM allows you to change the quorum minimum value only when the number of approvers is the same or higher than the current quorum minimum value. For example, if the quorum minimum value is two, at least two COs must approve to change the quorum minimum value.

To get quorum approval to change the quorum minimum value, you need a quorum token for the `setMValue` command (service 4). To get a quorum token for the `setMValue` command (service 4), the quorum minimum value for service 4 must be higher than one. This means that before you can change the quorum minimum value for COs (service 3), you might need to change the quorum minimum value for service 4.

The following table lists the HSM service identifiers along with their names, descriptions, and the commands that are included in the service.

<table>
<thead>
<tr>
<th>Service Identifier</th>
<th>Service Name</th>
<th>Service Description</th>
<th>HSM Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>USER_MGMT</td>
<td>HSM user management</td>
<td>• createUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• deleteUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• changePswd (applies only when changing the password of a different HSM user)</td>
</tr>
<tr>
<td>4</td>
<td>MISC_CO</td>
<td>Miscellaneous CO service</td>
<td>• setMValue</td>
</tr>
</tbody>
</table>

To change the quorum minimum value for crypto officers

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

   ```sh
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

2. Use the `enable_e2e` command to establish end-to-end encrypted communication.
3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 81).
4. Use the `getMValue` command to get the quorum minimum value for service 3. For more information, see the following example.
5. Use the `getMValue` command to get the quorum minimum value for service 4. For more information, see the following example.
6. If the quorum minimum value for service 4 is lower than the value for service 3, use the `setMValue` command to change the value for service 4. Change the value for service 4 to one that is the same or higher than the value for service 3. For more information, see the following example.
7. Get a quorum token (p. 66), taking care to specify service 4 as the service for which you can use the token.
8. Get approvals (signatures) from other COs (p. 67).
9. Approve the token on the HSM (p. 67).
10. Use the `setMValue` command to change quorum minimum value for service 3 (user management operations performed by COs).

**Example – Get quorum minimum values and change the value for service 4**

The following example command shows that the quorum minimum value for service 3 is currently two.

```
aws-cloudhsm>getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

The following example command shows that the quorum minimum value for service 4 is currently one.

```
aws-cloudhsm>getMValue 4
MValue of service 4[MISC_CO] on server 0 : [1]
MValue of service 4[MISC_CO] on server 1 : [1]
```

To change the quorum minimum value for service 4, use the `setMValue` command, setting a value that is the same or higher than the value for service 3. The following example sets the quorum minimum value for service 4 to two (2), the same value that is set for service 3.

```
aws-cloudhsm>setMValue 4 2
*****************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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
```

The following commands show that the quorum minimum value is now two for service 3 and service 4.

```
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]
```

**Using Trusted Keys to Control Key Unwraps**

It is possible that exportable keys (a data key for example) on the HSM could be wrapped with an arbitrary key (for example a bad wrapping key) and that could result in loss of the data key. Though this action would have to be initiated by a hostile insider, if this is a concern, there are solutions that address this concern.

The first solution is to block all key exports from the cluster. This solution has limitations because restricting key exports negatively impact applications that need to export and import data keys. However, a more flexible solution is to use a key unwrap template along with the trusted key and wrap-
with-trusted attributes. AWS CloudHSM 3.0 and higher supports trusted keys and unwrap templates. This article explains how to use a key unwrap template along with trusted key and wrap-with-trusted attributes.

**Background**

A key attribute is a property associated with the key, within the HSM, that specifies the permissions associated with the key. If you want to:

- **CKA_WRAP_WITH_TRUSTED**: When applied to an exportable data key, this attribute ensures the data key can only be wrapped with a key that has been marked as “CKA_TRUSTED” by a cryptographic officer. Once set to true, this becomes a read-only attribute and cannot be unset by the cryptographic user.

- **CKA_TRUSTED**: This is the only attribute that is specified by a cryptographic officer, rather than the cryptographic user who is the owner of a key. CKA_TRUSTED indicates that the cryptographic officer has done the necessary diligence, and recognizes this key is trusted.

- **CKA_UNWRAP_TEMPLATE**: An attribute template is a collection of attribute names and values. When an unwrap template is specified for a wrapping key, all attributes in that template are automatically applied to the unwrapped data key. When an application submits a key for unwrapping, it can separately provide an unwrap template. In this case, the HSM uses the union of both templates. However, if a value in the CKA_UNWRAP_TEMPLATE for the wrapping key conflicts with an attribute provided by the application during the unwrap request, the unwrap request fails.

For more information about the PKCS#11 attributes supported by AWS CloudHSM, see the article on **Supported PKCS #11 Attributes**.

**Process**

**Step 1: Generate the key bits for the trusted key**

To set up a trusted key, a security officer human establishes a cryptographic user (CU) account and generates the wrapping keys with the appropriate CKA_UNWRAP_TEMPLATE specification. Generally, you should include CKA_WRAP_WITH_TRUSTED = TRUE as part of this template. If you want the unwrapped keys to be non-exportable, set CKA_EXPORTABLE = FALSE. To generate the key, you must use a PKCS#11 application. The advanced attributes are not supported by command line tools.

**Step 2: Mark the key as trusted**

The security officer human logs in to cloudhsm_mgmt_util with a cryptographic officer (CO) account. To mark the key as trusted, call `setAttribute` with OBJ_ATTR_TRUSTED (value 134) set to TRUE. To learn more about the `setAttribute` function, see the article on `setAttribute`

```bash
loginHSM CO <user-name> <password>
setAttribute HH 134 1
quit
```

**Step 3: Share the trusted key with the application**

The security officer human logs in with the CU account and uses the `shareKey` function to share the trusted wrapping keys with the CU accounts that will be used by the applications. Then, the application CU account can use the trusted keys for wrapping and unwrapping data keys. However, users cannot modify attributes for keys shared with them, and cryptographic user accounts cannot be used to modify the key’s CKA_TRUSTED attribute. Once this is completed, the security officer can be assured the trusted wrapping keys will remain correct.
Step 4: Generate and wrap out all the data keys

Using their CU account, the security officer imports or generates all data keys, and wraps them with the trusted wrapping key. The wrapped keys are stored externally, as necessary. Since data keys can only be unwrapped with the original wrapping keys, the appropriate template must be applied at unwrap. An application can unwrap keys on demand as needed, and delete the unwrapped key from the HSM once the key is no longer required.

Sample: Generate a Key with an Unwrap Template in PKCS #11

This example uses an unWrap template to generate a key.

```cpp
std::vector CK_ATTRIBUTE unwrapTemplate = {
    CK_ATTRIBUTE { CKA_KEY_TYPE, &aes, sizeof(aes) },
    CK_ATTRIBUTE { CKA_CLASS, &aesClass, sizeof(aesClass) },
    CK_ATTRIBUTE { CKA_TOKEN, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_EXTRACTABLE, &falseValue, sizeof(falseValue) }
};
std::vector CK_ATTRIBUTE pubAttributes = {
    CK_ATTRIBUTE { CKA_KEY_TYPE, &rsa, sizeof(rsa) },
    CK_ATTRIBUTE { CKA_CLASS, &pubClass, sizeof(pubClass) },
    CK_ATTRIBUTE { CKA_TOKEN, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_MODULUS_BITS, &modulusBits, sizeof(modulusBits) },
    CK_ATTRIBUTE { CKA_PUBLIC_EXPONENT, publicExponent.data(),
                  publicExponent.size() },
    CK_ATTRIBUTE { CKA_VERIFY, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_WRAP, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_ENCRYPT, &trueValue, sizeof(trueValue) }
};
std::vector CK_ATTRIBUTE priAttributes = {
    CK_ATTRIBUTE { CKA_KEY_TYPE, &rsa, sizeof(rsa) },
    CK_ATTRIBUTE { CKA_CLASS, &priClass, sizeof(priClass) },
    CK_ATTRIBUTE { CKA_PRIVATE, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_TOKEN, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_SIGN, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_DECRYPT, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_UNWRAP, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_UNWRAP_TEMPLATE, unwrapTemplate.data(),
                  unwrapTemplate.size() * sizeof(CK_ATTRIBUTE) }
};
BOOST_TEST_CONTEXT("Generate RSA Key Pair with Unwrap Template") {
    BOOST_TEST_REQUIRE(CKR_OK == PKCS11_INVOKE_NOEXCEPT(
        get_module(),
        C_GenerateKeyPair, 
        get_session().get_handle(), 
        &rsaMechanism,
        pubAttributes.data(),
        pubAttributes.size(),
        priAttributes.data(),
        priAttributes.size(),
        &pubKey,
        &priKey
    ));
```
AWS CloudHSM Command Line Tools

AWS CloudHSM provides command line tools for managing and using AWS CloudHSM.

Topics
- cloudhsm_mgmt_util (p. 75)
- key_mgmt_util (p. 122)
- Configure Tool (p. 207)

Manage Clusters and HSMs
These tools get, create, delete, and tag AWS CloudHSM clusters and HSMs:
- CloudHSMv2 commands in AWS Command Line Interface (AWS CLI). To use these commands, you need to install and configure AWS CLI.
- HSM2 PowerShell cmdlets in the AWSPowerShell module. These cmdlets are available in a Windows PowerShell module and a cross-platform PowerShell Core module.

Manage Users
This tool creates and deletes HSM users, including implementing quorum authentication of user management tasks:
- cloudhsm_mgmt_util (p. 75). This tool is included in the AWS CloudHSM client software.

Manage Keys
This tool creates, deletes, imports, and exports symmetric keys and asymmetric key pairs:
- key_mgmt_util (p. 122). This tool is included in the AWS CloudHSM client software.

Helper Tools
These tools help you to use the tools and software libraries.
- configure (p. 207) updates your CloudHSM client configuration files. This enables the AWS CloudHSM to synchronize the HSMs in a cluster.
- pkpspeed (p. 348) measures the performance of your HSM hardware independent of software libraries.

cloudhsm_mgmt_util

The cloudhsm_mgmt_util command line tool helps crypto officers manage users in the HSMs. It includes tools that create, delete, and list users, and change user passwords.
cloudhsm_mgmt_util also includes commands that allow crypto users (CUs) to share keys and get and set key attributes. These commands complement the key management commands in the primary key management tool, key_mgmt_util (p. 122).

For a quick start, see Getting Started with cloudhsm_mgmt_util (p. 76). For detailed information about the cloudhsm_mgmt_util commands and examples of using the commands, see cloudhsm_mgmt_util Command Reference (p. 82).

Topics
• Getting Started with cloudhsm_mgmt_util (p. 76)
• cloudhsm_mgmt_util Command Reference (p. 82)

Getting Started with cloudhsm_mgmt_util

AWS CloudHSM includes two command line tools with the AWS CloudHSM client software (p. 34). The cloudhsm_mgmt_util (p. 82) tool includes commands to manage HSM users. The key_mgmt_util (p. 125) tool includes commands to manage keys. To get started with the cloudhsm_mgmt_util command line tool, complete the following tasks.

Tasks
• Prepare to run cloudhsm_mgmt_util (p. 76)
• Basic Usage of cloudhsm_mgmt_util (p. 80)
• Using cloudhsm_mgmt_util Across Cloned Clusters (p. 82)

Prepare to run cloudhsm_mgmt_util

Complete the following tasks before you use cloudhsm_mgmt_util. You need to do these steps the first time you use cloudhsm_mgmt_util and after you add or remove HSMs in your cluster. The steps update the HSM list in the configuration files that the AWS CloudHSM client and command line tools use. Keeping these files updated helps AWS CloudHSM to synchronize data and maintain consistency across all HSMs in the cluster.

Tasks
• Step 1: Stop the AWS CloudHSM Client (p. 76)
• Step 2: Update the AWS CloudHSM Configuration Files (p. 77)
• Step 3: Start the AWS CloudHSM Client (p. 78)
• Step 4: Update the cloudhsm_mgmt_util Configuration File (p. 79)

Step 1: Stop the AWS CloudHSM Client

Before you update the configuration files that the AWS CloudHSM and command line tools use, stop the AWS CloudHSM client. If the client is already stopped, running the stop command has no harmful effect.

Amazon Linux

```bash
# sudo stop cloudhsm-client
```

Amazon Linux 2

```bash
# sudo service cloudhsm-client stop
```
CentOS 6

```bash
$ sudo stop cloudhsm-client
```

CentOS 7

```bash
$ sudo service cloudhsm-client stop
```

RHEL 6

```bash
$ sudo stop cloudhsm-client
```

RHEL 7

```bash
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```bash
$ sudo service cloudhsm-client stop
```

Windows

- For Windows client 1.1.2+

  ```bash
  C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older

  Use Ctrl+C in the command window where you started the AWS CloudHSM client.

**Step 2: Update the AWS CloudHSM Configuration Files**

This step uses the `-a` parameter of the configure tool (p. 207) to add the elastic network interface (ENI) IP address of one of the HSMs in the cluster to the configuration file.

**Amazon Linux**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```

**Amazon Linux 2**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```

**CentOS 6**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```

**CentOS 7**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```

**RHEL 6**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```

**RHEL 7**

```bash
$ sudo /opt/cloudhsm/bin/configure -a <HSM ENI IP>
```
To get the ENI IP address of an HSM in your cluster, navigate to the AWS CloudHSM console, choose clusters, and select the desired cluster. You can also use the DescribeClusters operation, the describe-clusters command, or the Get-HSM2Cluster PowerShell cmdlet. Type only one ENI IP address. It does not matter which ENI IP address you use.

**Step 3: Start the AWS CloudHSM Client**

Next, start or restart the AWS CloudHSM client. When the AWS CloudHSM client starts, it uses the ENI IP address in its configuration file to query the cluster. Then it adds the ENI IP addresses of all HSMs in the cluster to the cluster information file.

Amazon Linux

```
# sudo start cloudhsm-client
```

Amazon Linux 2

```
# sudo service cloudhsm-client start
```

CentOS 6

```
# sudo start cloudhsm-client
```

CentOS 7

```
# sudo service cloudhsm-client start
```

RHEL 6

```
# sudo start cloudhsm-client
```

RHEL 7

```
# sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
# sudo service cloudhsm-client start
```
Windows

- For Windows client 1.1.2+

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

### Step 4: Update the cloudhsm_mgmt_util Configuration File

The final step uses the `-m` parameter of the configure tool (p. 207) to copy the updated ENI IP addresses from the cluster information file to the configuration file that cloudhsm_mgmt_util uses. If you skip this step, you might run into synchronization problems, such as inconsistent user data (p. 348) in your cluster’s HSMs.

Amazon Linux

```
$ sudo /opt/cloudhsm/bin/configure -m
```

Amazon Linux 2

```
# sudo /opt/cloudhsm/bin/configure -m
```

CentOS 6

```
# sudo /opt/cloudhsm/bin/configure -m
```

CentOS 7

```
# sudo /opt/cloudhsm/bin/configure -m
```

RHEL 6

```
# sudo /opt/cloudhsm/bin/configure -m
```

RHEL 7

```
# sudo /opt/cloudhsm/bin/configure -m
```

Ubuntu 16.04 LTS

```
# sudo /opt/cloudhsm/bin/configure -m
```

Windows

```
c:\Program Files\Amazon\CloudHSM>configure.exe -m
```

When this step is complete, you are ready to start cloudhsm_mgmt_util. If you add or delete HSMs at any time, be sure to repeat this procedure before using cloudhsm_mgmt_util.
Basic Usage of \texttt{cloudhsm\_mgmt\_util}

The following tasks cover the basic usage of the \texttt{cloudhsm\_mgmt\_util} tool.

\textbf{Warning}

The \texttt{cloudhsm\_mgmt\_util} tool doesn't support auto-completion using the Tab key. Using the Tab key with \texttt{cloudhsm\_mgmt\_util} can make it unresponsive.

\textbf{Tasks}

\begin{itemize}
  \item Start \texttt{cloudhsm\_mgmt\_util} (p. 80)
  \item Enable End-to-End Encryption (p. 81)
  \item Log in to the HSMs (p. 81)
  \item Log Out from the HSMs (p. 81)
  \item Stop \texttt{cloudhsm\_mgmt\_util} (p. 82)
\end{itemize}

\textbf{Start \texttt{cloudhsm\_mgmt\_util}}

Use the following command to start \texttt{cloudhsm\_mgmt\_util}.

\textbf{Amazon Linux}

\begin{verbatim}
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
\end{verbatim}

\begin{itemize}
  \item Amazon Linux 2
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item CentOS 6
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item CentOS 7
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item RHEL 6
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item RHEL 7
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item Ubuntu 16.04 LTS
  \begin{verbatim}
  $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
  \end{verbatim}
  \item Windows
  \begin{verbatim}
  C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
  \end{verbatim}
\end{itemize}
Output should be similar to the following depending on how many HSMs you have.

```
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.
```

The prompt changes to `aws-cloudhsm>` when cloudhsm_mgmt_util is running.

### Enable End-to-End Encryption

Starting with version 1.1.1 of the client and related software libraries, end-to-end encrypted communication between cloudhsm_mgmt_util and the HSMs in your cluster is enabled by default. If you are using an earlier version, you can use the `enable_e2e` command to enable end-to-end encryption each time you start cloudhsm_mgmt_util.

```
aws-cloudhsm> enable_e2e
E2E enabled on server 0(10.0.2.9)
E2E enabled on server 1(10.0.3.11)
E2E enabled on server 2(10.0.1.12)
```

### Log in to the HSMs

Use the `loginHSM` command to log in to the HSMs. Any user of any type can use this command to log in to the HSMs.

The command in the following example logs in `admin`, which is the default crypto officer (CO) (p. 10). You set this user's password when you activated the cluster (p. 38). The output shows that the command logged in the `admin` user to all of the HSMs in the cluster.

**Warning**

When you log in to cloudhsm_mgmt_util, verify that the ENI IP addresses in the success messages exactly match the ENI IP addresses of all HSMs in the cluster. If they do not, stop and run all steps in the the section called “Prepare to run cloudhsm_mgmt_util” (p. 76) procedure.

To get the ENI IP addresses of the HSMs in your cluster, use the `DescribeClusters` operation, the `describe-clusters` command, or the Get-HSM2Cluster PowerShell cmdlet.

```
aws-cloudhsm> loginHSM CO admin <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)
```

The following shows the syntax for the `loginHSM` command.

```
aws-cloudhsm> loginHSM <user type> <user name> <password>
```

### Log Out from the HSMs

Use the `logoutHSM` command to log out of the HSMs.

```
aws-cloudhsm> logoutHSM
```
Stop cloudhsm_mgmt_util

Use the `quit` command to stop cloudhsm_mgmt_util.

```
aws-cloudhsm> quit
disconnecting from servers, please wait...
```

Using cloudhsm_mgmt_util Across Cloned Clusters

In some cases, you will use cloudhsm_mgmt_util to synchronize changes across cloned clusters. In order to do so, you will need to manually create a new cloudhsm_mgmt_util configuration file that specifies the HSMs you want to sync. For this example, create a copy of your current config file (`/opt/cloudhsm/etc/cloudhsm_mgmt_config.cfg`) and change the copy's name to `syncConfig.cfg`.

Edit `syncConfig.cfg` to include the Elastic Network Interface (ENI) IPs of the HSMs to be synced. We recommend that you specify the source HSM first, followed by the destination HSMs. To find the ENI IP of an HSM, see Update the AWS CloudHSM Configuration Files (p. 77).

Initialize cloudhsm_mgmt_util with the new config file by issuing the following command:

```
aws-cloudhsm> /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/userSync.cfg
```

Check the status messages returned to ensure that the cloudhsm_mgmt_util is connected to all desired HSMs and determine which of the returned ENI IPs corresponds to each cluster.

When you are done synchronizing HSMs or clusters, initialize cloudhsm_mgmt_util with the original configuration file.

```
aws-cloudhsm> /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_config.cfg
```

cloudhsm_mgmt_util Command Reference

The cloudhsm_mgmt_util command line tool helps crypto officers manage users in the HSMs. It also includes commands that allow crypto users (CUs) to share keys, and get and set key attributes. These commands complement the primary key management commands in the `key_mgmt_util` command line tool.

For a quick start, see Getting Started with cloudhsm_mgmt_util (p. 76).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80), enable end-to-end encryption (p. 81), and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

To list all cloudhsm_mgmt_util commands, run the following command:

```
aws-cloudhsm> help
```

To get the syntax for a cloudhsm_mgmt_util command, run the following command:

```
aws-cloudhsm> help <command-name>
```
Note
Use the syntax as per the documentation. While the built-in software help may provide additional options, these should not be considered supported and should not be utilized in production code.

To run a command, enter the command name, or enough of the name to distinguish it from the names of other cloudhsm_mgmt_util commands.

For example, to get a list of users on the HSMs, enter `listUsers` or `listU`.

```
aws-cloudhsm> listUsers
```

To end your cloudhsm_mgmt_util session, run the following command:

```
aws-cloudhsm> quit
```

For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

The following topics describe commands in cloudhsm_mgmt_util.

**Note**
Some commands in key_mgmt_util and cloudhsm_mgmt_util have the same names. However, the commands typically have different syntax, different output, and slightly different functionality.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>User Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>changePswd (p. 84)</td>
<td>Changes the passwords of users on the HSMs. Any user can change their own password. COs can change anyone's password.</td>
<td>CO</td>
</tr>
<tr>
<td>createUser (p. 87)</td>
<td>Creates users of all types on the HSMs.</td>
<td>CO</td>
</tr>
<tr>
<td>deleteUser (p. 91)</td>
<td>Deletes users of all types from the HSMs.</td>
<td>CO</td>
</tr>
<tr>
<td>findAllKeys (p. 93)</td>
<td>Gets the keys that a user owns or shares. Also gets a hash of the key ownership and sharing data for all keys on each HSM.</td>
<td>CO, AU</td>
</tr>
<tr>
<td>getAttribute (p. 96)</td>
<td>Gets an attribute value for an AWS CloudHSM key and writes it to a file or stdout (standard output).</td>
<td>CU</td>
</tr>
<tr>
<td>getCert (p. 99)</td>
<td>Gets the certificate of a particular HSM and saves it in a desired certificate format.</td>
<td>All</td>
</tr>
<tr>
<td>getHSMInfo (p. 100)</td>
<td>Gets information about the hardware on which an HSM is running.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td>getKeyInfo (p. 102)</td>
<td>Gets owners, shared users, and the quorum authentication status of a key.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>User Type</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>info (p. 105)</td>
<td>Gets information about an HSM, including the IP address, hostname, port, and current user.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td>listUsers (p. 108)</td>
<td>Gets the users in each of the HSMs, their user type and ID, and other attributes.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td>loginHSM and logoutHSM (p. 109)</td>
<td>Log in and log out of an HSM.</td>
<td>All.</td>
</tr>
<tr>
<td>quit (p. 115)</td>
<td>Quits cloudhsm_mgmt_util.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td>server (p. 111)</td>
<td>Enters and exits server mode on an HSM.</td>
<td>All.</td>
</tr>
<tr>
<td>setAttribute (p. 112)</td>
<td>Changes the values of the label, encrypt, decrypt, wrap, and unwrap attributes of an existing key.</td>
<td>CU</td>
</tr>
<tr>
<td>shareKey (p. 116)</td>
<td>Shares an existing key with other users.</td>
<td>CU</td>
</tr>
<tr>
<td>syncKey (p. 118)</td>
<td>Syncs a key across cloned AWS CloudHSM clusters.</td>
<td>CU, CO</td>
</tr>
<tr>
<td>syncUser (p. 120)</td>
<td>Syncs a user across cloned AWS CloudHSM clusters.</td>
<td>CO</td>
</tr>
</tbody>
</table>

**changePswd**

The *changePswd* command in cloudhsm_mgmt_util changes the password of an existing user on the HSMs in the cluster.

Any user can change their own password. In addition, Crypto officers (COs and PCOs) can change the password of another CO, crypto user (CU), or application user (AU). You do not need to enter the current password to make the change.

**Note**

You cannot change the password of a user who is currently logged into the AWS CloudHSM client or key_mgmt_util.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following users can run this command.

- Crypto officers (CO)
• Crypto users (CU)

**Syntax**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
changePswd <user-type> <user-name> <password>
```

**Examples**

The following examples show how to use `changePswd` to reset the password for the current user or any other user in your HSMs.

**Example: Change Your Password**

Any user on the HSMs can use `changePswd` to change their own password. Before you change the password, use `info (p. 105)` to get information about each of the HSMs in the cluster, including the username and the user type of the logged in user.

The following output shows that Bob is currently logged in as a crypto user (CU).

```
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)'
```

To change password, Bob runs `changePswd` followed with the user type, username, and a new password.

```
aws-cloudhsm> changePswd CU bob newPassword
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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: Change the Password of Another User**

You must be a CO or PCO to change the password of another CO, CU, or AU on the HSMs. Before you change the password for another user, use the `info (p. 105)` command to confirm that your user type is either CO or PCO.
The following output confirms that Alice, who is a CO, is currently logged in.

```
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-ogi3syxwxbqw Logged in
as 'alice(CO)'
```

Alice wants to reset the password of another user, John. Before she changes the password, she uses the `listUsers (p. 108)` command to verify John's user type.

The following output lists John as a CO user.

```
aws-cloudhsm> listUsers
Users on server 0(10.1.9.193):
Number of users found:5

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>jane</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>CU</td>
<td>bob</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>CU</td>
<td>alice</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CO</td>
<td>john</td>
<td>NO</td>
<td>0</td>
</tr>
</tbody>
</table>

Users on server 1(10.1.10.7):
Number of users found:5

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>jane</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>CU</td>
<td>bob</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CO</td>
<td>john</td>
<td>NO</td>
<td>0</td>
</tr>
</tbody>
</table>
```

To change the password, Alice runs `changePswd` followed with John's user type, username, and a new password.

```
aws-cloudhsm> changePswd CO john newPassword
86
```
Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```bash
changePswd <user-type> <user-name> <password>
```

 `<user-type>`

  Specifies the current type of the user whose password you are changing. You cannot use `changePswd` to change the user type.

  Valid values are `CO`, `CU`, `AU`, `PCO`, and `PRECO`.

  To get the user type, use `listUsers (p. 108)`. For detailed information about the user types on an HSM, see `HSM Users (p. 10)`.

  Required: Yes

 `<user-name>`

  Specifies the user's friendly name. This parameter is not case-sensitive. You cannot use `changePswd` to change the user name.

  Required: Yes

 `<password>`

  Specifies a new password for the user. Enter a string of 7 to 32 characters. This value is case sensitive. The password appears in plaintext when you type it.

  Required: Yes

Related Topics

- `info (p. 105)`
- `listUsers (p. 108)`
- `createUser (p. 87)`
- `deleteUser (p. 91)`

createUser

The `createUser` command in cloudhsm_mgmt_util creates a user on the HSMs. Only crypto officers (COs and PCOs) can run this command. When you create a user, you specify the user type (CO or CU), a user name, and a password. When the command succeeds, it creates the user in all HSMs in the cluster.

However, if your HSM configuration is inaccurate, the user might not be created on all HSMs. To add the user to any HSMs in which it is missing, use the `syncUser (p. 120)` or `createUser (p. 87)` command.
only on the HSMs that are missing that user. To prevent configuration errors, run the configure (p. 207) tool with the \(-m\) option.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following types of users can run this command.

- Crypto officers (CO, PCO)

**Syntax**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

**User Type**: Crypto officer (CO, PCO)

```
createUser <user-type> <user-name> <password>
```

**Examples**

These examples show how to use `createUser` to create new users in your HSMs.

**Example: Create a Crypto Officer**

This example creates a crypto officer (CO) on the HSMs in a cluster. The first command uses `loginHSM` (p. 109) to log in to the HSM as a crypto officer.

```
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)
```

The second command uses the `createUser` command to create `alice`, a new crypto officer on the HSM.

The caution message explains that the command creates users on all of the HSMs in the cluster. But, if the command fails on any HSMs, the user will not exist on those HSMs. To continue, type `y`.

The output shows that the new user was created on all three HSMs in the cluster.

```
aws-cloudhsm> createUser CO alice 391019314
*************************CAUTION*******************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes 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

When the command completes, alice has the same permissions on the HSM as the admin CO user, including changing the password of any user on the HSMs.

The final command uses the listUsers (p. 108) command to verify that alice exists on all three HSMs on the cluster. The output also shows that alice is assigned user ID 3. You use the user ID to identify alice in other commands, such as findAllKeys (p. 93).

aws-cloudhsm> listUsers
Users on server 0 (10.0.0.1):
Number of users found: 3

User Id         User Type User Name          MofnPubKey
LoginFailureCnt User Type       User Name                          MofnPubKey
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 User Type       User Name                          MofnPubKey
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 User Type       User Name                          MofnPubKey
1              PCO             admin                                   YES
0               NO
2              AU              app_user                                 NO
0               NO
3              CO              alice                                    NO
0               NO

Example: Create a Crypto User

This example creates a crypto user (CU), bob, on the HSM. Crypto users can create and manage keys, but they cannot manage users.

After you type y to respond to the caution message, the output shows that bob was created on all three HSMs in the cluster. The new CU can log in to the HSM to create and manage keys.

The command used a password value of defaultPassword. Later, bob or any CO can use the changePswd (p. 84) command to change his password.

aws-cloudhsm> createUser CU bob defaultPassword

*************************CAUTION**************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

createUser <user-type> <user-name> <password>

$user-type$

Specifies the type of user. This parameter is required.

For detailed information about the user types on an HSM, see HSM Users (p. 10).

Valid values:
- **CO**: Crypto officers can manage users, but they cannot manage keys.
- **CU**: Crypto users can create and manage keys and use keys in cryptographic operations.
- **AU**: Appliance users can clone and synchronize operations. One AU is created for you on each HSM that you install.

PCO, PRECO, and preCO are also valid values, but they are rarely used. A PCO is functionally identical to a CO user. A PRECO user is a temporary type that is created automatically on each HSM. The PRECO is converted to a PCO when you assign a password during HSM activation (p. 38).

Required: Yes

$user-name$

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

$password$

Specifies a password for the user. Enter a string of 7 to 32 characters. This value is case-sensitive. The password appears in plaintext when you type it.

To change a user password, use changePswd (p. 84). Any HSM user can change their own password, but CO users can change the password of any user (of any type) on the HSMs.

Required: Yes

Related Topics

- listUsers (p. 108)
deleteUser

The `deleteUser` command in cloudhsm_mgmt_util deletes a user from the HSMs. Only crypto officers (COs and PCOs) can run this command, but any CO user can delete any user of any type from the HSMs. However, you cannot delete a user who is logged into the AWS CloudHSM client, key_mgmt_util, or cloudhsm_mgmt_util.

**Warning**
When you delete a crypto user (CU), all keys that the user owned are deleted, even if the keys were shared with other users. To make accidental or malicious deletion of users less likely, use quorum authentication (p. 65).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**
The following types of users can run this command.

- Crypto officers (CO, PCO)

**Syntax**
Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
deleteUser <user-type> <user-name>
```

**Example**
This example deletes a crypto officer (CO) from the HSMs in a cluster. The first command uses listUsers (p. 108) to list all users on the HSMs.

The output shows that user 3, alice, is a CO on the HSMs.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
</tbody>
</table>
```
Users on server 1(10.0.0.2):
Number of users found:3

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.3):
Number of users found:3

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
</tbody>
</table>

The second command uses the `deleteUser` command to delete *alice* from the HSMs.

The output shows that the command succeeded on all three HSMs in the cluster.

```
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)
```

The final command uses the `listUsers` command to verify that *alice* is deleted from all three of the HSMs on the cluster.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:2

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.2):
Number of users found:2

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.3):
Number of users found:2

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
</tbody>
</table>
```
Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
deleteUser <user-type> <user-name>
```

**<user-type>**

Specifies the type of user. This parameter is required.

**Warning**
When you delete a crypto user (CU), all keys that the user owned are deleted, even if the keys were shared with other users. To make accidental or malicious deletion of users less likely, use quorum authentication (p. 65).

Valid values are **CO, CU, AU, PCO**, and **PRECO**.

To get the user type, use listUsers (p. 108). For detailed information about the user types on an HSM, see HSM Users (p. 10).

Required: Yes

**<user-name>**

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

Related Topics

- listUsers (p. 108)
- createUser (p. 87)
- syncUser (p. 120)
- changePswd (p. 84)

**findFirstKeys**

The **findFirstKeys** command in cloudhsm_mgmt_util gets the keys that a specified crypto user (CU) owns or shares. It also returns a hash of the user data on each of the HSMs. You can use the hash to determine at a glance whether the users, key ownership, and key sharing data are the same on all HSMs in the cluster. In the output, the keys owned by the user are annotated by (o) and shared keys are annotated by (s).

findFirstKeys returns public keys only when the specified CU owns the key, even though all CUs on the HSM can use any public key. This behavior is different from findKey (p. 141) in key_mgmt_util, which returns public keys for all CU users.
Only crypto officers (COs and PCOs) and appliance users (AUs) can run this command. Crypto users (CUs) can run the following commands:

- `listUsers (p. 108)` to find all users
- `findKey (p. 141)` in `key_mgmt_util` to find the keys that they can use
- `getKeyInfo (p. 169)` in `key_mgmt_util` to find the owner and shared users of a particular key they own or share

Before you run any `cloudhsm_mgmt_util` command, you must start `cloudhsm_mgmt_util` (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following users can run this command.

- Crypto officers (CO, PCO)
- Appliance users (AU)

**Syntax**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

**Examples**

These examples show how to use `findAllKeys` to find all keys for a user and get a hash of key user information on each of the HSMs.

**Example : Find the Keys for a CU**

This example uses `findAllKeys` to find the keys in the HSMs that user 4 owns and shares. The command uses a value of 0 for the second argument to suppress the hash value. Because it omits the optional file name, the command writes to stdout (standard output).

The output shows that user 4 can use 6 keys: 8, 9, 17, 262162, 19, and 31. The output uses an (s) to indicate keys that are explicitly shared by the user. The keys that the user owns are indicated by an (o) and include symmetric and private keys that the user does not share, and public keys that are available to all crypto users.

```bash
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
```
Example: Verify That User Data Is Synchronized

This example uses `findAllKeys` to verify that all of the HSMs in the cluster contain the same users, key ownership, and key sharing values. To do this, it gets a hash of the key user data on each HSM and compares the hash values.

To get the key hash, the command uses a value of 1 in the second argument. The optional file name is omitted, so the command writes the key hash to stdout.

The example specifies user 6, but the hash value will be the same for any user that owns or shares any of the keys on the HSMs. If the specified user does not own or share any keys, such as a CO, the command does not return a hash value.

The output shows that the key hash is identical to both of the HSMs in the cluster. If one of the HSM had different users, different key owners, or different shared users, the key hash values would not be equal.

```
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)
```

This command demonstrates that the hash value represents the user data for all keys on the HSM. The command uses the `findAllKeys` for user 3. Unlike user 6, who owns or shares just 3 keys, user 3 own or shares 17 keys, but the key hash value is the same.

```
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),262179(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),262179(o)
```
Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```plaintext
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

**<user id>**

- Gets all keys that the specified user owns or shares. Enter the user ID of a user on the HSMs. To find the user IDs of all users, use `listUsers` (p. 108).

- All user IDs are valid, but `findAllKeys` returns keys only for crypto users (CUs).

  Required: Yes

**<key hash>**

- Includes (1) or excludes (0) a hash of the user ownership and sharing data for all keys in each HSM.

  When the `user id` argument represents a user who owns or shares keys, the key hash is populated. The key hash value is identical for all users who own or share keys on the HSM, even though they own and share different keys. However, when the `user id` represents a user who does not own or share any keys, such as a CO, the hash value is not populated.

  Required: Yes

**<output file>**

- Writes the output to the specified file.

  Required: No

  Default: Stdout

Related Topics

- `changePswd` (p. 84)
- `deleteUser` (p. 91)
- `listUsers` (p. 108)
- `syncUser` (p. 120)
- `findKey` (p. 141) in `key_mgmt_util`
- `getKeyInfo` (p. 169) in `key_mgmt_util`

getAttribute

The `getAttribute` command in `cloudhsm_mgmt_util` gets one attribute value for a key from all HSMs in the cluster and writes it to stdout (standard output) or to a file. Only crypto users (CUs) can run this command.
**Key attributes** are properties of a key. They include characteristics, like the key type, class, label, and ID, and values that represent actions that you can perform on the key, like encrypt, decrypt, wrap, sign, and verify.

You can use `getAttribute` only on keys that you own and key that are shared with you. You can run this command or the `getAttribute` (p. 96) command in key_mgmt_util, which writes one or all of the attribute values of a key to a file.

To get a list of attributes and the constants that represent them, use the `listAttributes` (p. 187) command. To change the attribute values of existing keys, use `setAttribute` (p. 190) in key_mgmt_util and `setAttribute` (p. 112) in cloudhsm_mgmt_util. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

### User Type

The following users can run this command.

- Crypto users (CU)

### Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
getAttribute <key handle> <attribute id> [<filename>]
```

### Example

This example gets the value of the extractable attribute for a key in the HSMs. You can use a command like this to determine whether you can export a key from the HSMs.

The first command uses `listAttributes` (p. 106) to find the constant that represents the extractable attribute. The output shows that the constant for `OBJ_ATTR_EXTRACTABLE` is 354. You can also find this information with descriptions of the attributes and their values in the Key Attribute Reference (p. 204).

```
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
```
The second command uses `getAttribute` to get the value of the extractable attribute for the key with key handle 262170 in the HSMs. To specify the extractable attribute, the command uses 354, the constant that represents the attribute. Because the command does not specify a file name, `getAttribute` writes the output to stdout.

The output shows that the value of the extractable attribute is 1 on all of the HSM. This value indicates that the owner of the key can export it. When the value is 0 (0x0), it cannot be exported from the HSMs. You set the value of the extractable attribute when you create a key, but you cannot change it.

```
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

**Arguments**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
getAttribute <key handle> <attribute id> [<filename>]
```

**<key-handle>**

Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use `findKey` (p. 141) in key_mgmt_util.

You must own the specified key or it must be shared with you. To find the users of a key, use `getKeyinfo` (p. 169) in key_mgmt_util.
Required: Yes

<attribute id>

Identifies the attribute. Enter a constant that represents an attribute, or 512, which represents all attributes. For example, to get the key type, enter 256, which is the constant for the OBJ_ATTR_KEY_TYPE attribute.

To list the attributes and their constants, use listAttributes (p. 187). For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

Required: Yes

<filename>

Writes the output to the specified file. Enter a file path.

If the specified file exists, getAttribute overwrites the file without warning.

Required: No

Default: Stdout

Related Topics

- getAttribute (p. 164) in key_mgmt_util
- listAttributes (p. 106)
- setAttribute (p. 112) in cloudhsm_mgmt_util
- setAttribute (p. 190) in key_mgmt_util
- Key Attribute Reference (p. 204)

getCert

With the getCert command in cloudhsm_mgmt_util, you can retrieve the certificates of a particular HSM in a cluster. When you run the command, you designate the type of certificate to retrieve. To do that, you use one of the corresponding integers as described in the Arguments (p. 100) section below. To learn about the role of each of these certificates, see Verify HSM Identity (p. 23).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User Type

The following users can run this command.

- All users.

Prerequisites

Before you begin, you must enter server mode on the target HSM. For more information, see server (p. 111).
Syntax

To use the `getCert` command once in server mode:

```
server> getCert <file-name> <certificate-type>
```

Example

First, enter server mode. This command enters server mode on an HSM with server number 0.

```
aws-cloudhsm> server 0
Server is in 'E2' mode...
```

Then, use the `getCert` command. In this example, we use `/tmp/PO.crt` as the name of the file to which the certificate will be saved and 4 (Customer Root Certificate) as the desired certificate type:

```
server0> getCert /tmp/PO.crt 4
getcert Success
```

Arguments

```
getcert <file-name> <certificate-type>
```

- `<file-name>`
  - Specifies the name of the file to which the certificate is saved.
  - Required: Yes

- `<certificate-type>`
  - An integer that specifies the type of certificate to retrieve. The integers and their corresponding certificate types are as follows:
    - 1 – Manufacturer Root Certificate
    - 2 – Manufacturer Hardware Certificate
    - 4 – Customer Root Certificate
    - 8 – Cluster Certificate (signed by Customer Root Certificate)
    - 16 – Cluster Certificate (chained to the Manufacturer Root Certificate)
  - Required: Yes

Related Topics

- Start `cloudhsm_mgmt_util` (p. 80)
- `server` (p. 111)

**getHSMInfo**

The `getHSMInfo` command in `cloudhsm_mgmt_util` gets information about the hardware on which each HSM runs, including the model, serial number, FIPS state, memory, temperature, and the version numbers of the hardware and firmware. The information also includes the server ID that `cloudhsm_mgmt_util` uses to refer to the HSM.
Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following types of users can run this command.

- All users. You do not have to be logged in to run this command.

**Syntax**

This command has no parameters.

getHSMInfo

**Example**

This example uses getHSMInfo to get information about the HSMs in the cluster.

```
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          :xxxxxxxxxxxxxxxx
...
getKeyInfo

The **getKeyInfo** command in the key_mgmt_util returns the HSM user IDs of users who can use the key, including the owner and crypto users (CU) with whom the key is shared. When quorum authentication is enabled on a key, **getKeyInfo** also returns the number of users who must approve cryptographic operations that use the key. You can run **getKeyInfo** only on keys that you own and keys that are shared with you.

When you run **getKeyInfo** on public keys, **getKeyInfo** returns only the key owner, even though all users of the HSM can use the public key. To find the HSM user IDs of users in your HSMs, use **listUsers**. To find the keys for a particular user, use **findKey** in key_mgmt_util. Crypto officers can use **findAllKeys** in cloudhsm_mgmt_util.

You own the keys that you create. You can share a key with other users when you create it. Then, to share or unshare an existing key, use **shareKey** in cloudhsm_mgmt_util.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util and log in to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following types of users can run this command.

- Crypto users (CU)

**Syntax**

```
getKeyInfo -k <key-handle> [<output file>]
```

**Examples**

These examples show how to use **getKeyInfo** to get information about the users of a key.

**Example : Get the Users for an Asymmetric Key**

This command gets the users who can use the AES (asymmetric) key with key handle 262162. The output shows that user 3 owns the key and has shares it with users 4 and 6.

Only users 3, 4, and 6 can run **getKeyInfo** on key 262162.

```
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):
```

Related Topics

- info (p. 105)
Example: Get the Users for a Symmetric Key Pair

These commands use `getKeyInfo` to get the users who can use the keys in an ECC (symmetric) key pair (p. 158). The public key has key handle 262179. The private key has key handle 262177.

When you run `getKeyInfo` on the private key (262177), it returns the key owner (3) and crypto users (CUs) 4, with whom the key is shared.

```
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
```

When you run `getKeyInfo` on the public key (262179), it returns only the key owner, user 3.

```
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
```

To confirm that user 4 can use the public key (and all public keys on the HSM), use the `-u` parameter of `findKey` (p. 141) in `key_mgmt_util`.

The output shows that user 4 can use both the public (262179) and private (262177) key in the key pair. User 4 can also use all other public keys and any private keys that they have created or that have been shared with them.
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: Get the Quorum Authentication Value (m_value) for a Key

This example shows how to get the m_value for a key. The m_value is the number of users in the quorum who must approve any cryptographic operations that use the key and operations to share the unshare the key.

When quorum authentication is enabled on a key, a quorum of users must approve any cryptographic operations that use the key. To enable quorum authentication and set the quorum size, use the -m_value parameter when you create the key.

This command uses `genSymKey` (p. 158) to create a 256-bit AES key that is shared with user 4. It uses the m_value parameter to enable quorum authentication and set the quorum size to two users. The number of users must be large enough to provide the required approvals.

The output shows that the command created key 10.

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

This command uses `getKeyInfo` in cloudhsm_mgmt_util to get information about the users of key 10. The output shows that the key is owned by user 3 and shared with user 4. It also shows that a quorum of two users must approve every cryptographic operation that uses the key.

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):
2 Users need to approve to use/manage this key

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

getKeyInfo -k <key-handle> <output file>

<key-handle>

Specifies the key handle of one key in the HSM. Enter the key handle of a key that you own or share. This parameter is required.

Required: Yes

<output file>

Writes the output to the specified file, instead of stdout. If the file exists, the command overwrites it without warning.

Required: No

Default: stdout

Related Topics

- getKeyInfo (p. 169) in key_mgmt_util
- findKey (p. 141) in key_mgmt_util
- findAllKeys (p. 93) in cloudhsm_mgmt_util
- listUsers (p. 108)
- shareKey (p. 116)

info

The info command in cloudhsm_mgmt_util gets information about each of the HSMs in the cluster, including the host name, port, IP address and the name and type of the user who is logged in to cloudhsm_mgmt_util on the HSM.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User Type

The following types of users can run this command.

- All users. You do not have to be logged in to run this command.
Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
info server <server ID>
```

Example

This example uses `info` to get information about an HSM in the cluster. The command uses 0 to refer to the first HSM in the cluster. The output shows the IP address, port, and the type and name of the current user.

```
aws-cloudhsm> info server 0
Id      Name                    Hostname         Port   State           Partition
LoginState
0       10.0.0.1                10.0.0.1        2225    Connected       hsm-udw0tkflab
Logged in as 'testuser(CU)'
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
info server <server ID>
```

**<server id>**

Specifies the server ID of the HSM. The HSMs are assigned ordinal numbers that represent the order in which they are added to the cluster, beginning with 0. To find the server ID of an HSM, use `getHSMInfo`.

Required: Yes

Related Topics

- `getHSMInfo (p. 100)`
- `loginHSM and logoutHSM (p. 109)`

listAttributes

The `listAttributes` command in cloudhsm_mgmt_util lists the attributes of an AWS CloudHSM key and the constants that represent them. You use these constants to identify the attributes in `getAttribute (p. 96)` and `setAttribute (p. 112)` commands.

For help interpreting the key attributes, see the `Key Attribute Reference (p. 204)`.

Before you run any `key_mgmt_util` command, you must `start key_mgmt_util (p. 123)` and `log in (p. 124)` to the HSM as a crypto user (CU).

User Type

The following users can run this command.

- All users. You do not have to be logged in to run this command.
Syntax

`listAttributes [-h]`

Example

This command lists the key attributes that you can get and change in `key_mgmt_util` and the constants that represent them. For help interpreting the key attributes, see the Key Attribute Reference (p. 204). To represent all attributes, use 512.

Command: `listAttributes`

**Description**

The following are all of the possible attribute values for `getAttribute`.

```plaintext
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_EKCV                    = 4099
OBJ_ATTR_WRAP_TEMPLATE          = 1073742353
OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
OBJ_ATTR_ALL                    = 512
```

Parameters

- **-h**

Displays help for the command.

Required: Yes

Related Topics

- `getAttribute (p. 96)`
- `setAttribute (p. 112)`
listUsers

The listUsers command in the cloudhsm_mgmt_util gets the users in each of the HSMs, along with their user type and other attributes. All types of users can run this command. You do not even need to be logged in to cloudhsm_mgmt_util to run this command.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User Type

The following types of users can run this command.

• All users. You do not need to be logged in to run this command.

Syntax

This command has no parameters.

```
listUsers
```

Example

This command lists the users on each of the HSMs in the cluster and displays their attributes. You can use the User ID attribute to identify users in other commands, such as deleteUser, changePswd, and findAllKeys.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:6

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2FA</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>NO</td>
<td>crypto_user1</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>NO</td>
<td>crypto_user2</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CO</td>
<td>officer1</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>NO</td>
<td>officer2</td>
<td>NO</td>
<td>0</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.2):
Number of users found:5

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2FA</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
The output includes the following user attributes:

- **User ID**: Identifies the user in key_mgmt_util and cloudhsm_mgmt_util (p. 75) commands.
- **User type (p. 10)**: Determines the operations that the user can perform on the HSM.
- **User Name**: Displays the user-defined friendly name for the user.
- **MofnPubKey**: Indicates whether the user has registered a key pair for signing quorum authentication tokens (p. 60).
- **LoginFailureCnt**: Indicates the number of times the user has unsuccessfully logged in.
- **2FA**: Indicates that the user has enabled multi-factor authentication.

### Related Topics

- listUsers (p. 188) in key_mgmt_util
- createUser (p. 87)
- deleteUser (p. 91)
- changePswd (p. 84)

### loginHSM and logoutHSM

You can use the **loginHSM** and **logoutHSM** commands in cloudhsm_mgmt_util to log in and out of each HSM in a cluster. Any user of any type can use these commands.

**Note**

If you exceed five incorrect login attempts, your account is locked out. If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts. To unlock the account, a cryptographic officer (CO) must reset your password using the changePswd (p. 84) command in cloudhsm_mgmt_util.

If you have more than one HSM in your cluster, you may be allowed additional incorrect login attempts before your account is locked out. This is because the CloudHSM client balances load across various HSMs. Therefore, the login attempt may not begin on the same HSM each time. If you are testing this functionality, we recommend you do so on a cluster with only one active HSM.

Before you run these cloudhsm_mgmt_util commands, you must **start cloudhsm_mgmt_util (p. 80)**.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective on all HSMs in the cluster.

### User Type

The following users can run these commands.

- Precrypto officer (PRECO)
- Crypto officer (CO)
- Crypto user (CU)
• Appliance user (AU)

Syntax

Because these commands do not have named parameters, you must enter the arguments in the order specified in the syntax diagrams.

```
loginHSM <user type> <user name> <password>
```

logoutHSM

Examples

These examples show how to use `loginHSM` and `logoutHSM` to log in and out of all HSMs in a cluster.

**Example : Log In to the HSMs in a Cluster**

This command logs in to all HSMs in a cluster with the credentials of a CO user named `admin` and a password of `co12345`. The output shows that the command was successful and that the user has connected to the HSMs (which, in this case, are `server 0` and `server 1`).

```
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 : Log Out of an HSM**

This command logs out of the HSMs that you are currently logged in to (which, in this case, are `server 0` and `server 1`). The output shows that the command was successful and that the user has disconnected from the HSMs.

```
aws-cloudhsm>logoutHSM
logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
```

Arguments

Because these commands do not have named parameters, you must enter the arguments in the order specified in the syntax diagrams.

```
loginHSM <user type> <user name> <password>
```

**<user type>**

Specifies the type of user who is logging in to the HSMs. For more information, see User Type (p. 109) above.

Required: Yes

**<user name>**

Specifies the user name of the user who is logging in to the HSMs.

Required: Yes
<password>

Specifies the password of the user who is logging in to the HSMs.

Required: Yes

Related Topics

- Getting Started with cloudhsm_mgmt_util (p. 76)
- Activate the Cluster (p. 38)

server

Normally, when you issue a command in cloudhsm_mgmt_util, the command effects all HSMs in the designated cluster (global mode). However, there may be circumstances for which you need to issue commands to a single HSM. For instance, in the event that automatic synchronization fails, you may need to sync keys and users on an HSM in order to maintain consistency across the cluster. You can use the server command in the cloudhsm_mgmt_util to enter server mode and interact directly with a particular HSM instance.

Upon successful initiation, the aws-cloudhsm> command prompt is replaced with the server> command prompt.

In order to exit server mode, use the exit command. Upon successful exit, you will be returned to the cloudhsm_mgmt_util command prompt.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80).

User Type

The following users can run this command.

- All users

Prerequisites

In order to enter server mode, you must first know the server number of the target HSM. Server numbers are listed in the trace output generated by cloudhsm_mgmt_util upon initiation. Server numbers are assigned in the same order that the HSMs appear in the configuration file. For this example, we assume that server 0 is the server that corresponds to the desired HSM.

Syntax

To start server mode:

server <server-number>

To exit server mode:

server> exit

Example

This command enters server mode on an HSM with server number 0.

aws-cloudhsm> server 0
Server is in 'E2' mode...

In order to exit server mode, use the **exit** command.

```bash
server0> exit
```

**Arguments**

<table>
<thead>
<tr>
<th>server</th>
<th>&lt;server-number&gt;</th>
</tr>
</thead>
</table>

**<server-number>**

- Specifies the server number of the target HSM.
- Required: Yes

There are no arguments for the **exit** command.

**Related Topics**

- Start cloudbhsm_mgmt_util (p. 80)
- syncKey (p. 118)
- createUser (p. 87)
- deleteUser (p. 91)

**setAttribute**

The **setAttribute** command in cloudbhsm_mgmt_util changes the value of the label, encrypt, decrypt, wrap, and unwrap attributes of a key in the HSMs. You can also use the **setAttribute** (p. 190) command in key_mgmt_util to convert a session key to a persistent key. You can only change the attributes of keys that you own.

Before you run any cloudbhsm_mgmt_util command, you must start cloudbhsm_mgmt_util (p. 80) and log in (p. 81) to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files (p. 76) that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following users can run this command.

- Crypto users (CU)

**Syntax**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```bash
setAttribute <key handle> <attribute id>
```
Example

This example shows how to disable the decrypt functionality of a symmetric key. You can use a command like this one to configure a wrapping key, which should be able to wrap and unwrap other keys but not encrypt or decrypt data.

The first step is to create the wrapping key. This command uses `genSymKey` (p. 158) in key_mgmt_util to generate a 256-bit AES symmetric key. The output shows that the new key has key handle 14.

```
$ 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
```

Next, we want to confirm the current value of the decrypt attribute. To get the attribute ID of the decrypt attribute, use `listAttributes` (p. 106). The output shows that the constant that represents the `OBJ_ATTR_DECRYPT` attribute is 261. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

```
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_ALWAYSSENSITIVE        = 357
   OBJ_ATTR_DESTROYABLE            = 370
   OBJ_ATTR_KCV                    = 371
   OBJ_ATTR_WRAP_WITH_TRUSTED      = 528
   OBJ_ATTR_EKCV                   = 4099
   OBJ_ATTR_WRAPTEMPLATE           = 1073742353
   OBJ_ATTR_UNWRAPTEMPLATE         = 1073742354
   OBJ_ATTR_ALL                    = 512
```

To get the current value of the decrypt attribute for key 14, the next command uses `getAttribute` (p. 96) in cloudhsm_mgmt_util.
The output shows that the value of the decrypt attribute is true (1) on both HSMs in the cluster.

```
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
```

This command uses `setAttribute` to change the value of the decrypt attribute (attribute 261) of key 14 to 0. This disables the decrypt functionality on the key.

The output shows that the command succeeded on both HSMs in the cluster.

```
aws-cloudhsm>setAttribute 14 261 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes 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)
```

The final command repeats the `getAttribute` command. Again, it gets the decrypt attribute (attribute 261) of key 14.

This time, the output shows that the value of the decrypt attribute is false (0) on both HSMs in the cluster.

```
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
```

Arguments

`setAttribute <key handle> <attribute id>`

**<key-handle>**

Specifies the key handle of a key that you own. You can specify only one key in each command. To get the key handle of a key, use `findKey` (p. 141) in `key_mgmt_util`. To find the users of a key, use `getKeyInfo` (p. 102).

Required: Yes

**<attribute id>**

Specifies the constant that represents the attribute that you want to change. You can specify only one attribute in each command. To get the attributes and their integer values,
use `listAttributes (p. 187)`. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

Valid values:

- 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`.

Required: Yes

**Related Topics**

- `setAttribute (p. 190)` in `key_mgmt_util`
- `getAttribute (p. 96)`
- `listAttributes (p. 106)`
- Key Attribute Reference (p. 204)

**quit**

The `quit` command in the cloudhsm_mgmt_util exits the cloudhsm_mgmt_util. Any user of any type can use this command.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 80).

**User Type**

The following users can run this command.

- All users. You do not need to be logged in to run this command.

**Syntax**

```bash
quitar
```

**Example**

This command exits cloudhsm_mgmt_util. Upon successful completion, you are returned to your regular command line. This command has no output parameters.

```bash
aws-cloudhsm> quit
```
The `shareKey` command in cloudhsm_mgmt_util shares and unshares keys that you own with other crypto users. Only the key owner can share and unshare a key. You can also share a key when you create it.

Users who share the key can use the key in cryptographic operations, but they cannot delete, export, share, or unshare the key, or change its attributes. When quorum authentication is enabled on a key, the quorum must approve any operations that share or unshare the key.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util and log in to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following types of users can run this command.

- Crypto users (CU)

**Syntax**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

**User Type**: Crypto user (CU)

```
shareKey <key handle> <user id> <share/unshare key?> 1/0
```

**Example**

The following examples show how to use `shareKey` to share and unshare keys that you own with other crypto users.

**Example : Share a Key**

This example uses `shareKey` to share an **ECC private key** that the current user owns with another crypto user on the HSMs. Public keys are available to all users of the HSM, so you cannot share or unshare them.

The first command uses `getKeyInfo` to get the user information for key 262177, an ECC private key on the HSMs.
The output shows that key 262177 is owned by user 3, but is not shared.

```
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
```

This command uses `shareKey` to share key 262177 with user 4, another crypto user on the HSMs. The final argument uses a value of 1 to indicate a share operation.

The output shows that the operation succeeded on both HSMs in the cluster.

```
aws-cloudhsm>shareKey 262177 4 1
*****************************************************************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes 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)
```

To verify that the operation succeeded, the example repeats the first `getKeyInfo` command.

The output shows that key 262177 is now shared with user 4.

```
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: Unshare a Key**

This example unshares a symmetric key, that is, it removes a crypto user from the list of shared users for the key.
This command uses `shareKey` to remove user 4 from the list of shared users for key 6. The final argument uses a value of 0 to indicate an unshare operation.

The output shows that the command succeeded on both HSMs. As a result, user 4 can no longer use key 6 in cryptographic operations.

```
aws-cloudhsm>shareKey 6 4 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes 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)
```

**Arguments**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
shareKey <key handle> <user id> <share/unshare key? 1/0>
```

**<key-handle>**

Specifies the key handle of a key that you own. You can specify only one key in each command. To get the key handle of a key, use `findKey` in `key_mgmt_util`. To verify that you own a key, use `getKeyInfo` (p. 102).

Required: Yes

**<user id>**

Specifies the user ID the crypto user (CU) with whom you are sharing or unsharing the key. To find the user ID of a user, use `listUsers` (p. 108).

Required: Yes

**<share 1 or unshare 0>**

To share the key with the specified user, type 1. To unshare the key, that is, to remove the specified user from the list of shared users for the key, type 0.

Required: Yes

**Related Topics**

- `getKeyInfo` (p. 102)

**syncKey**

You can use the `syncKey` command in `cloudhsm_mgmt_util` to manually synchronize keys across HSM instances within a cluster or across cloned clusters. In general, you will not need to use this command, as HSM instances within a cluster sync keys automatically. However, key synchronization across cloned clusters must be done manually. Cloned clusters are usually created in different AWS Regions in order to simplify the global scaling and disaster recovery processes.
You cannot use `syncKey` to synchronize keys across arbitrary clusters: one of the clusters must have been created from a backup of the other. Additionally, both clusters must have consistent CO and CU credentials in order for the operation to be successful. For more information, see HSM Users (p. 10).

To use `syncKey`, you must first create an AWS CloudHSM configuration file (p. 82) that specifies one HSM from the source cluster and one from the destination cluster. This will allow `cloudhsm_mgmt_util` to connect to both HSM instances. Use this configuration file to start `cloudhsm_mgmt_util` (p. 80). Then log in (p. 81) with the credentials of a CO or a CU who owns the keys you want to synchronize.

**User Type**

The following types of users can run this command.

- Crypto officers (CO)
- Crypto users (CU)

**Note**

COs can use `syncKey` on any keys, while CUs can only use this command on keys that they own. For more information, see HSM Users (p. 10).

**Prerequisites**

Before you begin, you must know the key handle of the key on the source HSM to be synchronized with the destination HSM. To find the key handle, use the `listUsers` (p. 108) command to list all identifiers for named users. Then, use the `findAllKeys` (p. 93) command to find all keys that belong to a particular user.

You also need to know the server IDs assigned to the source and destination HSMs, which are shown in the trace output returned by `cloudhsm_mgmt_util` upon initiation. These are assigned in the same order that the HSMs appear in the configuration file.

Follow the instructions in Using `cloudhsm_mgmt_util` Across Cloned Clusters (p. 82) and initialize `cloudhsm_mgmt_util` with the new config file. Then, enter server mode on the source HSM by issuing the `server` (p. 111) command.

**Syntax**

**Note**

To run `syncKey`, first enter server mode on the HSM which contains the key to be synchronized.

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

**User Type:** Crypto user (CU)

```
syncKey <key handle> <destination hsm>
```

**Example**

Run the `server` command to log into the source HSM and enter server mode. For this example, we assume that `server 0` is the source HSM.

```
aws-cloudhsm> server 0
```

Now run the `syncKey` command. In this example, we assume key 261251 is to be synced to `server 1`. 
**Arguments**

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```bash
aws-cloudhsm> syncKey 261251 1
syncKey success
```

**<key handle>**

Specifies the key handle of the key to sync. You can specify only one key in each command. To get the key handle of a key, use `findAllKeys (p. 93)` while logged in to an HSM server.

Required: Yes

**<destination hsm>**

Specifies the number of the server to which you are syncing a key.

Required: Yes

**Related Topics**

- listUsers (p. 108)
- findAllKeys (p. 93)
- describe-clusters in AWS CLI
- server (p. 111)

**syncUser**

You can use the `syncUser` command in cloudhsm_mgmt_util to manually synchronize crypto users (CUs) or crypto officers (COs) across HSM instances within a cluster or across cloned clusters. AWS CloudHSM does not automatically synchronize users. Generally, you manage users in global mode so that all HSMs in a cluster are updated together. You might need to use `syncUser` if an HSM is accidentally desynchronized (for example, due to password changes) or if you want to rotate user credentials across cloned clusters. Cloned clusters are usually created in different AWS Regions to simplify the global scaling and disaster recovery processes.

Before you run any cloudhsm_mgmt_util command, you must `start cloudhsm_mgmt_util (p. 80)` and `log in (p. 81)` to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

If you add or delete HSMs, `update the configuration files (p. 76)` that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

**User Type**

The following types of users can run this command.

- Crypto officers (CO)
Prerequisites

Before you begin, you must know the user ID of the user on the source HSM to be synchronized with the destination HSM. To find the user ID, use the listUsers (p. 108) command to list all users on the HSMs in a cluster.

You also need to know the server ID assigned to the source and destination HSMs, which are shown in the trace output returned by cloudhsm_mgmt_util upon initiation. These are assigned in the same order that the HSMs appear in the configuration file.

If you are synchronizing HSMs across cloned clusters, follow the instructions in Using cloudhsm_mgmt_util Across Cloned Clusters (p. 82) and initialize cloudhsm_mgmt_util with the new config file.

When you are ready to run syncUser, enter server mode on the source HSM by issuing the server (p. 111) command.

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
syncUser <user ID> <server ID>
```

Example

Run the server command to log into the source HSM and enter server mode. For this example, we assume that server 0 is the source HSM.

```
aws-cloudhsm> server 0
```

Now run the syncUser command. For this example, we assume that user 6 is the user to be synced, and server 1 is the destination HSM.

```
server 0> syncUser 6 1
ExtractMaskedObject: 0x0 !
InsertMaskedObject: 0x0 !
syncUser success
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
syncUser <user ID> <server ID>
```

`<user ID>`

Specifies the ID of the user to sync. You can specify only one user in each command. To get the ID of a user, use listUsers (p. 108).

- Required: Yes

`<server ID>`

Specifies the server number of the HSM to which you are syncing a user.

- Required: Yes
key_mgmt_util

The **key_mgmt_util** command line tool helps Crypto Users (CU) manage keys in the HSMs. It includes multiple commands that generate, delete, import, and export keys, get and set attributes, find keys, and perform cryptographic operations.

For a quick start, see **Getting Started with key_mgmt_util** (p. 122). For detailed information about the commands, see **key_mgmt_util Command Reference** (p. 125). For help interpreting the key attributes, see the **Key Attribute Reference** (p. 204).

To use **key_mgmt_util** if you are using Linux, connect to your client instance and then see **Install and Configure the AWS CloudHSM Client (Linux)** (p. 34). If you are using Windows, see **Install and Configure the AWS CloudHSM Client (Windows)** (p. 36).

Topics
- Getting Started with **key_mgmt_util** (p. 122)
- **key_mgmt_util Command Reference** (p. 125)

Getting Started with **key_mgmt_util**

AWS CloudHSM includes two command line tools with the **AWS CloudHSM client software** (p. 34). The **cloudhsm_mgmt_util** (p. 82) tool includes commands to manage HSM users. The **key_mgmt_util** (p. 125) tool includes commands to manage keys. To get started with the **key_mgmt_util** command line tool, see the following topics.

Topics
- Set Up **key_mgmt_util** (p. 122)
- Basic Usage of **key_mgmt_util** (p. 124)

If you encounter an error message or unexpected outcome for a command, see the **Troubleshooting AWS CloudHSM** (p. 340) topics for help. For details about the **key_mgmt_util** commands, see **key_mgmt_util Command Reference** (p. 125).

Set Up **key_mgmt_util**

Complete the following setup before you use **key_mgmt_util**.

Start the AWS CloudHSM Client

Before you use **key_mgmt_util**, you must start the AWS CloudHSM client. The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster. The **key_mgmt_util** tool uses the client connection to communicate with the HSMs in your cluster. Without it, **key_mgmt_util** doesn’t work.

To start the AWS CloudHSM client
Use the following command to start the AWS CloudHSM client.

**Amazon Linux**

```
$ sudo start cloudhsm-client
```

**Amazon Linux 2**

```
$ sudo service cloudhsm-client start
```

**CentOS 6**

```
$ sudo start cloudhsm-client
```

**CentOS 7**

```
$ sudo service cloudhsm-client start
```

**RHEL 6**

```
$ sudo start cloudhsm-client
```

**RHEL 7**

```
$ sudo service cloudhsm-client start
```

**Ubuntu 16.04 LTS**

```
$ sudo service cloudhsm-client start
```

**Windows**

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  ```
  C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
  ```

**Start key_mgmt_util**

After you start the AWS CloudHSM client, use the following command to start key_mgmt_util.

**Amazon Linux**

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

**Amazon Linux 2**

```
$ /opt/cloudhsm/bin/key_mgmt_util
```
RHEL 6

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 7

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 6

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 7

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 16.04 LTS

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

Windows

```bash
c:\Program Files\Amazon\CloudHSM>key_mgmt_util.exe
```

The prompt changes to `Command: when key_mgmt_util is running.

If the command fails, such as returning a Daemon socket connection error message, try updating your configuration file (p. 346).

**Basic Usage of key_mgmt_util**

See the following topics for the basic usage of the key_mgmt_util tool.

**Topics**
- Log In to the HSMs (p. 124)
- Log Out from the HSMs (p. 125)
- Stop key_mgmt_util (p. 125)

**Log In to the HSMs**

Use the `loginHSM` command to log in to the HSMs. The following command logs in as a crypto user (CU) (p. 10) named `example_user`. The output indicates a successful login for all three HSMs in the cluster.

```bash
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
```
The following shows the syntax for the `loginHSM` command.

Command: `loginHSM -u <user type> -s <username> -p <password>`

**Log Out from the HSMs**

Use the `logoutHSM` command to log out from the HSMs.

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

**Stop key_mgmt_util**

Use the `exit` command to stop `key_mgmt_util`.

Command: `exit`

**key_mgmt_util Command Reference**

The `key_mgmt_util` command line tool helps you to manage keys in the HSMs in your cluster, including creating, deleting, and finding keys and their attributes. It includes multiple commands, each of which is described in detail in this topic.

For a quick start, see Getting Started with `key_mgmt_util` (p. 122). For help interpreting the key attributes, see the Key Attribute Reference (p. 204). For information about the cloudhsm_mgmt_util command line tool, which includes commands to manage the HSM and users in your cluster, see cloudhsm_mgmt_util (p. 75).

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

To list all `key_mgmt_util` commands, type:

Command: `help`

To get help for a particular `key_mgmt_util` command, type:

Command: `<command-name> -h`

To end your `key_mgmt_util` session, type:

Command: `exit`

The following topics describe commands in `key_mgmt_util`.

**Note**

Some commands in `key_mgmt_util` and `cloudhsm_mgmt_util` have the same names. However, the commands typically have different syntax, different output, and slightly different functionality.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aesWrapUnwrap (p. 127)</td>
<td>Encrypts and decrypts the contents of a key in a file.</td>
</tr>
<tr>
<td>deleteKey (p. 130)</td>
<td>Deletes a key from the HSMs.</td>
</tr>
<tr>
<td>Error2String (p. 131)</td>
<td>Gets the error that corresponds to a key_mgmt_util hexadecimal error code.</td>
</tr>
<tr>
<td>exit (p. 132)</td>
<td>Exits the key_mgmt_util.</td>
</tr>
<tr>
<td>exportPrivateKey (p. 133)</td>
<td>Exports a copy of a private key from an to a file on disk.</td>
</tr>
<tr>
<td>exportPubKey (p. 134)</td>
<td>Exports a copy of a public key from an HSM to a file.</td>
</tr>
<tr>
<td>exSymKey (p. 136)</td>
<td>Exports a plaintext copy of a symmetric key from the HSMs to a file.</td>
</tr>
<tr>
<td>extractMaskedObject (p. 140)</td>
<td>Extracts a key from an HSM as a masked object file.</td>
</tr>
<tr>
<td>findKey (p. 141)</td>
<td>Search for keys by key attribute value.</td>
</tr>
<tr>
<td>findSingleKey (p. 145)</td>
<td>Verifies that a key exists on all HSMs in the cluster.</td>
</tr>
<tr>
<td>genDSAKeyPair (p. 145)</td>
<td>Generates a Digital Signing Algorithm (DSA) key pair in your HSMs.</td>
</tr>
<tr>
<td>genECCKeyPair (p. 150)</td>
<td>Generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs.</td>
</tr>
<tr>
<td>genPBEKey (p. 154)</td>
<td>(This command is not supported on the FIPS-validated HSMs.)</td>
</tr>
<tr>
<td>genRSAKeyPair (p. 154)</td>
<td>Generates an RSA asymmetric key pair in your HSMs.</td>
</tr>
<tr>
<td>genSymKey (p. 158)</td>
<td>Generates a symmetric key in your HSMs</td>
</tr>
<tr>
<td>getAttribute (p. 164)</td>
<td>Gets the attribute values for an AWS CloudHSM key and writes them to a file.</td>
</tr>
<tr>
<td>getCaviumPrivKey (p. 167)</td>
<td>Creates a fake PEM-format version of a private key and exports it to a file.</td>
</tr>
<tr>
<td>getCert (p. 168)</td>
<td>Retrieves an HSM's partitions certificates and saves them to a file.</td>
</tr>
<tr>
<td>getKeyInfo (p. 169)</td>
<td>Gets the HSM user IDs of users who can use the key.</td>
</tr>
<tr>
<td></td>
<td>If the key is quorum controlled, it gets the number of users in the quorum.</td>
</tr>
<tr>
<td>help (p. 172)</td>
<td>Displays help information about the commands available in key_mgmt_util.</td>
</tr>
<tr>
<td>importPrivateKey (p. 173)</td>
<td>Imports a private key into an HSM.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>importPubKey (p. 176)</td>
<td>Imports a public key into an HSM.</td>
</tr>
<tr>
<td>imSymKey (p. 178)</td>
<td>Imports a plaintext copy of a symmetric key from a file into the HSM.</td>
</tr>
<tr>
<td>insertMaskedObject (p. 184)</td>
<td>Inserts a masked object from a file on disk into an HSM contained by related cluster to the object's origin cluster. Related clusters are any clusters generated from a backup of the origin cluster (p. 46).</td>
</tr>
<tr>
<td>??? (p. 186)</td>
<td>Determines whether or not a given file contains a real private key or a fake PEM key.</td>
</tr>
<tr>
<td>listAttributes (p. 187)</td>
<td>Lists the attributes of an AWS CloudHSM key and the constants that represent them.</td>
</tr>
<tr>
<td>listUsers (p. 188)</td>
<td>Gets the users in the HSMs, their user type and ID, and other attributes.</td>
</tr>
<tr>
<td>loginHSM and logoutHSM (p. 189)</td>
<td>Log in and out of the HSMs in a cluster.</td>
</tr>
<tr>
<td>setAttribute (p. 190)</td>
<td>Converts a session key to a persistent key.</td>
</tr>
<tr>
<td>sign (p. 193)</td>
<td>Generate a signature for a file using a chosen private key.</td>
</tr>
<tr>
<td>unWrapKey (p. 195)</td>
<td>Imports a wrapped (encrypted) key from a file into the HSMs.</td>
</tr>
<tr>
<td>verify (p. 199)</td>
<td>Verifies whether a given key was used to sign a given file.</td>
</tr>
<tr>
<td>wrapKey (p. 201)</td>
<td>Exports an encrypted copy of a key from the HSM to a file.</td>
</tr>
</tbody>
</table>

**aesWrapUnwrap**

The **aesWrapUnwrap** command encrypts or decrypts the contents of a file on disk. This command is designed to wrap and unwrap encryption keys, but you can use it on any file that contains less than 4 KB (4096 bytes) of data.

**aesWrapUnwrap** uses AES Key Wrap. It uses an AES key on the HSM as the wrapping or unwrapping key. Then it writes the result to another file on disk.

Before you run any key_mgmt_util command, you must start **key_mgmt_util (p. 123)** and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
aesWrapUnwrap -h
```

```
aesWrapUnwrap -m <wrap-unwrap mode> -f <file-to-wrap-unwrap> -w <wrapping-key-handle> [-i <wrapping-IV>] [-out <output-file>]
```
Examples

These examples show how to use `aesWrapUnwrap` to encrypt and decrypt an encryption key in a file.

Example: Wrap an Encryption Key

This command uses `aesWrapUnwrap` to wrap a Triple DES symmetric key that was exported from the HSM in plaintext (p. 136) into the `3DES.key` file. You can use a similar command to wrap any key saved in a file.

The command uses the `-m` parameter with a value of 1 to indicate wrap mode. It uses the `-w` parameter to specify an AES key in the HSM (key handle 6) as the wrapping key. It writes the resulting wrapped key to the `3DES.key.wrapped` file.

The output shows that the command was successful and that the operation used the default IV, which is preferred.

```plaintext
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: Unwrap an Encryption Key

This example shows how to use `aesWrapUnwrap` to unwrap (decrypt) a wrapped (encrypted) key in a file. You might want to do an operation like this one before importing a key to the HSM. For example, if you try to use the `imSymKey` (p. 178) command to import an encrypted key, it returns an error because the encrypted key doesn't have the format that is required for a plaintext key of that type.

The command unwraps the key in the `3DES.key.wrapped` file and writes the plaintext to the `3DES.key.unwrapped` file. The command uses the `-m` parameter with a value of 0 to indicate unwrap mode. It uses the `-w` parameter to specify an AES key in the HSM (key handle 6) as the wrapping key. It writes the resulting wrapped key to the `3DES.key.unwrapped` file.

```plaintext
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
```
Parameters

-h
Displays help for the command.
Required: Yes

-m
Specifies the mode. To wrap (encrypt) the file content, type 1; to unwrap (decrypt) the file content, type 0.
Required: Yes

-f
Specifies the file to wrap. Enter a file that contains less than 4 KB (4096 bytes) of data. This operation is designed to wrap and unwrap encryption keys.
Required: Yes

-w
Specifies the wrapping key. Enter the key handle of an AES key or RSA key on the HSM. This parameter is required. To find key handles, use the findKey (p. 141) command.

To create a wrapping key, use genSymKey (p. 158) to generate an AES key (type 31) or genRSAKeyPair (p. 154) to generate an RSA key pair (type 0). If you are using an RSA key pair, be sure to wrap the key with one of the keys, and unwrap it with the other. To verify that a key can be used as a wrapping key, use getAttribute (p. 164) to get the value of the OBJ_ATTR_WRAP attribute, which is represented by constant 262.
Required: Yes

-i
Specifies an alternate initial value (IV) for the algorithm. Use the default value unless you have a special condition that requires an alternative.
Default: 0xA6A6A6A6A6A6A6A6. The default value is defined in the AES Key Wrap algorithm specification.
Required: No

-out
Specifies an alternate name for the output file that contains the wrapped or unwrapped key. The default is wrapped_key (for wrap operations) and unwrapped_key (for unwrap operations) in the local directory.
If the file exists, the aesWrapUnwrap overwrites it without warning. If the command fails, aesWrapUnwrap creates an output file with no contents.
Default: For wrap: wrapped_key. For unwrap: unwrapped_key.
Required: No

Related Topics
- exSymKey (p. 136)
- imSymKey (p. 178)
- unWrapKey (p. 195)
• wrapKey (p. 201)

deleteKey

The **deleteKey** command in key_mgmt_util deletes a key from the HSM. You can only delete one key at a time. Deleting one key in a key pair has no effect on the other key in the pair.

Only the key owner can delete a key. Users who share the key can use it in cryptographic operations, but not delete it.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
deleteKey -h
deleteKey -k
```

**Examples**

These examples show how to use **deleteKey** to delete keys from your HSMs.

**Example : Delete a Key**

This command deletes the key with key handle 6. When the command succeeds, **deleteKey** returns success messages from each HSM in the cluster.

<table>
<thead>
<tr>
<th>Command: deleteKey -k 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Cluster Error Status</td>
</tr>
<tr>
<td>Node id 1 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Node id 2 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

**Example : Delete a Key (Failure)**

When the command fails because no key has the specified key handle, **deleteKey** returns an invalid object handle error message.

<table>
<thead>
<tr>
<th>Command: deleteKey -k 252126</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfm3FindKey returned: 0xa8 : HSM Error: Invalid object handle is passed to this operation</td>
</tr>
<tr>
<td>Cluster Error Status</td>
</tr>
<tr>
<td>Node id 1 and err state 0x000000a8 : HSM Error: Invalid object handle is passed to this operation</td>
</tr>
<tr>
<td>Node id 2 and err state 0x000000a8 : HSM Error: Invalid object handle is passed to this operation</td>
</tr>
</tbody>
</table>

When the command fails because the current user is not the owner of the key, the command returns an access denied error.

<table>
<thead>
<tr>
<th>Command: deleteKey -k 262152</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied.</td>
</tr>
</tbody>
</table>
Parameters

-h
Displays command line help for the command.
Required: Yes

-k
Specifies the key handle of the key to delete. To find the key handles of keys in the HSM, use findKey (p. 141).
Required: Yes

Related Topics
- findKey (p. 141)

Error2String

The Error2String helper command in key_mgmt_util returns the error that corresponds to a key_mgmt_util hexadecimal error code. You can use this command when troubleshooting your commands and scripts.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

Error2String -h
Error2String -r <response-code>

Examples

These examples show how to use Error2String to get the error string for a key_mgmt_util error code.

Example : Get an Error Description

This command gets the error description for the 0xdb error code. The description explains that an attempt to log in to key_mgmt_util failed because the user has the wrong user type. Only crypto users (CU) can log in to key_mgmt_util.

Command: Error2String -r 0xdb
Error Code db maps to HSM Error: Invalid User Type.

Example : Find the Error Code

This example shows where to find the error code in a key_mgmt_util error. The error code, 0xc6, appears after the string: Cfms3command-name returned: .

In this example, getKeyInfo (p. 169) indicates that the current user (user 4) can use the key in cryptographic operations. Nevertheless, when the user tries to use deleteKey (p. 130) to delete the key, the command returns error code 0xc6.
Command: \texttt{deleteKey -k 262162}

cfm3DeleteKey returned: \texttt{0xc6} : HSM Error: Key Access is denied

Cluster Error Status

Command: \texttt{getKeyInfo -k 262162}

cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

Owned by user 3
also, shared to following 1 user(s):

4

If the \texttt{0xc6} error is reported to you, you can use an \texttt{Error2String} command like this one to look up the error. In this case, the \texttt{deleteKey} command failed with an access denied error because the key is shared with the current user but owned by a different user. Only key owners have permission to delete a key.

Command: \texttt{Error2String -r 0xa8}

Error Code c6 maps to HSM Error: Key Access is denied

\textbf{Parameters}

\begin{itemize}
\item[-h]
Displays help for the command.

Required: Yes

\item[-r]
Specifies a hexadecimal error code. The 0x hexadecimal indicator is required.

Required: Yes
\end{itemize}

\textbf{exit}

The \texttt{exit} command in \texttt{key_mgmt_util} exits the \texttt{key_mgmt_util}. Upon successful exit, you will be returned to your standard command line.

Before you run any \texttt{key_mgmt_util} command, you must \texttt{start key_mgmt_util (p. 123)}.

\textbf{Syntax}

\texttt{exit}

\textbf{Parameters}

There are no parameters for this command.

\textbf{Related Topics}

- Start \texttt{key_mgmt_util (p. 123)}
**exportPrivateKey**

The `exportPrivateKey` command in `key_mgmt_util` exports an asymmetric private key in an HSM to a file. You can use it to export private keys that you generate on the HSM. You can also use the command to export private keys that were imported into an HSM, such as those imported with the `importPrivateKey` (p. 173) command.

During the export process, `exportPrivateKey` uses an AES key that you select (the *wrapping key*) to wrap (encrypt) the private key. This way, the private key file maintains integrity during transit. For more information, see `wrapKey` (p. 201).

The `exportPrivateKey` command copies the key material to a file that you specify. But it does not remove the key from the HSM, change its key attributes (p. 204), or prevent you from using the key in further cryptographic operations. You can export the same key multiple times.

You can only export private keys that have `OBJ_ATTR_EXTRACTABLE` attribute value 1. To find a key's attributes, use the `getAttribute` (p. 164) command.

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
exportPrivateKey -h

exportPrivateKey -k <private-key-handle>
  -w <wrapping-key-handle>
  -out <key-file>
  [-m <wrapping-mechanism>]
  [-wk <wrapping-key-file>]
```

**Examples**

This example shows how to use `exportPrivateKey` to export a private key out of an HSM.

**Example : Export a Private Key**

This command exports a private key with handle 15 using a wrapping key with handle 16 to a PEM file called `exportKey.pem`. When the command succeeds, `exportPrivateKey` returns a success message.

```
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
```

**Parameters**

This command takes the following parameters.

- **-h**
  
  Displays command line help for the command.
  
  Required: Yes
-k

Specifies the key handle of the private key to be exported.

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey (p. 141) command.

To determine whether a key can be used as a wrapping key, use getAttribute (p. 164) to get the value of the OBJ_ATTR_WRAP attribute (262). To create a wrapping key, use genSymKey (p. 158) to create an AES key (type 31).

If you use the -wk parameter to specify an external unwrapping key, the -w wrapping key is used to wrap, but not unwrap, the key during export.

Required: Yes

-out

Specifies the name of the file to which the exported private key will be written.

Required: Yes

-m

Specifies the wrapping mechanism with which to wrap the private key being exported. The only valid value is 4, which represents the NIST_AES_WRAP mechanism.

Default: 4 (NIST_AES_WRAP)

Required: No

-wk

Specifies the key to be used to unwrap the key being exported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, exportPrivateKey uses the key in the -w file to wrap the key being exported and uses the key specified by the -wk parameter to unwrap it.

Default: Use the wrapping key specified in the -w parameter to both wrap and unwrap.

Required: No

Related Topics

- importPrivateKey (p. 173)
- wrapKey (p. 201)
- unWrapKey (p. 195)
- genSymKey (p. 158)

exportPubKey

The exportPubKey command in key_mgmt_util exports a public key in an HSM to a file. You can use it to export public keys that you generate in an HSM. You can also use this command to export public keys that were imported into an HSM, such as those imported with the importPubKey (p. 176) command.
The `exportPubKey` operation copies the key material to a file that you specify. But it does not remove the key from the HSM, change its key attributes (p. 204), or prevent you from using the key in further cryptographic operations. You can export the same key multiple times.

You can only export public keys that have a `OBJ_ATTR_EXTRACTABLE` value of 1. To find a key’s attributes, use the `getAttribute (p. 164)` command.

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util (p. 123)` and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
exportPubKey -h
exportPubKey -k <public-key-handle> -out <key-file>
```

**Examples**

This example shows how to use `exportPubKey` to export a public key from an HSM.

**Example : Export a Public Key**

This command exports a public key with handle 10 to a file called `public.pem`. When the command succeeds, `exportPubKey` returns a success message.

```
Command: exportPubKey -k 10 -out public.pem
Pem formatted public key is written to public.pem
Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS
```

**Parameters**

This command takes the following parameters.

- `-h`
  
  Displays command line help for the command.
  
  Required: Yes

- `-k`
  
  Specifies the key handle of the public key to be exported.
  
  Required: Yes

- `-out`
  
  Specifies the name of the file to which the exported public key will be written.
  
  Required: Yes

**Related Topics**

- `importPubKey (p. 176)`
- `Generate Keys (p. 55)`
**exSymKey**

The `exSymKey` command in the `key_mgmt_util` tool exports a plaintext copy of a symmetric key from the HSM and saves it in a file on disk. To export an encrypted (wrapped) copy of a key, use `wrapKey` (p. 201). To import a plaintext key, like the ones that `exSymKey` exports, use `imSymKey` (p. 178).

During the export process, `exSymKey` uses an AES key that you specify (the *wrapping key*) to wrap (encrypt) and then unwrap (decrypt) the key to be exported. However, the result of the export operation is a plaintext (*unwrapped*) key on disk.

Only the owner of a key, that is, the CU user who created the key, can export it. Users who share the key can use it in cryptographic operations, but they cannot export it.

The `exSymKey` operation copies the key material to a file that you specify, but it does not remove the key from the HSM, change its key attributes (p. 204), or prevent you from using the key in cryptographic operations. You can export the same key multiple times.

`exSymKey` exports only symmetric keys. To export public keys, use `exportPubKey` (p. 134). To export private keys, use `exportPrivateKey` (p. 133).

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
exSymKey -h
exSymKey -k <key-to-export>
   -w <wrapping-key>
   -out <key-file>
   [-m 4]
   [-wk <unwrapping-key-file> ]
```

**Examples**

These examples show how to use `exSymKey` to export symmetric keys that you own from your HSMs.

**Example : Export a 3DES Symmetric Key**

This command exports a Triple DES (3DES) symmetric key (key handle 7). It uses an existing AES key (key handle 6) in the HSM as the wrapping key. Then it writes the plaintext of the 3DES key to the 3DES.key file.

The output shows that key 7 (the 3DES key) was successfully wrapped and unwrapped, and then written to the 3DES.key file.

**Warning**

Although the output says that a "Wrapped Symmetric Key" was written to the output file, the output file contains a plaintext (unwrapped) key.

```
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: Exporting with Session-Only Wrapping Key

This example shows how to use a key that exists only in the session as the wrapping key. Because the key to be exported is wrapped, immediately unwrapped, and delivered as plaintext, there is no need to retain the wrapping key.

This series of commands exports an AES key with key handle 8 from the HSM. It uses an AES session key created especially for the purpose.

The first command uses `genSymKey (p. 158)` to create a 256-bit AES key. It uses the `-sess` parameter to create a key that exists only in the current session.

The output shows that the HSM creates key 262168.

```plaintext
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
```

Next, the example verifies that key 8, the key to be exported, is a symmetric key that is extractable. It also verifies that the wrapping key, key 262168, is an AES key that exists only in the session. You can use the `findKey (p. 141)` command, but this example exports the attributes of both keys to files and then uses `grep` to find the relevant attribute values in the file.

These commands use `getAttribute` with an `-a` value of 512 (all) to get all attributes for keys 8 and 262168. For information about the key attributes, see the section called "Key Attribute Reference" (p. 204).

```plaintext
getAttribute -o 8 -a 512 -out attributes/attr_8
getAttribute -o 262168 -a 512 -out attributes/attr_262168
```

These commands use `grep` to verify the attributes of the key to be exported (key 8) and the session-only wrapping key (key 262168).

```plaintext
// 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
```
Finally, we use an **exSymKey** command to export key 8 using the session key (key 262168) as the wrapping key.

When the session ends, key 262168 no longer exists.

```
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 : Use an External Unwrapping Key**

This example shows how to use an external unwrapping key to export a key from the HSM.

When you export a key from the HSM, you specify an AES key on the HSM to be the wrapping key. By default, that wrapping key is used to wrap and unwrap the key to be exported. However, you can use the --wk parameter to tell **exSymKey** to use an external key in a file on disk for unwrapping. When you do, the key specified by the --w parameter wraps the target key, and the key in the file specified by the --wk parameter unwraps the key.

Because the wrapping key must be an AES key, which is symmetric, the wrapping key in the HSM and unwrapping key on disk must be have the same key material. To do this, you must import the wrapping key to the HSM or export the wrapping key from the HSM before the export operation.

This example creates a key outside of the HSM and imports it into the HSM. It uses the internal copy of the key to wrap a symmetric key that is being exported, and the copy of key in the file to unwrap it.

The first command uses OpenSSL to generate a 256-bit AES key. It saves the key to the `aes256-forImport.key` file. The OpenSSL command does not return any output, but you can use several commands to confirm its success. This example uses the `wc` (wordcount) tool, which confirms that the file contains 32 bytes of data.

```
$ openssl rand -out keys/aes256-forImport.key 32
$ wc keys/aes256-forImport.key
  0  2 32 keys/aes256-forImport.key
```

This command uses the **imSymKey** (p. 178) command to import the AES key from the `aes256-forImport.key` file to the HSM. When the command completes, the key exists in the HSM with key handle 262167 and in the `aes256-forImport.key` file.

```
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
```
This command uses the key in an export operation. The command uses `exSymKey` to export key 21, a 192-bit AES key. To wrap the key, it uses key 262167, which is the copy that was imported into the HSM. To unwrap the key, it uses the same key material in the `aes256-forImport.key` file. When the command completes, key 21 is exported to the `aes192_H21.key` file.

Parameter usage:

- **-h**
  Displays help for the command.
  Required: Yes

- **-k**
  Specifies the key handle of the key to export. This parameter is required. Enter the key handle of a symmetric key that you own. This parameter is required. To find key handles, use the `findKey` command.
  Required: Yes

- **-w**
  Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the `findKey` command.
  A **wrapping key** is a key in the HSM that is used to encrypt (wrap) and then decrypt (unwrap) the key to be exported. Only AES keys can be used as wrapping keys.
  You can use any AES key (of any size) as a wrapping key. Because the wrapping key wraps, and then immediately unwraps, the target key, you can use as session-only AES key as a wrapping key. To determine whether a key can be used as a wrapping key, use `getAttribute` to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by the constant 262. To create a wrapping key, use `genSymKey` to create an AES key (type 31).

  If you use the `-wk` parameter to specify an external unwrapping key, the `-w` wrapping key is used to wrap, but not to unwrap, the key during export.

  **Note**
  Key 4 represents an unsupported internal key. We recommend that you use an AES key that you create and manage as the wrapping key.
  Required: Yes

- **-out**
  Specifies the path and name of the output file. When the command succeeds, this file contains the exported key in plaintext. If the file already exists, the command overwrites it without warning.
Required: Yes

-\textbf{m}

Specifies the wrapping mechanism. The only valid value is 4, which represents the \texttt{NIST\_AES\_WRAP} mechanism.

Required: No
Default: 4

-\textbf{wk}

Use the AES key in the specified file to unwrap the key that is being exported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, \texttt{exSymKey} uses the key in the HSM that is specified by the \texttt{\-w} parameter to wrap the key that is being exported and it uses the key in the \texttt{\-wk} file to unwrap it. The \texttt{\-w} and \texttt{\-wk} parameter values must resolve to the same plaintext key.

Required: No
Default: Use the wrapping key on the HSM to unwrap.

\textbf{Related Topics}

- \texttt{genSymKey (p. 158)}
- \texttt{imSymKey (p. 178)}
- \texttt{wrapKey (p. 201)}

\textbf{extractMaskedObject}

The \texttt{extractMaskedObject} command in \texttt{key\_mgmt\_util} extracts a key from an HSM and saves it to a file as a masked object. Masked objects are \textit{cloned} objects that can only be used after inserting them back into the original cluster by using the \texttt{insertMaskedObject (p. 184)} command. You can only insert a masked object into the same cluster from which it was generated, or a clone of that cluster. This includes any cloned versions of the cluster generated by \texttt{copying a backup across regions (p. 45)} and \texttt{using that backup to create a new cluster (p. 46)}.

Masked objects are an efficient way to offload and synchronize keys, including nonextractable keys (that is, keys that have a \texttt{OBJ\_ATTR\_EXTRACTABLE (p. 204)} value of 0). This way, keys can be securely synced across related clusters in different regions without the need to update the AWS CloudHSM configure file (p. 207).

\textbf{Important}

Upon insertion, masked objects are decrypted and given a key handle that is different from the key handle of the original key. A masked object includes all metadata associated with the original key, including attributes, ownership and sharing information, and quorum settings. If you need to sync keys across clusters in an application, use \texttt{syncKey (p. 118)} in the \texttt{cloudhsm\_mgmt\_util} instead.

Before you run any \texttt{key\_mgmt\_util} command, you must \texttt{start key\_mgmt\_util (p. 123)} and \texttt{log in (p. 124)} to the HSM. The \texttt{extractMaskedObject} command can be used either by the CU who owns the key or any CO.

\textbf{Syntax}

\begin{verbatim}
extractMaskedObject -h
\end{verbatim}
extractMaskedObject -o <object-handle> -out <object-file>

Examples
This example shows how to use `extractMaskedObject` to extract a key from an HSM as a masked object.

Example: Extract a Masked Object
This command extracts a masked object out of an HSM from a key with handle 524295 and saves it as a file called maskedObj. When the command succeeds, `extractMaskedObject` returns a success message.

```
Command: extractMaskedObject -o 524295 -out maskedObj
Object was masked and written to file "maskedObj"
Cfm3ExtractMaskedObject returned: 0x00 : HSM Return: SUCCESS
```

Parameters
This command takes the following parameters.

- **-h**
  Displays command line help for the command.
  Required: Yes
- **-o**
  Specifies the handle of the key to extract as a masked object.
  Required: Yes
- **-out**
  Specifies the name of the file to which the masked object will be saved.
  Required: Yes

Related Topics
- `insertMaskedObject` (p. 184)
- `syncKey` (p. 118)
- Copying a Backup Across Regions (p. 45)
- Creating an AWS CloudHSM Cluster from a Previous Backup (p. 46)

**findKey**
Use the `findKey` command in key_mgmt_util to search for keys by the values of the key attributes. When a key matches all the criteria that you set, `findKey` returns the key handle. With no parameters, `findKey` returns the key handles of all the keys that you can use in the HSM. To find the attribute values of a particular key, use `getAttribute` (p. 164).

Like all key_mgmt_util commands, `findKey` is user specific. It returns only the keys that the current user can use in cryptographic operations. This includes keys that current user owns and keys that have been shared with the current user.
Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

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>]
```

Examples

These examples show how to use `findKey` to find and identify keys in your HSMs.

Example : Find All Keys

This command finds all keys for the current user in the HSM. The output includes keys that the user owns and shares, and all public keys in the HSMs.

To get the attributes of a key with a particular key handle, use `getAttribute` (p. 164). To determine whether the current user owns or shares a particular key, use `getKeyInfo` (p. 169) or `findAllKeys` (p. 93) in cloudhsm_mgmt_util.

```
Command: findKey
Total number of keys present 13
number of keys matched from start index 0:12
6, 7, 524296, 9, 262154, 262155, 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 : Find Keys by Type, User, and Session

This command finds persistent AES keys that the current user and user 3 can use. (User 3 might be able to use other keys that the current user cannot see.)

```
Command: findKey -t 31 -sess 0 -u 3
```

Example : Find Keys by Class and Label

This command finds all public keys for the current user with the 2018-sept label.

```
Command: findKey -c 2 -l 2018-sept
```

Example : Find RSA Keys by Modulus

This command finds RSA keys (type 0) for the current user that were created by using the modulus in the m4.txt file.
Command: `findKey -t 0 -m m4.txt`

**Parameters**

- **-h**
  
  Displays help for the command.
  
  Required: Yes

- **-t**
  
  Finds keys of the specified type. Enter the constant that represents the key class. For example, to find 3DES keys, type `-t 21`.

  Valid values:
  
  - 0: RSA
  - 1: DSA
  - 3: EC
  - 16: GENERIC_SECRET
  - 18: RC4
  - 21: Triple DES (3DES)
  - 31: AES

  Required: No

- **-c**
  
  Finds keys in the specified class. Enter the constant that represents the key class. For example, to find public keys, type `-c 2`.

  Valid values for each key type:
  
  - 2: Public. This class contains the public keys of public–private key pairs.
  - 3: Private. This class contains the private keys of public–private key pairs.
  - 4: Secret. This class contains all symmetric keys.

  Required: No

- **-l**
  
  Finds keys with the specified label. Type the exact label. You cannot use wildcard characters or regular expressions in the `-l` value.

  Required: No

- **-id**
  
  Finds the key with the specified ID. Type the exact ID string. You cannot use wildcard characters or regular expressions in the `-id` value.

  Required: No

- **-sess**
  
  Finds keys by session status. To find keys that are valid only in the current session, type 1. To find persistent keys, type 0.

  Required: No
-u
Finds keys the specified users and the current user share. Type a comma-separated list of HSM user IDs, such as `-u 3` or `-u 4, 7`. To find the IDs of users on an HSM, use `listUsers (p. 188)`. When you specify one user ID, `findKey` returns the keys for that user. When you specify multiple user IDs, `findKey` returns the keys that all the specified users can use.

Because `findKey` only returns keys that the current user can use, the `-u` results are always identical to or a subset of the current user’s keys. To get all keys that are owned by or shared with any user, crypto officers (COs) can use `findAllKeys (p. 93)` in cloudhsm_mgmt_util.

Required: No

-m
Finds keys that were created by using the RSA modulus in the specified file. Type the path to file that stores the modulus.

Required: No

-kcv
Finds keys with the specified key check value.

The key check value (KCV) is a 3-byte hash or checksum of a key that is generated when the HSM imports or generates a key. You can also calculate a KCV outside of the HSM, such as after you export a key. You can then compare the KCV values to confirm the identity and integrity of the key. To get the KCV of a key, use `getAttribute (p. 164)`.

AWS CloudHSM uses the following standard method to generate a key check value:
- Symmetric keys: First 3 bytes of the result of encrypting a zero-block with the key.
- Asymmetric key pairs: First 3 bytes of the SHA-1 hash of the public key.
- HMAC keys: KVC for HMAC keys is not supported at this time.

Required: No

Output

The `findKey` output lists the total number of matching keys and their key handles.

<table>
<thead>
<tr>
<th>Command: findKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of keys present 10</td>
</tr>
<tr>
<td>number of keys matched from start index 0::9</td>
</tr>
<tr>
<td>6, 7, 8, 9, 10, 11, 262156, 262157, 262158, 262159</td>
</tr>
<tr>
<td>Cluster Error Status</td>
</tr>
<tr>
<td>Node id 1 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Node id 2 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

Related Topics
- `findSingleKey (p. 145)`
- `getKeyInfo (p. 169)`
- `getAttribute (p. 164)`
- `findAllKeys (p. 93)` in cloudhsm_mgmt_util
findSingleKey

The **findSingleKey** command in the key_mgmt_util tool verifies that a key exists on all HSMs in the cluster.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
findSingleKey -h
findSingleKey -k <key-handle>
```

**Example**

This command verifies that key 252136 exists on all three HSMs in the cluster.

```
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
```

**Parameters**

- **-h**
  - Displays help for the command.
  - Required: Yes

- **-k**
  - Specifies the key handle of one key in the HSM. This parameter is required.
  - To find key handles, use the **findKey** (p. 188) command.
  - Required: Yes

**Related Topics**

- **findKey** (p. 188)
- **getKeyInfo** (p. 188)
- **getAttribute** (p. 141)

**genDSAKeyPair**

The **genDSAKeyPair** command in the key_mgmt_util tool generates a Digital Signing Algorithm (DSA) key pair in your HSMs. You must specify the modulus length; the command generates the modulus value.
You can also assign an ID, share the key with other HSM users, create nonextractable keys, and create keys that expire when the session ends. When the command succeeds, it returns the *key handles* that the HSM assigns to the public and private keys. You can use the key handles to identify the keys to other commands.

Before you run any *key_mgmt_util* command, you must start *key_mgmt_util* (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Tip**
To find the attributes of a key that you have created, such as the type, length, label, and ID, use *getAttribute* (p. 164). To find the keys for a particular user, use *getKeyInfo* (p. 169). To find keys based on their attribute values, use *findKey* (p. 141).

**Syntax**

```bash
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]
```

**Examples**

These examples show how to use *genDSAKeyPair* to create a DSA key pair.

**Example : Create a DSA Key Pair**

This command creates a DSA key pair with a DSA label. The output shows that the key handle of the public key is 19 and the handle of the private key is 21.

**Command:**

```bash
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 : Create a Session-Only DSA Key Pair**

This command creates a DSA key pair that is valid only in the current session. The command assigns a unique ID of *DSA_temp_pair* in addition to the required (nonunique) label. You might want to create a key pair like this to sign and verify a session-only token. The output shows that the key handle of the public key is 12 and the handle of the private key is 14.

**Command:**

```bash
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
```
To confirm that the key pair exists only in the session, use the \texttt{-sess} parameter of \texttt{findKey (p. 141)} with a value of 1 (true).

\begin{verbatim}
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
\end{verbatim}

\textbf{Example : Create a Shared, Nonextractable DSA Key Pair}

This command creates a DSA key pair. The private key is shared with three other users, and it cannot be exported from the HSM. Public keys can be used by any user and can always be extracted.

\begin{verbatim}
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
\end{verbatim}

\textbf{Example : Create a Quorum-Controlled Key Pair}

This command creates a DSA key pair with the label \texttt{DSA-mV2}. The command uses the \texttt{-u} parameter to share the private key with user 4 and 6. It uses the \texttt{-m_value} parameter to require a quorum of at least two approvals for any cryptographic operations that use the private key. The command also uses the \texttt{-attest} parameter to verify the integrity of the firmware on which the key pair is generated.

The output shows that the command generates a public key with key handle 12 and a private key with key handle 17, and that the attestation check on the cluster firmware passed.

\begin{verbatim}
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
\end{verbatim}

This command uses \texttt{getKeyInfo (p. 169)} on the private key (key handle 17). The output confirms that the key is owned by the current user (user 3) and that it is shared with users 4 and 6 (and no others). The output also shows that quorum authentication is enabled and the quorum size is two.
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

Parameters

-h
Displays help for the command.
Required: Yes

-m
Specifies the length of the modulus in bits. The only valid value is 2048.
Required: Yes

-l
Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair.
You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.
Required: Yes

-id
Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.
Default: No ID value.
Required: No

-min_srv
Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.
Default: 1
Required: No

-m_value
Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from 0 to 8.
This parameter establishes a quorum authentication requirement for the private key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the m_value of a key, use `getKeyInfo (p. 169)`.

This parameter is valid only when the `-u` parameter in the command shares the key pair with enough users to satisfy the m_value requirement.

Default: 0

Required: No

- **-nex**

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM (p. 58). Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.

Required: No

- **-sess**

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use `setAttribute (p. 190)`.

Default: The key is persistent.

Required: No

- **-timeout**

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the min_srv parameter.

This parameter is valid only when the min_srv parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

- **-u**

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use `listUsers (p. 188)`. To share and unshare existing keys, use `shareKey (p. 116)` in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No
-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related Topics

- genRSAKeyPair (p. 154)
- genSymKey (p. 158)
- genECCKeyPair (p. 150)

**genECCKeyPair**

The `genECCKeyPair` command in the key_mgmt_util tool generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs. When running the `genECCKeyPair` command, you must specify the elliptic curve identifier and a label for the key pair. You can also share the private key with other CU users, create non-extractable keys, quorum-controlled keys, and keys that expire when the session ends. When the command succeeds, it returns the key handles that the HSM assigns to the public and private ECC keys. You can use the key handles to identify the keys to other commands.

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Tip**

To find the attributes of a key that you have created, such as the type, length, label, and ID, use `getAttribute` (p. 164). To find the keys for a particular user, use `getKeyInfo` (p. 169). To find keys based on their attribute values, use `findKey` (p. 141).

**Syntax**

```
genECCKeyPair -h

genECCKeyPair -i <EC curve id>
    -l <label>
    [-id <key ID>]
    [-min_grv <minimum number of servers>]
    [-m_value <0..8>]
    [-nex]
    [-sess]
    [-timeout <number of seconds>]
    [-u <user-ids>]
    [-attest]
```

**Examples**

The following examples show how to use `genECCKeyPair` to create ECC key pairs in your HSMs.

**Example : Create and Examine an ECC Key Pair**

This command uses an NID_sect571r1 elliptic curve and an ecc14 label to create an ECC key pair. The output shows that the key handle of the private key is 262177 and the key handle of the public key is 262179. The label applies to both the public and private keys.
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

After generating the key, you can examine its attributes. Use `getAttribute (p. 164)` to write all of the attributes (represented by the constant 512) of the new ECC private key to the `attr_262177` file.

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

Then use the `cat` command to view the contents of the `attr_262177` attribute file. The output shows the key is an elliptic curve private key that can be used for signing, but not for encrypting, decrypting, wrapping, unwrapping, or verifying. The key is persistent and exportable.

$ `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
044a0f9d01d10f7437d9f9a20995f0cc742552e5ba16d37e9a65a33e20ad3e569e68eb62477a9960a87911e6121d12b698e469
```
Example Using an Invalid EEC Curve

This command attempts to create an ECC key pair by using an NID_X9_62_prime192v1 curve. Because this elliptic curve is not valid for FIPS-mode HSMs, the command fails. The message reports that a server in the cluster is unavailable, but this does not typically indicate a problem with the HSMs in the cluster.

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

Parameters
-h
Displays help for the command.
Required: Yes

-i
Specifies the identifier for the elliptic curve. Enter an identifier.
Valid values:
• 2: NID_X9_62_prime256v1
• 14: NID_secp384r1
• 16: NID_secp256k1
Required: Yes

-l
Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair.
You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.
Required: Yes

-id
Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.
Default: No ID value.
Required: No

-min_srv
Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.
AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1
Required: No

**-m_value**

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from 0 to 8.

This parameter establishes a quorum authentication requirement for the private key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the m_value of a key, use `getKeyInfo (p. 169)`.

This parameter is valid only when the -u parameter in the command shares the key pair with enough users to satisfy the m_value requirement.

Default: 0
Required: No

**-nex**

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM (p. 58). Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.
Required: No

**-sess**

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use `setAttribute (p. 190)`.

Default: The key is persistent.
Required: No

**-timeout**

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.
Required: No
-u

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as -u 5, 6. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers (p. 188). To share and unshare existing keys, use shareKey (p. 116) in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related Topics

• genSymKey (p. 158)
• genRSAKeyPair (p. 154)
• genDSAKeyPair (p. 145)

genPBEKey

The genPBEKey command in the key_mgmt_util tool generates a Triple DES (3DES) symmetric key based on a password. This command is not supported on the FIPS-validated HSMs that AWS CloudHSM provides.

To create symmetric keys, use genSymKey (p. 158). To create asymmetric key pairs, use genRSAKeyPair (p. 154), genDSAKeyPair (p. 145), or genECCKeyPair (p. 150).

genRSAKeyPair

The genRSAKeyPair command in the key_mgmt_util tool generates an RSA asymmetric key pair. You specify the key type, modulus length, and a public exponent. The command generates a modulus of the specified length and creates the key pair. You can assign an ID, share the key with other HSM users, create nonextractable keys and keys that expire when the session ends. When the command succeeds, it returns a key handle that the HSM assigns to the key. You can use the key handle to identify the key to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Tip

To find the attributes of a key that you have created, such as the type, length, label, and ID, use getAttribute (p. 164). To find the keys for a particular user, use getKeyInfo (p. 169). To find keys based on their attribute values, use findKey (p. 141).

Syntax

| genRSAKeyPair -h |
Examples

These examples show how to use genRSAKeyPair to create asymmetric key pairs in your HSMs.

Example: Create and Examine an RSA Key Pair

This command creates an RSA key pair with a 2048-bit modulus and an exponent of 65541. The output shows that the public key handle is 262159 and the private key handle is 262160.

Example: Create and Examine an RSA Key Pair

```bash
Command: genRSAKeyPair -m 2048 -e 65541 -l rsa_test
```

Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS
Cfm3GenerateKeyPair: public key handle: 262159 private key handle: 262160
Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

The next command uses getAttribute (p. 164) to get the attributes of the public key that we just created. It writes the output to the attr_262159 file. It is followed by a cat command that gets the content of the attribute file. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

The resulting hexadecimal values confirm that it is a public key (OBJ_ATTR_CLASS 0x02) with a type of RSA (OBJ_ATTR_KEY_TYPE 0x00). You can use this public key to encrypt (OBJ_ATTR_ENCRYPT 0x01), but not to decrypt (OBJ_ATTR_DECRYPT 0x00) or wrap (OBJ_ATTR_WRAP 0x00). The results also include the key length (512, 0x200), the modulus, the modulus length (2048, 0x800), and the public exponent (65541, 0x10005).

Example: Create and Examine an RSA Key Pair

```bash
Command: getAttribute -o 262159 -a 512 -out attr_262159
```

got all attributes of size 731 attr cnt 20
Attributes dumped into attr_262159 file

```bash
$ cat attr_262159
OBJ_ATTR_CLASS
0x02
OBJ_ATTR_KEY_TYPE
0x00
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x00
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
```

155
Example : Generate a Shared RSA Key Pair

This command generates an RSA key pair and shares the private key with user 4, another CU on the HSM. The command uses the m_value parameter to require at least two approvals before the private key in the pair can be used in a cryptographic operation. When you use the m_value parameter, you must also use -u in the command and the m_value cannot exceed the total number of users (number of values in -u + owner).

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
```

Parameters

-h

Displays help for the command.

Required: Yes

-m

Specifies the length of the modulus in bits. The minimum value is 2048.
Required: Yes

-e

Specifies the public exponent. The value must be an odd number greater than or equal to 65537.

Required: Yes

-l

Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id

Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from 0 to 8.

This parameter establishes a quorum authentication requirement for the private key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the m_value of a key, use getKeyInfo (p. 169).

This parameter is valid only when the -u parameter in the command shares the key pair with enough users to satisfy the m_value requirement.

Default: 0

Required: No

-nex

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM (p. 58). Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.
-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute (p. 190).

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the min_srv parameter.

This parameter is valid only when the min_srv parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as -u 5,6. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers (p. 188). To share and unshare existing keys, use shareKey (p. 116) in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related Topics

- genSymKey (p. 158)
- genDSAKeyPair (p. 145)
- genECCKeyPair (p. 150)

**genSymKey**

The genSymKey command in the key_mgmt_util tool generates a symmetric key in your HSMs. You can specify the key type and size, assign an ID and label, and share the key with other HSM users. You
can also create nonextractable keys and keys that expire when the session ends. When the command
succeeds, it returns a key handle that the HSM assigns to the key. You can use the key handle to identify
the key to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log
in (p. 124) to the HSM as a crypto user (CU).

**Tip**
To find the attributes of a key that you have created, such as the type, length, label, and ID, use
getAttribute (p. 164). To find the keys for a particular user, use getKeyInfo (p. 169). To find
keys based on their attribute values, use findKey (p. 141).

**Syntax**

```bash
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]
```

**Examples**

These examples show how to use `genSymKey` to create symmetric keys in your HSMs.

**Example : Generate an AES Key**

This command creates a 256-bit AES key with an `aes256` label. The output shows that the key handle of
the new key is 6.

**Command:**

```bash
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 : Create a Session Key**

This command creates a nonextractable 192-bit AES key that is valid only in the current session. You
might want to create a key like this to wrap (and then immediately unwrap) a key that is being exported.

**Command:**

```bash
genSymKey -t 31 -s 24 -l tmpAES -id wrap01 -nex -sess
```

**Example : Return Quickly**

This command creates a generic 512-byte key with a label of `IT_test_key`. The command does not
wait for the key to be synchronized to all HSMs in the cluster. Instead, it returns as soon as the key is
created on any one HSM (`-min_srv 1`) or in 1 second (`-timeout 1`), whichever is shorter. If the key
is not synchronized to the specified minimum number of HSMs before the timeout expires, it is not
generated. You might want to use a command like this in a script that creates numerous keys, like the for loop in the following example.

```
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 Cfm3Util singlecmd loginHSM -u CU -s example_user -p example_pwd genSymKey -l aes -t 31 -s 32 -min_srv 1 -timeout 1;
done;

**Example : Create a Quorum Authorized Generic Key**

This command creates a 2048-bit generic secret key with the label `generic-mV2`. The command uses the `-u` parameter to share the key with another CU, user 6. It uses the `-m_value` parameter to require a quorum of at least two approvals for any cryptographic operations that use the key. The command also uses the `-attest` parameter to verify the integrity of the firmware on which the key is generated.

The output shows that the command generated a key with key handle 9 and that the attestation check on the cluster firmware passed.

```
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 : Create and Examine a Key**

This command creates a Triple DES key with a `3DES_shared` label and an ID of `IT-02`. The key can be used by the current user, and users 4 and 5. The command fails if the ID is not unique in the cluster or if the current user is user 4 or 5.

The output shows that the new key has key handle 7.

```
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
```

To verify that the new 3DES key is owned by the current user and shared with users 4 and 5, use `getKeyInfo (p. 169)`. The command uses the handle that was assigned to the new key (Key Handle: 7).

The output confirms that the key is owned by user 3 and shared with users 4 and 5.

```
Command: getKeyInfo -k 7
```

```
Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS
```
Owned by user 3
also, shared to following 2 user(s):
  4, 5

To confirm the other properties of the key, use `getAttribute` (p. 164). The first command uses `getAttribute` to get all attributes (-a 512) of key handle 7 (-o 7). It writes them to the `attr_7` file. The second command uses `cat` to get the contents of the `attr_7` file.

This command confirms that key 7 is a 192-bit (OBJ_ATTR_VALUE_LEN 0x00000018 or 24-byte) 3DES (OBJ_ATTR_KEY_TYPE 0x15) symmetric key (OBJ_ATTR_CLASS 0x04) with a label of 3DES_shared (OBJ_ATTR_LABEL 3DES_shared) and an ID of IT_02 (OBJ_ATTR_ID IT-02). The key is persistent (OBJ_ATTR_TOKEN 0x01) and extractable (OBJ_ATTR_EXTRACTABLE 0x01) and can be used for encryption, decryption, and wrapping.

For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

```
Command:   getAttribute -o 7 -a 512 -attr_7

got all attributes of size 444 attr cnt 17
Attributes dumped into attr_7 file

$ 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
```
Parameters

-h
Displays help for the command.
Required: Yes

-t
Specifies the type of the symmetric key. Enter the constant that represents the key type. For example, to create an AES key, type -t 31.
Valid values:
- 16: GENERIC_SECRET. A generic secret key is a byte array that does not conform to any particular standard, such as the requirements for an AES key.
- 18: RC4. RC4 keys are not valid on FIPS-mode HSMs
- 21: Triple DES (3DES).
- 31: AES
Required: Yes

-s
Specifies the key size in bytes. For example, to create a 192-bit key, type 24.
Valid values for each key type:
- AES: 16 (128 bits), 24 (192 bits), 32 (256 bits)
- 3DES: 24 (192 bits)
- Generic Secret: <3584 (28672 bits)
Required: Yes

-l
 Specifies a user-defined label for the key. Type a string.
You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.
Required: Yes

-attest
Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.
Default: No attestation check.
Required: No

-id
Specifies a user-defined identifier for the key. Type a string that is unique in the cluster. The default is an empty string.
Default: No ID value.
Required: No

-min_srv
Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.
AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1
Required: No

-m_value
Specifies the number of users who must approve any cryptographic operation that uses the key. Type a value from 0 to 8.

This parameter establishes a quorum authentication requirement for the key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the key, and operations that share or unshare the key.

To find the `m_value` of a key, use `getKeyInfo (p. 169).` This parameter is valid only when the `-u` parameter in the command shares the key with enough users to satisfy the `m_value` requirement.

Default: 0
Required: No

-nex
Makes the key nonextractable. The key that is generated cannot be exported from the HSM (p. 58).

Default: The key is extractable.
Required: No

-sess
Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use `setAttribute (p. 190).`

Default: The key is persistent.
Required: No

-timeout
Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u
Shares the key with the specified users. This parameter gives other HSM crypto users (CUs) permission to use this key in cryptographic operations.
Type a comma-separated list of HSM user IDs, such as -u 5, 6. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers (p. 188). To share and unshare existing keys, use shareKey (p. 116) in the cloudhsm_mgmt_util.

Default: Only the current user can use the key.
Required: No

Related Topics

- exSymKey (p. 136)
- genRSAKeyPair (p. 154)
- genDSAKeyPair (p. 145)
- genECCKeyPair (p. 150)

getAttribute

The getAttribute command in key_mgmt_util writes one or all of the attribute values for an AWS CloudHSM key to a file. If the attribute you specify does not exist for the key type, such as the modulus of an AES key, getAttribute returns an error.

Key attributes are properties of a key. They include characteristics, like the key type, class, label, and ID, and values that represent actions that you can perform with the key, like encrypt, decrypt, wrap, sign, and verify.

You can use getAttribute only on keys that you own and key that are shared with you. You can run this command or the getAttribute (p. 96) command in cloudhsm_mgmt_util, which gets one attribute value of a key from all HSMs in a cluster, and writes it to stdout or to a file.

To get a list of attributes and the constants that represent them, use the listAttributes (p. 187) command. To change the attribute values of existing keys, use setAttribute (p. 190) in key_mgmt_util and setAttribute (p. 112) in cloudhsm_mgmt_util. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

getAttribute -h
getAttribute -o <key handle> -a <attribute constant> -out <file>

Examples

These examples show how to use getAttribute to get the attributes of keys in your HSMs.

Example : Get the Key Type

This example gets the type of the key, such an AES, 3DES, or generic key, or an RSA or elliptic curve key pair.

The first command runs listAttributes (p. 187), which gets the key attributes and the constants that represent them. The output shows that the constant for key type is 256. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).
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
```

The second command runs `getAttribute`. It requests the key type (attribute 256) for key handle 524296 and writes it to the `attribute.txt` file.

Command: `getAttribute -o 524296 -a 256 -out attribute.txt`

Attributes dumped into attribute.txt file

The final command gets the content of the key file. The output reveals that the key type is 0x15 or 21, which is a Triple DES (3DES) key. For definitions of the class and type values, see the **Key Attribute Reference** (p. 204).

```
$ cat attribute.txt
OBJ_ATTR_KEY_TYPE
0x00000015
```

**Example: Get All Attributes of a Key**

This command gets all attributes of the key with key handle 6 and writes them to the `attr_6` file. It uses an attribute value of 512, which represents all attributes.

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>
```

This command shows the content of a sample attribute file with all attribute values. Among the values, it reports that key is a 256-bit AES key with an ID of test_01 and a label of aes256. The key is extractable and persistent, that is, not a session-only key. For help interpreting the key attributes, see the **Key Attribute Reference** (p. 204).

```
$ cat attribute.txt
```

165
Parameters

-h
Displays help for the command.
Required: Yes

-o
Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use findKey (p. 141).

Also, you must own the specified key or it must be shared with you. To find the users of a key, use getKeyInfo (p. 169).

Required: Yes

-a
Identifies the attribute. Enter a constant that represents an attribute, or 512, which represents all attributes. For example, to get the key type, type 256, which is the constant for the OBJ_ATTR_KEY_TYPE attribute.

To list the attributes and their constants, use listAttributes (p. 187). For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

Required: Yes
-out

Writes the output to the specified file. Type a file path. You cannot write the output to stdout.

If the specified file exists, getAttribute overwrites the file without warning.

Required: Yes

Related Topics

- getAttribute (p. 96) in cloudhsm_mgmt_util
- listAttributes (p. 187)
- setAttribute (p. 190)
- findKey (p. 141)
- Key Attribute Reference (p. 204)

getCaviumPrivKey

The getCaviumPrivKey command in key_mgmt_util exports a private key from an HSM in fake PEM format. The fake PEM file, which does not contain the actual private key material but instead references the private key in the HSM, can then be used to establish SSL/TLS offloading from your web server to AWS CloudHSM. For more information, see SSL/TLS Offload on Linux (p. 271).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and login (p. 124) to the HSM as a crypto user (CU).

Syntax

```
getCaviumPrivKey -h
exportPrivateKey -k <private-key-handle> -out <fake-PEM-file>
```

Examples

This example shows how to use getCaviumPrivKey to export a private key in fake PEM format.

Example: Export a Fake PEM File

This command creates and exports a fake PEM version of a private key with handle 15 and saves it to a file called cavKey.pem. When the command succeeds, exportPrivateKey returns a success message.

```
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
```

Parameters

This command takes the following parameters.

- h

    Displays command line help for the command.
Required: Yes
-k
  Specifies the key handle of the private key to be exported in fake PEM format.
  Required: Yes
-out
  Specifies the name of the file to which the fake PEM key will be written.
  Required: Yes

Related Topics
- importPrivateKey (p. 173)
- SSL/TLS Offload on Linux (p. 271)

getCert

The **getCert** command in key_mgmt_util retrieves an HSM's partition certificates and saves them to a file. When you run the command, you designate the type of certificate to retrieve. To do that, you use one of the corresponding integers as described in the Parameters (p. 168) section that follows. To learn about the role of each of these certificates, see **Verify HSM Identity (p. 23)**.

Before you run any key_mgmt_util command, you must **start key_mgmt_util (p. 123)** and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
getCert -h
getCert -f <file-name>
  -t <certificate-type>
```

**Example**

This example shows how to use **getCert** to retrieve a cluster's customer root certificate and save it as a file.

**Example : Retrieve a Customer Root Certificate**

This command exports a customer root certificate (represented by integer 4) and saves it to a file called **userRoot.crt**. When the command succeeds, **getCert** returns a success message.

```
Command: getCert -f userRoot.crt -s 4
Cfm3GetCert() returned 0 :HSM Return: SUCCESS
```

**Parameters**

This command takes the following parameters.

- **-h**
  Displays command line help for the command.
Required: Yes

-\( \text{-f} \)

Specifies the name of the file to which the retrieved certificate will be saved.

Required: Yes

-\( \text{-s} \)

An integer that specifies the type of partition certificate to retrieve. The integers and their corresponding certificate types are as follows:

- 1 – Manufacturer root certificate
- 2 – Manufacturer hardware certificate
- 4 – Customer root certificate
- 8 – Cluster certificate (signed by customer root certificate)
- 16 – Cluster certificate (chained to the manufacturer root certificate)

Required: Yes

Related Topics

- Verify HSM Identity (p. 23)
- getCert (p. 99) (in cloudhsm_mgmt_util (p. 75))

getKeyInfo

The `getKeyInfo` command in the key_mgmt_util returns the HSM user IDs of users who can use the key, including the owner and crypto users (CU) with whom the key is shared. When quorum authentication is enabled on a key, `getKeyInfo` also returns the number of users who must approve cryptographic operations that use the key. You can run `getKeyInfo` only on keys that you own and keys that are shared with you.

When you run `getKeyInfo` on public keys, `getKeyInfo` returns only the key owner, even though all users of the HSM can use the public key. To find the HSM user IDs of users in your HSMs, use `listUsers` (p. 188). To find the keys for a particular user, use `findKey` (p. 141) –u.

You own the keys that you create. You can share a key with other users when you create it. Then, to share or unshare an existing key, use `shareKey` (p. 116) in cloudhsm_mgmt_util.

Before you run any key_mgmt_util command, you must `start key_mgmt_util` (p. 123) and `log in` (p. 124) to the HSM as a crypto user (CU).

Syntax

```
getKeyInfo -h
getKeyInfo -k <key-handle>
```

Examples

These examples show how to use `getKeyInfo` to get information about the users of a key.

**Example : Get the Users for a Symmetric Key**

This command gets the users who can use the AES (symmetric) key with key handle 9. The output shows that user 3 owns the key and has shared it with user 4.
Example: Get the Users for an Asymmetric Key Pair

These commands use `getKeyInfo` to get the users who can use the keys in an RSA (asymmetric) key pair. The public key has key handle 21. The private key has key handle 20.

When you run `getKeyInfo` on the private key (20), it returns the key owner (3) and crypto users (CUs) 4 and 5, with whom the key is shared.

Command: `getKeyInfo -k 20`

Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

Owned by user 3

also, shared to following 2 user(s):

4

5

When you run `getKeyInfo` on the public key (21), it returns only the key owner (3).

Command: `getKeyInfo -k 21`

Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

Owned by user 3

To confirm that user 4 can use the public key (and all public keys on the HSM), use the `-u` parameter of `findKey` (p. 141).

The output shows that user 4 can use both the public (21) and private (20) key in the key pair. User 4 can also use all other public keys and any private keys that they have created or that have been shared with them.

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: Get the Quorum Authentication Value (m_value) for a Key

This example shows how to get the m_value for a key, that is, the number of users in the quorum who must approve any cryptographic operations that use the key.
When quorum authentication is enabled on a key, a quorum of users must approve any cryptographic operations that use the key. To enable quorum authentication and set the quorum size, use the \texttt{-m\_value} parameter when you create the key.

This command uses \texttt{genRSAKeyPair (p. 154)} to create an RSA key pair that is shared with user 4. It uses the \texttt{m\_value} parameter to enable quorum authentication on the private key in the pair and set the quorum size to two users. The number of users must be large enough to provide the required approvals.

The output shows that the command created public key 27 and private key 28.

\begin{verbatim}
Command: genRSAKeyPair \texttt{-m 2048 -e 195193 \text{-l rsa_mofn \text{-id rsa_mv2 \text{-u 4 \text{-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
\end{verbatim}

This command uses \texttt{getKeyInfo} to get information about the users of the private key. The output shows that the key is owned by user 3 and shared with user 4. It also shows that a quorum of two users must approve every cryptographic operation that uses the key.

\begin{verbatim}
Command: getKeyInfo \texttt{-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
\end{verbatim}

**Parameters**

-\texttt{-h}
  Displays command line help for the command.
  Required: Yes

-\texttt{-k}
  Specifies the key handle of one key in the HSM. Enter the key handle of a key that you own or share. This parameter is required.
  To find key handles, use the \texttt{findKey (p. 188)} command.
  Required: Yes

**Related Topics**

- \texttt{getKeyInfo (p. 102)} in cloudhsm\_mgmt\_util
- \texttt{listUsers (p. 188)}
- \texttt{findKey (p. 141)}
- \texttt{findAllKeys (p. 93)} in cloudhsm\_mgmt\_util
help

The help command in key_mgmt_util displays information about all available key_mgmt_util commands.

Before you run help, you must start key_mgmt_util (p. 123).

Syntax

```plaintext
help
```

Example

This example shows the output of the help command.

Example

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exit</td>
<td>Exits this application</td>
</tr>
<tr>
<td>help</td>
<td>Displays this information</td>
</tr>
</tbody>
</table>

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
Parameters

There are no parameters for this command.

Related Topics

- loginHSM and logoutHSM (p. 189)

importPrivateKey

The importPrivateKey command in key_mgmt_util imports an asymmetric private key into an HSM. You can use it to import private keys that were generated outside of the HSM. You can also use this command to import keys that were exported from an HSM, such as those exported by the exportPrivateKey (p. 133) command.

During the import process, importPrivateKey uses an AES key (the wrapping key) that you select to wrap (encrypt) the private key. By wrapping the private key, it remains confidential during transit. For more information, see wrapKey (p. 201).

Note

This command does not offer the option to mark the imported key as non-exportable.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and login (p. 124) to the HSM as a crypto user (CU).

Syntax

```bash
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]

Examples

This example shows how to use importPrivateKey to import a private key into an HSM.

Example: Import a Private Key

This command imports the private key from a file named rsa2048.key with the label rsa2048-imported and a wrapping key with handle 524299. When the command succeeds, importPrivateKey returns a key handle for the imported key and a success message.

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

Parameters

This command takes the following parameters.

\-h

Displays command line help for the command.

Required: Yes

\-l

Specifies the user-defined private key label.

Required: Yes

\-f

Specifies the file name of the key to import.

Required: Yes
-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey (p. 141) command.

To determine whether a key can be used as a wrapping key, use getAttribute (p. 164) to get the value of the OBJ_ATTR_WRAP attribute (262). To create a wrapping key, use genSymKey (p. 158) to create an AES key (type 31).

If you use the -wk parameter to specify an external unwrapping key, the -w wrapping key is used to wrap, but not unwrap, the key during import.

Required: Yes

-sess

Specifies the imported key as a session key.

Default: The imported key is held as a persistent (token) key in the cluster.

Required: No

-id

Specifies the ID of the key to be imported.

Default: No ID value.

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the imported key. Enter a value from 0 to 8.

This parameter is valid only when the -u parameter in the command shares the key with enough users to satisfy the m_value requirement.

Default: 0

Required: No

-min_srv

Specifies the minimum number of HSMs on which the imported key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-timeout

Specifies the number of seconds to wait for the key to sync across HSMs when the min_srv parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No
-u

Specifies the list of users with whom to share the imported private key. This parameter gives other HSM crypto users (CUs) permission to use the imported key in cryptographic operations.

Enter a comma-separated list of HSM user IDs, such as -u 5, 6. Do not include the HSM user ID of the current user. To find the HSM user IDs of CUs on the HSM, use listUsers (p. 188).

Default: Only the current user can use the imported key.

Required: No

-wk

Specifies the key to be used to wrap the key that is being imported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, importPrivateKey uses the key in the -wk file to wrap the key being imported. It also uses the key specified by the -w parameter to unwrap it.

Default: Use the wrapping key specified in the -w parameter to both wrap and unwrap.

Required: No

-attest

Performs an attestation check on the firmware response to ensure that the firmware on which the cluster runs has not been compromised.

Required: No

Related Topics

- wrapKey (p. 201)
- unWrapKey (p. 195)
- genSymKey (p. 158)
- exportPrivateKey (p. 133)

importPubKey

The importPubKey command in key_mgmt_util imports a PEM format public key into an HSM. You can use it to import public keys that were generated outside of the HSM. You can also use the command to import keys that were exported from an HSM, such as those exported by the exportPubKey (p. 134) command.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

```
importPubKey -h
importPubKey -l <label> -f <key-file> [-sess] [-id <key-id>] [min_srv <minimum-number-of-servers>] [-timeout <number-of-seconds>]
```
Examples

This example shows how to use `importPubKey` to import a public key into an HSM.

Example: Import a Public Key

This command imports a public key from a file named `public.pem` with the label `importedPublicKey`. When the command succeeds, `importPubKey` returns a key handle for the imported key and a success message.

```
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

Parameters

This command takes the following parameters.

-h
   Displays command line help for the command.
   Required: Yes

-l
   Specifies the user-defined public key label.
   Required: Yes

-f
   Specifies the file name of the key to import.
   Required: Yes

-sess
   Designates the imported key as a session key.
   Default: The imported key is held as a persistent (token) key in the cluster.
   Required: No

-id
   Specifies the ID of the key to be imported.
   Default: No ID value.
   Required: No

-min_srv
   Specifies the minimum number of HSMs to which the imported key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.
AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

- **-timeout**

Specifies the number of seconds to wait for the key to sync across HSMs when the `min-serv` parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No

**Related Topics**

- exportPubKey (p. 134)
- Generate Keys (p. 55)

**imSymKey**

The `imSymKey` command in the `key_mgmt_util` tool imports a plaintext copy of a symmetric key from a file into the HSM. You can use it to import keys that you generate by any method outside of the HSM and keys that were exported from an HSM, such as the keys that the `exSymKey` (p. 136), command writes to a file.

During the import process, `imSymKey` uses an AES key that you select (the *wrapping key*) to wrap (encrypt) and then unwrap (decrypt) the key to be imported. However, `imSymKey` works only on files that contain plaintext keys. To export and import encrypted keys, use the `wrapKey` (p. 201) and `unwrapKey` (p. 195) commands.

Also, the `imSymKey` command exports only symmetric keys. To import public keys, use `importPubKey` (p. 176). To import private keys, use `importPrivateKey` (p. 173) or `wrapKey` (p. 201).

Imported keys work very much like keys generated in the HSM. However, the value of the `OBJ_ATTR_LOCAL` attribute (p. 204) is zero, which indicates that it was not generated locally. You can use the following command to share a symmetric key as you import it. You can use the `shareKey` command in `cloudhsm_mgmt_util` (p. 75) to share the key after it is imported.

```
imSymKey -l aesShared -t 31 -f kms.key -w 3296 -u 5
```

After you import a key, be sure to mark or delete the key file. This command does not prevent you from importing the same key material multiple times. The result, multiple keys with distinct key handles and the same key material, make it difficult to track use of the key material and prevent it from exceeding its cryptographic limits.

Before you run any `key_mgmt_util` command, you must `start key_mgmt_util` (p. 123) and `log in` (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
imSymKey -h
imSymKey -f <key-file>
    -w <wrapping-key-handle>
```
Examples

These examples show how to use `imSymKey` to import symmetric keys into your HSMs.

Example: Import an AES Symmetric Key

This example uses `imSymKey` to import an AES symmetric key into the HSMs.

The first command uses OpenSSL to generate a random 256-bit AES symmetric key. It saves the key in the `aes256.key` file.

```
$ openssl rand -out aes256-forImport.key 32
```

The second command uses `imSymKey` to import the AES key from the `aes256.key` file into the HSMs. It uses key 20, an AES key in the HSM, as the wrapping key and it specifies a label of `imported`. Unlike the ID, the label does not need to be unique in the cluster. The value of the `-t` (type) parameter is 31, which represents AES.

The output shows that the key in the file was wrapped and unwrapped, then imported into the HSM, where it was assigned the key handle 262180.

```
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
```

The next command uses `getAttribute` to get the OBJ_ATTR_LOCAL attribute (attribute 355) of the newly imported key and writes it to the `attr_262180` file.

```
Command: getAttribute -o 262180 -a 355 -out attributes/attr_262180_imported

Attributes dumped into attributes/attr_262180_imported file
Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

When you examine the attribute file, you can see that the value of the OBJ_ATTR_LOCAL attribute is zero, which indicates that the key material was not generated in the HSM.

```
$ cat attributes/attr_262180_local
```

```
179
```
Example: Move a Symmetric Key Between Clusters

This example shows how to use exSymKey (p. 136) and imSymKey to move a plaintext AES key between clusters. You might use a process like this one to create an AES wrapping that exists on the HSMs both clusters. Once the shared wrapping key is in place, you can use wrapKey (p. 201) and unWrapKey (p. 195) to move encrypted keys between the clusters.

The CU user who performs this operation must have permission to log in to the HSMs on both clusters.

The first command uses exSymKey (p. 136) to export key 14, a 32-bit AES key, from the cluster 1 into the aes.key file. It uses key 6, an AES key on the HSMs in cluster 1, as the wrapping key.

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"

The user then logs into key_mgmt_util in cluster 2 and runs an imSymKey command to import the key in the aes.key file into the HSMs in cluster 2. This command uses key 262152, an AES key on the HSMs in cluster 2, as the wrapping key.

Because the wrapping keys that exSymKey (p. 136) and imSymKey use wrap and immediately unwrap the target keys, the wrapping keys on the different clusters need not be the same.

The output shows that the key was successfully imported into cluster 2 and assigned a key handle of 21.

Command:  
```
imSymKey -f aes.key -w 262152 -t 31 -l xcluster
```

Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS
Cfm3UnDispKey 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

To prove that key 14 of cluster 1 and key 21 in cluster 2 have the same key material, get the key check value (KCV) of each key. If the KCV values are the same, the key material is the same.

The following command uses getAttribute (p. 164) in cluster 1 to write the value of the KCV attribute (attribute 371) of key 14 to the attr_14_kcv file. Then, it uses a cat command to get the content of the attr_14_kcv file.

Command:  
```
getAttribute -o 14 -a 371 -out attr_14_kcv
```

Attributes dumped into attr_14_kcv file

$ cat attr_14_kcv
OBJ_ATTR_KCV 180
This similar command uses `getAttribute (p. 164)` in cluster 2 to write the value of the KCV attribute (attribute 371) of key 21 to the `attr_21_kcv` file. Then, it uses a `cat` command to get the content of the `attr_21_kcv` file.

Command:  
```
getAttribute -o 21 -a 371 -out attr_21_kcv
```

```bash
$ cat attr_21_kcv
OBJ_ATTR_KCV
0xc33cbd
```

The output shows that the KCV values of the two keys are the same, which proves that the key material is the same.

Because the same key material exists in the HSMs of both clusters, you can now share encrypted keys between the clusters without ever exposing the plaintext key. For example, you can use the `wrapKey` command with wrapping key 14 to export an encrypted key from cluster 1, and then use `unWrapKey` with wrapping key 21 to import the encrypted key into cluster 2.

### Example: Import a Session Key

This command uses the `--sess` parameters of `imSymKey` to import a 192-bit Triple DES key that is valid only in the current session.

The command uses the `--f` parameter to specify the file that contains the key to import, the `--t` parameter to specify the key type, and the `--w` parameter to specify the wrapping key. It uses the `--l` parameter to specify a label that categorizes the key and the `--id` parameter to create a friendly, but unique, identifier for the key. It also uses the `--attest` parameter to verify the firmware that is importing the key.

The output shows that the key was successfully wrapped and unwrapped, imported into the HSM, and assigned the key handle 37. Also, the attestation check passed, which indicates that the firmware has not been tampered.

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
```

Next, you can use the `getAttribute (p. 164)` or `findKey (p. 141)` commands to verify the attributes of the newly imported key. The following command uses `findKey` to verify that key 37 has the type, label, and ID specified by the command, and that it is a session key. As shown on line 5 of the output, `findKey` reports that the only key that matches all of the attributes is key 37.

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

Parameters

-attest
Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.
Default: No attestation check.
Required: No

-f
Specifies the file that contains that key to import.
The file must contain a plaintext copy of an AES or Triple DES key of the specified length. RC4 and DES keys are not valid on FIPS-mode HSMs.
- AES: 16, 24 or 32 bytes
- Triple DES (3DES): 24 bytes
Required: Yes

-h
Displays help for the command.
Required: Yes

-id
Specifies a user-defined identifier for the key. Type a string that is unique in the cluster. The default is an empty string.
Default: No ID value.
Required: No

-l
Specifies a user-defined label for the key. Type a string.
You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.
Required: Yes

-min_srv
Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.
AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.
Default: 1
**-sess**

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute (p. 190).

Default: The key is persistent.

**-timeout**

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the min_srv parameter.

This parameter is valid only when the min_srv parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

**-t**

Specifies the type of the symmetric key. Enter the constant that represents the key type. For example, to create an AES key, enter `-t 31`.

Valid values:
- 21: Triple DES (3DES).
- 31: AES

**-u**

Shares the key you are importing with specified users. This parameter gives other HSM crypto users (CUs) permission to use this key in cryptographic operations.

Type one ID or a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find the an ID, you can use the listUsers command in the cloudhsm_mgmt_util command line tool or the listUsers command in the key_mgmt_util command line tool.

**-w**

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey (p. 141) command.

A *wrapping key* is a key in the HSM that is used to encrypt ("wrap") and then decrypt ("unwrap") the key during the import process. Only AES keys can be used as wrapping keys.

You can use any AES key (of any size) as a wrapping key. Because the wrapping key wraps, and then immediately unwraps, the target key, you can use as session-only AES key as a wrapping key. To determine whether a key can be used as a wrapping key, use getAttribute (p. 164) to get the value of the OBJ_ATTR_WRAP attribute (262). To create a wrapping key, use genSymKey (p. 158) to create an AES key (type 31).
If you use the `-wk` parameter to specify an external wrapping key, the `-w` wrapping key is used to unwrap, but not to wrap, the key that is being imported.

**Note**

Key 4 is an unsupported internal key. We recommend that you use an AES key that you create and manage as the wrapping key.

**Required:** Yes

**-wk**

Use the AES key in the specified file to wrap the key that is being imported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, `imSymKey` uses the key in the `-wk` file to wrap the key being imported and it uses the key in the HSM that is specified by the `-w` parameter to unwrap it. The `-w` and `-wk` parameter values must resolve to the same plaintext key.

**Default:** Use the wrapping key on the HSM to unwrap.

**Required:** No

**Related Topics**

- `genSymKey` (p. 158)
- `exSymKey` (p. 136)
- `wrapKey` (p. 201)
- `unWrapKey` (p. 195)
- `exportPrivateKey` (p. 133)
- `exportPubKey` (p. 134)

**insertMaskedObject**

The `insertMaskedObject` command in key_mgmt_util inserts a masked object from a file into a designated HSM. Masked objects are cloned objects that are extracted from an HSM by using the `extractMaskedObject` command. They can only be used after inserting them back into the original cluster. You can only insert a masked object into the same cluster from which it was generated, or a clone of that cluster. This includes any cloned versions of the original cluster generated by copying a backup across regions and using that backup to create a new cluster.

Masked objects are an efficient way to offload and synchronize keys, including nonextractable keys (that is, keys that have `OBJ_ATTR_EXTRACTABLE` value of 0). This way, keys can be securely synced across related clusters in different regions without the need to update the AWS CloudHSM configure file.

Before you run any key_mgmt_util command, you must start `key_mgmt_util` and `log in` to the HSM as a crypto user (CU).

**Syntax**

```
insertMaskedObject -h
insertMaskedObject -f <filename>
    [-min_srv <minimum-number-of-servers>]
    [-timeout <number-of-seconds>]
```
Examples

This example shows how to use `insertMaskedObject` to insert a masked object file into an HSM.

Example: Insert a Masked Object

This command inserts a masked object into an HSM from a file named `maskedObj`. When the command succeeds, `insertMaskedObject` returns a key handle for the key decrypted from the masked object, and a success message.

```
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
```

Parameters

This command takes the following parameters.

- `-h`
  Displays command line help for the command.
  Required: Yes

- `-f`
  Specifies the file name of the masked object to insert.
  Required: Yes

- `-min_srv`
  Specifies the minimum number of servers on which the inserted masked object is synchronized before the value of the `-timeout` parameter expires. If the object is not synchronized to the specified number of servers in the time allotted, it is not inserted.
  Default: 1
  Required: No

- `-timeout`
  Specifies the number of seconds to wait for the key to sync across servers when the `-min_srv` parameter is included. If no number is specified, the polling continues forever.
  Default: No limit
  Required: No

Related Topics

- `extractMaskedObject (p. 140)`
- `syncKey (p. 118)`
- `Copying a Backup Across Regions (p. 45)`
IsValidKeyHandlefile

The IsValidKeyHandlefile command in key_mgmt_util is used to find out whether a key file in an HSM contains a real private key or a fake PEM key. A fake PEM file does not contain the actual private key material but instead references the private key in the HSM. Such a file can be used to establish SSL/TLS offloading from your web server to AWS CloudHSM. For more information, see SSL/TLS Offload on Linux (p. 271).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

```
IsValidKeyHandlefile -h
IsValidKeyHandlefile -k <private-key-handle>
   -f <private-key-file>
```

Examples

These examples show how to use IsValidKeyHandlefile to determine whether a given key file contains the real key material or fake PEM key material.

**Example : Validate a Real Private Key**

This command confirms that the file called `privateKey.pem` contains real key material.

```
Command: IsValidKeyHandlefile -f privateKey.pem
Input key file has real private key
```

**Example : Invalidate a Fake PEM Key**

This command confirms that the file called `caviumKey.pem` contains fake PEM key material made from key handle 15.

```
Command: IsValidKeyHandlefile -f caviumKey.pem
Input file has invalid key handle: 15
```

Parameters

This command takes the following parameters.

- `-h`
  
  Displays command line help for the command.
  
  Required: Yes

- `-f`
  
  Specifies the name of the file to be checked for valid key material.
  
  Required: Yes
Related Topics

- `getCaviumPrivKey (p. 167)`
- `SSL/TLS Offload on Linux (p. 271)`

**listAttributes**

The `listAttributes` command in key_mgmt_util lists the attributes of an AWS CloudHSM key and the constants that represent them. You use these constants to identify the attributes in `getAttribute (p. 164)` and `setAttribute (p. 190)` commands. For help interpreting the key attributes, see the [Key Attribute Reference (p. 204)].

Before you run any key_mgmt_util command, you must [start key_mgmt_util (p. 123)] and [log in (p. 124)] to the HSM as a crypto user (CU).

**Syntax**

This command has no parameters.

```
listAttributes
```

**Example**

This command lists the key attributes that you can get and change in key_mgmt_util and the constants that represent them. For help interpreting the key attributes, see the [Key Attribute Reference (p. 204)].

To represent all attributes in the `getAttribute (p. 164)` command in key_mgmt_util, use 512.

```
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
```

**Related Topics**

- `listAttributes (p. 106)` in cloudhsm_mgmt_util
- `getAttribute (p. 164)`
- `setAttribute (p. 190)`
• Key Attribute Reference (p. 204)

listUsers

The listUsers command in the key_mgmt_util gets the users in the HSMs, along with their user type and other attributes.

In key_mgmt_util, listUsers returns output that represents all HSMs in the cluster, even if they are not consistent. To get information about the users in each HSM, use the listUsers (p. 188) command in cloudhsm_mgmt_util.

The user commands in key_mgmt_util, listUsers and getKeyInfo (p. 169), are read-only commands that crypto users (CUs) have permission to run. The remaining user management commands are part of cloudhsm_mgmt_util. They are run by crypto officers (CO) who have user management permissions.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

listUsers

listUsers -h

Example

This command lists the users of HSMs in the cluster and their attributes. You can use the User ID attribute to identify users in other commands, such as findKey (p. 141), getAttribute (p. 164), and getKeyInfo (p. 169).

Command:  listUsers

<table>
<thead>
<tr>
<th>Index</th>
<th>User ID</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginFailureCnt</td>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>CU</td>
<td>alice</td>
<td>YES</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>CU</td>
<td>bob</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>CU</td>
<td>trent</td>
<td>YES</td>
</tr>
</tbody>
</table>

Cfm3ListUsers returned: 0x00 : HSM Return: SUCCESS

The output includes the following user attributes:

• **User ID**: Identifies the user in key_mgmt_util and cloudhsm_mgmt_util (p. 75) commands.
• **User type (p. 10)**: Determines the operations that the user can perform on the HSM.
• **User Name**: Displays the user-defined friendly name for the user.
• **MofnPubKey**: Indicates whether the user has registered a key pair for signing quorum authentication tokens (p. 60).
• **LoginFailureCnt**: 188
• **2FA**: Indicates that the user has enabled multi-factor authentication.

**Parameters**

-h
Displays help for the command.
Required: Yes

**Related Topics**

- listUsers (p. 188) in cloudhsm_mgmt_util
- findKey (p. 141)
- getAttribute (p. 164)
- getKeyInfo (p. 169)

**loginHSM and logoutHSM**

The `loginHSM` and `logoutHSM` commands in key_mgmt_util allow you to log in and out of the HSMs in a cluster. Once logged in to the HSMs, you can use key_mgmt_util to perform a variety of key management operations, including public and private key generation, synchronization, and wrapping.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123). In order to manage keys with key_mgmt_util, you must log in to the HSMs as a crypto user (CU) (p. 11).

**Note**
If you exceed five incorrect login attempts, your account is locked out. If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts. To unlock the account, a cryptographic officer (CO) must reset your password using the changePswd (p. 84) command in cloudhsm_mgmt_util.
If you have more than one HSM in your cluster, you may be allowed additional incorrect login attempts before your account is locked out. This is because the CloudHSM client balances load across various HSMs. Therefore, the login attempt may not begin on the same HSM each time.
If you are testing this functionality, we recommend you do so on a cluster with only one active HSM.

**Syntax**

```
loginHSM -h
loginHSM -u <user type>
    -p <password>
    -s <username>
```  

**Example**

This example shows how to log in and out of the HSMs in a cluster with the `loginHSM` and `logoutHSM` commands.

**Example : Log in to the HSMs**

This command logs into the HSMs as a crypto user (CU) with the username `example_user` and password `aws`. Upon successful completion, the command's output will indicate that you have logged into all HSMs in the cluster.
Command: `loginHSM -u CU -s example_user -p aws`

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

Example: Log out of the HSMs

This command logs out of the HSMs. Upon successful completion, the command's output will indicate that you have logged out of all HSMs in the cluster.

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

Parameters

- **-h**
  
  Displays help for this command.

  Required: Yes

- **-u**
  
  Specifies the login user type. In order to use key_mgmt_util, you must log in as a CU.

  Required: Yes

- **-s**
  
  Specifies the login username.

- **-p**
  
  Specifies the login password.

  Required: Yes

Related Topics

- [exit](p. 132)

**setAttribute**

The `setAttribute` command in key_mgmt_util converts a key that is valid only in the current session to a persistent key that exists until you delete it. It does this by changing the value of the token attribute of the key (OBJ_ATTR_TOKEN) from false (0) to true (1). You can only change the attributes of keys that you own.

You can also use the `setAttribute` command in cloudhsm_mgmt_util to change the label, wrap, unwrap, encrypt, and decrypt attributes.
Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

Syntax

```
setAttribute -h
setAttribute -o <object handle>
   -a 1
```

Example

This example shows how to convert a session key to a persistent key.

The first command uses the -sess parameter of `genSymKey` (p. 158) to create a 192-bit AES key that is valid only in the current session. The output shows that the key handle of the new session key is 262154.

```
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
```

This command uses `findKey` (p. 141) to find the session keys in the current session. The output verifies that key 262154 is a session key.

```
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
```

This command uses `setAttribute` to convert key 262154 from a session key to a persistent key. To do so, it changes the value of the token attribute (OBJ_ATTR_TOKEN) of the key from 0 (false) to 1 (true). For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

```
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
```

191
To confirm that key 262154 is now persistent, this command uses `findKey` to search for session keys (`-sess 1`) and persistent keys (`-sess 0`). This time, the command does not find any session keys, but it returns 262154 in the list of persistent keys.

**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

**Parameters**

- **-h**
  Displays help for the command.
  Required: Yes

- **-o**
  Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use `findKey` (p. 141).
  Required: Yes

- **-a**
  Specifies the constant that represents the attribute that you want to change. The only valid value is 1, which represents the token attribute, OBJ_ATTR_TOKEN.

  To get the attributes and their integer values, use `listAttributes` (p. 187).
  Required: Yes

**Related Topics**

- `setAttribute` (p. 112) in cloudhsm_mgmt_util
- `getAttribute` (p. 164)
The `sign` command in `key_mgmt_util` uses a chosen private key to generate a signature for a file.

In order to use `sign`, you must first have a private key in your HSM. You can generate a private key with the `genSymKey` (p. 158), `genRSAKeyPair` (p. 154), or `genECCKeyPair` (p. 150) commands. You can also import one with the `importPrivateKey` (p. 173) command. For more information, see Generate Keys (p. 55).

The `sign` command uses a user-designated signing mechanism, represented by an integer, to sign a message file. For a list of possible signing mechanisms, see Parameters (p. 193).

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

**Syntax**

```
sign -h
sign -f <file name>  
-k <private key handle>  
-m <signature mechanism>  
-out <signed file name>
```

**Example**

This example shows how to use `sign` to sign a file.

**Example : Sign a file**

This command signs a file named `messageFile` with a private key with handle 266309. It uses the SHA256_RSA_PKCS (1) signing mechanism and saves the resulting signed file as `signedFile`.

**Parameters**

This command takes the following parameters.

- `<f>`
  The name of the file to sign.

  Required: Yes
-k

The handle of the private key to be used for signing.

Required: Yes

-m

An integer that represents the signing mechanism to be used for signing. The possible mechanisms correspond to the following integers:

<table>
<thead>
<tr>
<th>Signing Mechanism</th>
<th>Corresponding Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1_RSA_PKCS</td>
<td>0</td>
</tr>
<tr>
<td>SHA256_RSA_PKCS</td>
<td>1</td>
</tr>
<tr>
<td>SHA384_RSA_PKCS</td>
<td>2</td>
</tr>
<tr>
<td>SHA512_RSA_PKCS</td>
<td>3</td>
</tr>
<tr>
<td>SHA224_RSA_PKCS</td>
<td>4</td>
</tr>
<tr>
<td>SHA1_RSA_PKCS_PSS</td>
<td>5</td>
</tr>
<tr>
<td>SHA256_RSA_PKCS_PSS</td>
<td>6</td>
</tr>
<tr>
<td>SHA384_RSA_PKCS_PSS</td>
<td>7</td>
</tr>
<tr>
<td>SHA512_RSA_PKCS_PSS</td>
<td>8</td>
</tr>
<tr>
<td>SHA224_RSA_PKCS_PSS</td>
<td>9</td>
</tr>
<tr>
<td>ECDSA_SHA1</td>
<td>15</td>
</tr>
<tr>
<td>ECDSA_SHA224</td>
<td>16</td>
</tr>
<tr>
<td>ECDSA_SHA256</td>
<td>17</td>
</tr>
<tr>
<td>ECDSA_SHA384</td>
<td>18</td>
</tr>
<tr>
<td>ECDSA_SHA512</td>
<td>19</td>
</tr>
</tbody>
</table>

Required: Yes

-out

The name of the file to which the signed file will be saved.

Required: Yes

Related Topics

- verify (p. 199)
- importPrivateKey (p. 173)
- genRSAKeyPair (p. 154)
- genECCKeyPair (p. 150)
- genSymKey (p. 158)
• Generate Keys (p. 55)

unWrapKey

The unWrapKey command in the key_mgmt_util tool imports a wrapped (encrypted) symmetric or private key from a file into the HSM. It is designed to import encrypted keys that were wrapped by the wrapKey (p. 201) command in key_mgmt_util, but it can also be used to unwrap keys that were wrapped with other tools. However, in those situations, we recommend using the PKCS#11 (p. 214) or JCE (p. 239) software libraries to unwrap the key.

Imported keys work like keys generated by AWS CloudHSM. However, the value of their OBJ_ATTR_LOCAL attribute (p. 204) is zero, which indicates that they were not generated locally.

After you import a key, ensure that you mark or delete the key file. This command does not prevent you from importing the same key material multiple times. The results—multiple keys with distinct key handles and the same key material—make it difficult to track use of the key materials and prevent them from exceeding their cryptographic limits.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

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>]
```

Example

These examples show how to use unWrapKey to import a wrapped key from a file into the HSMs. In the first example, we unwrap a key that was wrapped with the wrapKey (p. 201) key_mgmt_util command, and thus has a header. In the second example, we unwrap a key that was wrapped outside of key_mgmt_util, and thus does not have a header.

**Example : Unwrap a Key (With Header)**

This command imports a wrapped copy of a 3DES symmetric key into an HSM. The key is unwrapped with an AES key with label 6, which is cryptographically identical to the one that was used to wrap the 3DES key. The output shows that the key in the file was unwrapped and imported, and that the imported key's handle is 29.
Example: Unwrap a Key (No Header)

This command imports a wrapped copy of a 3DES symmetric key into an HSM. The key is unwrapped with an AES key with label 6, which is cryptographically identical to the one that was used to wrap the 3DES key. As this 3DES key was not wrapped with key_mgmt_util, the noheader parameter is specified, along with its required accompanying parameters: a key label (unwrapped3DES), key class (4), and key type (21). The output shows that the key in the file was unwrapped and imported, and that the imported key's handle is 8.

Parameters

-h
Displays help for the command.
Required: Yes

-f
The path and name of the file that contains the wrapped key.
Required: Yes

-w
Specifies the wrapping key. Enter the key handle of an AES key or RSA key on the HSM. This parameter is required. To find key handles, use the findKey (p. 141) command.

To create a wrapping key, use genSymKey (p. 158) to generate an AES key (type 31) or genRSAKeyPair (p. 154) to generate an RSA key pair (type 0). If you are using an RSA key pair, be sure to wrap the key with one of the keys, and unwrap it with the other. To verify that a key can be used as a wrapping key, use getAttribute (p. 164) to get the value of the OBJ_ATTR_WRAP attribute, which is represented by constant 262.

Required: Yes

-sess
Creates a key that exists only in the current session. The key cannot be recovered after the session ends.
Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute (p. 190).

Default: The key is persistent.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the min_srv parameter.

This parameter is valid only when the min_srv parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

-m

The value representing the wrapping mechanism. CloudHSM supports the following mechanisms:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_KEY_WRAP_PAD_PKCS5</td>
<td>4</td>
</tr>
<tr>
<td>RSA_AES</td>
<td>7</td>
</tr>
<tr>
<td>RSA_OAEP (for maximum data size, see the note later in this section)</td>
<td>8</td>
</tr>
<tr>
<td>NIST_TDEA_WRAP (key data size must be multiple of 4 bytes)</td>
<td>9</td>
</tr>
<tr>
<td>AES_GCM</td>
<td>10</td>
</tr>
<tr>
<td>CLOUDHSM_AES_GCM</td>
<td>11</td>
</tr>
</tbody>
</table>
Required: Yes

-t

<table>
<thead>
<tr>
<th>Hash algorithm</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1</td>
<td>2</td>
</tr>
<tr>
<td>SHA256</td>
<td>3</td>
</tr>
<tr>
<td>SHA384</td>
<td>4</td>
</tr>
<tr>
<td>SHA512</td>
<td>5</td>
</tr>
<tr>
<td>SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)</td>
<td>6</td>
</tr>
</tbody>
</table>

**Note**
When using the RSA_OAEP wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key and the length of the specified hash as follows: Maximum key size = modulusLengthInBytes-(2*hashLengthInBytes)-2.

Required: No

-noheader

If you are unwrapping a key that was wrapped outside of key_mgmt_util, you must specify this parameter and all other associated parameters.

Required: No

**Note**
If you specify this parameter, you must also specify the following -noheader parameters:

- **-l**
  Specifies the label to be added to the unwrapped key.
  Required: Yes

- **-kc**
  Specifies the class of the key to be unwrapped. The following are acceptable values:
  3 = private key from a public-private key pair
  4 = secret (symmetric) key
  Required: Yes

- **-kt**
  Specifies the type of key to be unwrapped. The following are acceptable values:
  0 = RSA
  1 = DSA
  3 = ECC
  16 = GENERIC_SECRET
  21 = DES3
31 = AES

Required: Yes

You can also optionally specify the following -noheader parameters:

- **-id**
  The ID to be added to the unwrapped key.
  Required: No

- **-i**
  The unwrapping initialization vector (IV) to be used.
  Required: No

**Related Topics**

- wrapKey (p. 201)
- exSymKey (p. 136)
- imSymKey (p. 178)

**verify**

The *verify* command in key_mgmt_util confirms whether or not a file has been signed by a given key. To do so, the *verify* command compares a signed file against a source file and analyzes whether they are cryptographically related based on a given public key and signing mechanism. Files can be signed in AWS CloudHSM with the *sign (p. 193)* operation.

Signing mechanisms are represented by the integers listed in the *parameters (p. 200)* section.

Before you run any key_mgmt_util command, you must *start key_mgmt_util (p. 123)* and *log in (p. 124)* to the HSM as a crypto user (CU).

**Syntax**

```
verify -h
verify -f <message-file>
   -s <signature-file>
   -k <public-key-handle>
   -m <signature-mechanism>
```

**Example**

These examples show how to use *verify* to check whether a certain public key was used to sign a given file.

**Example: Successfully Verify a File Signature**

This command attempts to verify whether a file named hardwarCert.crt was signed by public key 262276 using the SHA256_RSA_PKCS signing mechanism to produce the hardwareCertSigned signed file. Because the given parameters represent a true signing relationship, the command returns a success message.
Command: `verify -f hardwareCert.crt -s hardwareCertSigned -k 262276 -m 1`
Signature verification successful
Cfm3Verify returned: 0x00 : HSM Return: SUCCESS

Example: Prove False Signing Relationship

This command verifies whether a file named `hardwareCert.crt` was signed by public key 262276 using the SHA256_RSA_PKCS signing mechanism to produce the `userCertSigned` signed file. Because the given parameters do not make up a true signing relationship, the command returns an error message.

Command: `verify -f hardwareCert.crt -s userCertSigned -k 262276 -m 1`
Cfm3Verify returned: 0x1b
CSP Error: ERR_BAD_PKCS_DATA

Parameters

This command takes the following parameters.

\(-f\)

The name of the origin message file.

Required: Yes

\(-s\)

The name of the signed file.

Required: Yes

\(-k\)

The handle of the public key that is thought to be used to sign the file.

Required: Yes

\(-m\)

An integer that represents the proposed signing mechanism that is used to sign the file. The possible mechanisms correspond to the following integers:

<table>
<thead>
<tr>
<th>Signing Mechanism</th>
<th>Corresponding Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1_RSA_PKCS</td>
<td>0</td>
</tr>
<tr>
<td>SHA256_RSA_PKCS</td>
<td>1</td>
</tr>
<tr>
<td>SHA384_RSA_PKCS</td>
<td>2</td>
</tr>
<tr>
<td>SHA512_RSA_PKCS</td>
<td>3</td>
</tr>
<tr>
<td>SHA224_RSA_PKCS</td>
<td>4</td>
</tr>
<tr>
<td>SHA1_RSA_PKCS_PSS</td>
<td>5</td>
</tr>
<tr>
<td>SHA256_RSA_PKCS_PSS</td>
<td>6</td>
</tr>
</tbody>
</table>
### Signing Mechanism

<table>
<thead>
<tr>
<th>Signing Mechanism</th>
<th>Corresponding Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA384_RSA_PKCS_PSS</td>
<td>7</td>
</tr>
<tr>
<td>SHA512_RSA_PKCS_PSS</td>
<td>8</td>
</tr>
<tr>
<td>SHA224_RSA_PKCS_PSS</td>
<td>9</td>
</tr>
<tr>
<td>ECDSA_SHA1</td>
<td>15</td>
</tr>
<tr>
<td>ECDSA_SHA224</td>
<td>16</td>
</tr>
<tr>
<td>ECDSA_SHA256</td>
<td>17</td>
</tr>
<tr>
<td>ECDSA_SHA384</td>
<td>18</td>
</tr>
<tr>
<td>ECDSA_SHA512</td>
<td>19</td>
</tr>
</tbody>
</table>

Required: Yes

### Related Topics

- [sign](#) (p. 193)
- [getCert](#) (p. 150)
- [Generate Keys](#) (p. 55)

### wrapKey

The `wrapKey` command in `key_mgmt_util` exports an encrypted copy of a symmetric or private key from the HSM to a file. When you run `wrapKey`, you specify the key to export, a key on the HSM to encrypt (wrap) the key that you want to export, and the output file.

The `wrapKey` command writes the encrypted key to a file that you specify, but it does not remove the key from the HSM or prevent you from using it in cryptographic operations. You can export the same key multiple times.

Only the owner of a key, that is, the crypto user (CU) who created the key, can export it. Users who share the key can use it in cryptographic operations, but they cannot export it.

To import the encrypted key back into the HSM, use `unWrapKey` (p. 195). To export a plaintext key from an HSM, use `exSymKey` (p. 136) or `exportPrivateKey` (p. 133) as appropriate. The `aesWrapUnwrap` (p. 127) command cannot decrypt (unwrap) keys that `wrapKey` encrypts.

Before you run any `key_mgmt_util` command, you must start `key_mgmt_util` (p. 123) and log in (p. 124) to the HSM as a crypto user (CU).

### Syntax

```bash
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>]
```
Example

This command exports a 192-bit Triple DES (3DES) symmetric key (key handle 7). It uses a 256-bit AES key in the HSM (key handle 14) to wrap key 7. Then, it writes the encrypted 3DES key to the 3DES-encrypted.key file.

The output shows that key 7 (the 3DES key) was successfully wrapped and written to the specified file. The encrypted key is 307 bytes long.

```
Example

Example

This command exports a 192-bit Triple DES (3DES) symmetric key (key handle 7). It uses a 256-bit AES key in the HSM (key handle 14) to wrap key 7. Then, it writes the encrypted 3DES key to the 3DES-encrypted.key file.

The output shows that key 7 (the 3DES key) was successfully wrapped and written to the specified file. The encrypted key is 307 bytes long.

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
```

Parameters

- **-h**
  
  Displays help for the command.
  
  Required: Yes

- **-k**
  
  The key handle of the key that you want to export. Enter the key handle of a symmetric or private key that you own. To find key handles, use the `findKey (p. 141)` command.

  To verify that a key can be exported, use the `getAttribute (p. 164)` command to get the value of the `OBJ_ATTR_EXTRACTABLE` attribute, which is represented by constant 354. For help interpreting the key attributes, see the Key Attribute Reference (p. 204).

  You can export only those keys that you own. To find the owner of a key, use the `getKeyInfo (p. 169)` command.

  Required: Yes

- **-w**
  
  Specifies the wrapping key. Enter the key handle of an AES key or RSA key on the HSM. This parameter is required. To find key handles, use the `findKey (p. 141)` command.

  To create a wrapping key, use `genSymKey (p. 158)` to generate an AES key (type 31) or `genRSAKeyPair (p. 154)` to generate an RSA key pair (type 0). If you are using an RSA key pair, be sure to wrap the key with one of the keys, and unwrap it with the other. To verify that a key can be used as a wrapping key, use `getAttribute (p. 164)` to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by constant 262.

  Required: Yes
-out

The path and name of the output file. When the command succeeds, this file contains an encrypted copy of the exported key. If the file already exists, the command overwrites it without warning.

Required: Yes

-m

The value representing the wrapping mechanism. CloudHSM supports the following mechanisms:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES_KEY_WRAP_PAD_PKCS5</td>
<td>4</td>
</tr>
<tr>
<td>RSA_AES</td>
<td>7</td>
</tr>
<tr>
<td>RSA_OAEP (for maximum data size, see the note later in this section)</td>
<td>8</td>
</tr>
<tr>
<td>NIST_TDEA_WRAP (key data size must be multiple of 4 bytes)</td>
<td>9</td>
</tr>
<tr>
<td>AES_GCM</td>
<td>10</td>
</tr>
<tr>
<td>CLOUDHSM_AES_GCM</td>
<td>11</td>
</tr>
</tbody>
</table>

Required: Yes

-t

The value representing the hash algorithm. CloudHSM supports the following algorithms:

<table>
<thead>
<tr>
<th>Hash algorithm</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1</td>
<td>2</td>
</tr>
<tr>
<td>SHA256</td>
<td>3</td>
</tr>
<tr>
<td>SHA384</td>
<td>4</td>
</tr>
<tr>
<td>SHA512</td>
<td>5</td>
</tr>
<tr>
<td>SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)</td>
<td>6</td>
</tr>
</tbody>
</table>

Required: No

**Note**

When using the RSA_OAEP wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key and the length of the specified hash as follows: Maximum key size = (modulusLengthInBytes-2*hashLengthInBytes-2).

-aad

The file name containing AAD.

**Note**

Valid only for AES_GCM and CLOUDHSM_AES_GCM mechanisms.
Required: No

-noheader

Omits the header that specifies CloudHSM-specific key attributes (p. 125). Use this parameter only if you want to unwrap the key with tools outside of key_mgmt_util.

Required: No

-i

The initialization vector (IV) (hex value).

Note
Valid only when passed with the -noheader parameter for CLOUDHSM_AES_KEY_WRAP, NIST_AES_WRAP, and NIST_TDEA_WRAP mechanisms.

Required: No

-iv_file

The file in which you want to write the IV value obtained in response.

Note
Valid only when passed with the -noheader parameter for AES_GCM mechanism.

Required: No

-tag_size

The size of tag to be saved along with wrapped blob.

Note
Valid only when passed with the -noheader parameter for AES_GCM and CLOUDHSM_AES_GCM mechanisms. Minimum tag size is eight.

Required: No

Related Topics

• exSymKey (p. 136)
• imSymKey (p. 178)
• unWrapKey (p. 195)

Key Attribute Reference

The key_mgmt_util commands use constants to represent the attributes of keys in an HSM. This topic can help you to identify the attributes, find the constants that represent them in commands, and understand their values.

You set the attributes of a key when you create it. To change the token attribute, which indicates whether a key is persistent or exists only in the session, use the setAttribute (p. 190) command in key_mgmt_util. To change the label, wrap, unwrap, encrypt, or decrypt attributes, use the setAttribute command in cloudhsm_mgmt_util.

To get a list of attributes and their constants, use listAttributes (p. 187). To get the attribute values for a key, use getAttribute (p. 164).

The following table lists the key attributes, their constants, and their valid values.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Constant</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ATTR_CLASS</td>
<td>0</td>
<td>2: Public key in a public–private key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: Private key in a public–private key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: Secret (symmetric) key.</td>
</tr>
<tr>
<td>OBJ_ATTR_TOKEN</td>
<td>1</td>
<td>0: False. Session key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True. Persistent key.</td>
</tr>
<tr>
<td>OBJ_ATTR_PRIVATE</td>
<td>2</td>
<td>0: False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True. This attribute indicates whether unauthenticated users can list the attributes of the key. Since the CloudHSM PKCS#11 provider currently does not support public sessions, all keys (including public keys in a public-private key pair) have this attribute set to 1.</td>
</tr>
<tr>
<td>OBJ_ATTR_LABEL</td>
<td>3</td>
<td>User-defined string. It does not have to be unique in the cluster.</td>
</tr>
<tr>
<td>OBJ_ATTR_TRUSTED</td>
<td>134</td>
<td>0: False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_KEY_TYPE</td>
<td>256</td>
<td>0: RSA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: DSA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: EC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16: Generic secret.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18: RC4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21: Triple DES (3DES).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31: AES.</td>
</tr>
<tr>
<td>OBJ_ATTR_ID</td>
<td>258</td>
<td>User-defined string. Must be unique in the cluster. The default is an empty string.</td>
</tr>
<tr>
<td>OBJ_ATTR_SENSITIVE</td>
<td>259</td>
<td>0: False. Public key in a public–private key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_ENCRYPT</td>
<td>260</td>
<td>0: False. The key can be used to encrypt data.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Constant</td>
<td>Values</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OBJ_ATTR_DECRYPT</td>
<td>261</td>
<td>0: False. 1: True. The key can be used to decrypt data.</td>
</tr>
<tr>
<td>OBJ_ATTR_WRAP</td>
<td>262</td>
<td>0: False. 1: True. The key can be used to encrypt keys.</td>
</tr>
<tr>
<td>OBJ_ATTR_UNWRAP</td>
<td>263</td>
<td>0: False. 1: True. The key can be used to decrypt keys.</td>
</tr>
<tr>
<td>OBJ_ATTR_SIGN</td>
<td>264</td>
<td>0: False. 1: True. The key can be used for signing (private keys).</td>
</tr>
<tr>
<td>OBJ_ATTR_VERIFY</td>
<td>266</td>
<td>0: False. 1: True. The key can be used for verification (public keys).</td>
</tr>
<tr>
<td>OBJ_ATTR_DERIVE</td>
<td>268</td>
<td>0: False. 1: True. The function derives the key.</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS</td>
<td>288</td>
<td>The modulus that was used to generate an RSA key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS_BITS</td>
<td>289</td>
<td>The length of the modulus used to generate an RSA key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_PUBLIC_EXPONENT</td>
<td>290</td>
<td>The public exponent used to generate an RSA key pair.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_VALUE_LEN</td>
<td>353</td>
<td>Key length in bytes.</td>
</tr>
<tr>
<td>OBJ_ATTR_EXTRACTABLE</td>
<td>354</td>
<td>0: False. 1: True. The key can be exported from the HSMs.</td>
</tr>
<tr>
<td>OBJ_ATTR_LOCAL</td>
<td>355</td>
<td>0: False. The key was imported into the HSMs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Constant</td>
<td>Values</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>OBJ_ATTR_NEVER_EXTRACTABLE</td>
<td>356</td>
<td>0: False. The key cannot be exported from the HSMs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_ALWAYS_SENSITIVE</td>
<td>357</td>
<td>0: False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_DESTROYABLE</td>
<td>370</td>
<td>0: False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_KCV</td>
<td>371</td>
<td>Key check value of the key. For more information, see Additional Details (p. 207).</td>
</tr>
<tr>
<td>OBJ_ATTR_ALL</td>
<td>512</td>
<td>Represents all attributes.</td>
</tr>
<tr>
<td>OBJ_ATTR_WRAP_WITH_TRUSTED</td>
<td>528</td>
<td>0: False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_EKCV</td>
<td>4099</td>
<td>EKCV is a checksum value generated using the key bytes.</td>
</tr>
<tr>
<td>OBJ_ATTR_WRAP_TEMPLATE</td>
<td>1073742353</td>
<td>Values should use the attribute template to match the key wrapped using this wrapping key.</td>
</tr>
<tr>
<td>OBJ_ATTR_UNWRAP_TEMPLATE</td>
<td>1073742354</td>
<td>Values should use the attribute template applied to any key unwrapped using this wrapping key.</td>
</tr>
</tbody>
</table>

**Additional Details**

**Key check value (kcv)**

The key check value (KCV) is a 3-byte hash or checksum of a key that is generated when the HSM imports or generates a key. You can also calculate a KCV outside of the HSM, such as after you export a key. You can then compare the KCV values to confirm the identity and integrity of the key. To get the KCV of a key, use `getAttribute` (p. 164).

AWS CloudHSM uses the following standard method to generate a key check value:

- **Symmetric keys**: First 3 bytes of the result of encrypting a zero-block with the key.
- **Asymmetric key pairs**: First 3 bytes of the SHA-1 hash of the public key.
- **HMAC keys**: KVC for HMAC keys is not supported at this time.

**Configure Tool**

AWS CloudHSM automatically synchronizes data among all HSMs in a cluster. The `configure` tool updates the HSM data in the configuration files that the synchronization mechanisms use. Use `configure` to
refresh the HSM data before you use the command line tools, especially when the HSMs in the cluster have changed.

**Syntax**

```bash
configure -h | --help
    -a <ENI IP address>
    -m [-i <daemon_id>]
    --ssl --pkey <private key file> --cert <certificate file>
```

**Examples**

These examples show how to use the `configure` tool.

**Example : Update the HSM Data for the AWS CloudHSM Client and key_mgmt_util**

This example uses the `-a` parameter of `configure` to update the HSM data for the AWS CloudHSM client and key_mgmt_util. This command is also the first step in updating the cloudhsm_mgmt_util configuration file.

Before updating the `-a` parameter, stop the AWS CloudHSM client. This prevents conflicts that might occur while `configure` edits the client's configuration file. If the client is already stopped, this command has no effect, so you can use it in a script.

Amazon Linux

```bash
$ sudo stop cloudhsm-client
```

Amazon Linux 2

```bash
$ sudo service cloudhsm-client stop
```

CentOS 6

```bash
$ sudo stop cloudhsm-client
```

CentOS 7

```bash
$ sudo service cloudhsm-client stop
```

RHEL 6

```bash
$ sudo stop cloudhsm-client
```

RHEL 7

```bash
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```bash
$ sudo service cloudhsm-client stop
```

Windows

- For Windows client 1.1.2+:
For Windows clients 1.1.1 and older:

Use `Ctrl+C` in the command window where you started the AWS CloudHSM client.

Next, get the ENI IP address of any one of the HSMs in your cluster. This command uses the `describe-clusters` command in the AWS CLI, but you can also use the `DescribeClusters` operation or the `Get-HSM2Cluster` PowerShell cmdlet.

This excerpt of the output shows the ENI IP addresses of the HSMs in a sample cluster. We can use either of the IP addresses in the next command.

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { "Hsms": [
            { "EniIp": "10.0.0.9",
            { "EniIp": "10.0.1.6",
            ...
```

This step uses the `-a` parameter of `configure` to add the `10.0.0.9` ENI IP address to the configurations files.

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 6

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 6

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```
Ubuntu 16.04 LTS

```bash
# sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Windows

```bash
c:\Program Files\Amazon\CloudHSM>configure.exe -a 10.0.0.9
```

Next, restart the AWS CloudHSM client. When the client starts, it uses the ENI IP address in its configuration file to query the cluster. Then, it writes the ENI IP addresses of all HSMs in the cluster to the cluster.info file.

Amazon Linux

```bash
# sudo start cloudhsm-client
```

Amazon Linux 2

```bash
# sudo service cloudhsm-client start
```

CentOS 6

```bash
# sudo start cloudhsm-client
```

CentOS 7

```bash
# sudo service cloudhsm-client start
```

RHEL 6

```bash
# sudo start cloudhsm-client
```

RHEL 7

```bash
# sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```bash
# sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:
  ```bash
  C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:
  ```bash
  C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
  ```
When the command completes, the HSM data that the AWS CloudHSM client and key_mgmt_util use is complete and accurate. Before using cloudhsm_mgmt_util, update the `-m` parameter of `configure`, as shown in the following example.

**Example: Update the HSM Data for cloudhsm_mgmt_util**

This example uses the `-m configure` command to copy the updated HSM data from the `cluster.info` file to the `cloudhsm_mgmt_util.cfg` file that cloudhsm_mgmt_util uses.

Before running the `-m`, stop the AWS CloudHSM client, run the `-a` command, and then restart the AWS CloudHSM client, as shown in the previous example (p. 208). This ensures that the data copied into the `cloudhsm_mgmt_util.cfg` file from the `cluster.info` file is complete and accurate.

**Amazon Linux**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**Amazon Linux 2**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**CentOS 6**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**CentOS 7**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**RHEL 6**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**RHEL 7**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**Ubuntu 16.04 LTS**

```bash
# sudo /opt/cloudhsm/bin/configure -m
```

**Windows**

```bash
c:\Program Files\Amazon\CloudHSM>configure.exe -m
```

**Parameters**

- **-h | --help**
  
  Displays command syntax.

  Required: Yes
-a <ENI IP address>

Adds the specified HSM elastic network interface (ENI) IP address to AWS CloudHSM configuration files. Enter the ENI IP address of any one of the HSMs in the cluster. It does not matter which one you select.

To get the ENI IP addresses of the HSMs in your cluster, use the DescribeClusters operation, the describe-clusters AWS CLI command, or the Get-HSM2Cluster PowerShell cmdlet.

Note
Before running the -a configure command, stop the AWS CloudHSM client. Then, when the -a command completes, restart the AWS CloudHSM client. For details, see the examples (p. 208).

This parameter edits the following configuration files:
- /opt/cloudhsm/etc/cloudhsm_client.cfg: Used by AWS CloudHSM client and key_mgmt_util (p. 122).
- /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg: Used by cloudhsm_mgmt_util (p. 75).

When the AWS CloudHSM client starts, it uses the ENI IP address in its configuration file to query the cluster and update the cluster.info file (/opt/cloudhsm/daemon/1/cluster.info) with the correct ENI IP addresses for all HSMs in the cluster.

Required: Yes

-m

Updates the HSM ENI IP addresses in the configuration file that cloudhsm_mgmt_util uses.

When you update the -a parameter of configure and then start the AWS CloudHSM client, the client daemon queries the cluster and updates the cluster.info files with the correct HSM IP addresses for all HSMs in the cluster. Running the -m configure command completes the update by copying the HSM IP addresses from the cluster.info to the cloudhsm_mgmt_util.cfg configuration file that cloudhsm_mgmt_util uses.

Be sure to run -a configure command and restart the AWS CloudHSM client before running the -m command. This ensures that the data copied into cloudhsm_mgmt_util.cfg from cluster.info is complete and accurate.

Required: Yes

-i

Specifies an alternate client daemon. The default value represents the AWS CloudHSM client.

Default: 1

Required: No

--ssl

Replaces the SSL key and certificate for the cluster with the specified private key and certificate. When you use this parameter, the --pkey and --cert parameters are required.

Required: No

--pkey

Specifies the new private key. Enter the path and file name of the file that contains the private key.

Required: Yes if --ssl is specified. Otherwise, this should not be used.
---cert

Specifies the new certificate. Enter the path and file name of the file that contains the certificate. The certificate should chain up to the customerCA.crt certificate, the self-signed certificate used to initialize the cluster. For more information, see Initialize the Cluster.

Required: Yes if --ssl is specified. Otherwise, this should not be used.

Related Topics

- Set Up key_mgmt_util (p. 122)
- Prepare to run cloudhsm_mgmt_util (p. 76)
Using the AWS CloudHSM Software Libraries

The AWS CloudHSM software libraries integrate your applications with the HSMs in your cluster. The libraries enable your application to perform cryptographic operations on the HSMs.

For detailed information about supported platforms and a full version history, see AWS CloudHSM Client and Software Information (p. 355).

Topics
- AWS CloudHSM Software Library for PKCS #11 (p. 214)
- AWS CloudHSM Dynamic Engine for OpenSSL (p. 236)
- AWS CloudHSM Software Library for Java (p. 239)
- KSP and CNG Providers for Windows (p. 261)

AWS CloudHSM Software Library for PKCS #11

The AWS CloudHSM software library for PKCS #11 is a PKCS #11 standard implementation that communicates with the HSMs in your AWS CloudHSM cluster. It is supported only on Linux and compatible operating systems. This library is compliant with PKCS #11 version 2.40, and implements the following key types, mechanisms, and API operations.

Topics
- Installing the AWS CloudHSM Software Library for PKCS #11 (p. 214)
- Authenticating to PKCS #11 (p. 220)
- Supported PKCS #11 Key Types (p. 220)
- Supported PKCS #11 Mechanisms (p. 220)
- Supported PKCS #11 API Operations (p. 224)
- Supported PKCS #11 Attributes (p. 225)
- Code Samples for AWS CloudHSM Software Library for PKCS#11 (p. 236)

Installing the AWS CloudHSM Software Library for PKCS #11

With the AWS CloudHSM software libraries for PKCS #11, you can to build PKCS #11–compatible applications that use the HSMs in your AWS CloudHSM cluster. You can use the standard AWS CloudHSM PKCS #11 library or the AWS CloudHSM PKCS #11 library that uses a Redis cache (p. 216). Both of the AWS CloudHSM libraries for PKCS #11 are supported only on Linux operating systems.

Topics
- Prerequisites (p. 215)
- Install the PKCS #11 Library (p. 215)
- Install the PKCS #11 Library with Redis (Not applicable for SDK version 3.0 or higher) (p. 216)
Prerequisites

The AWS CloudHSM software libraries for PKCS #11 require the AWS CloudHSM client. If you haven't installed and configured the AWS CloudHSM client, do that now by following the steps at Install the Client (Linux) (p. 34). After you install and configure the client, use the following command to start it.

Amazon Linux

```bash
# sudo start cloudhsm-client
```

Amazon Linux 2

```bash
# sudo service cloudhsm-client start
```

CentOS 6

```bash
# sudo start cloudhsm-client
```

CentOS 7

```bash
# sudo service cloudhsm-client start
```

RHEL 6

```bash
# sudo start cloudhsm-client
```

RHEL 7

```bash
# sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```bash
# sudo service cloudhsm-client start
```

Install the PKCS #11 Library

The following command downloads and installs (or updates) the AWS CloudHSM software library for PKCS #11. This step is required for the standard AWS CloudHSM PKCS #11 library and the AWS CloudHSM PKCS #11 library with Redis (p. 216).

Amazon Linux

```bash
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm

$ sudo yum install -y ./cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```bash
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```
Installing the PKCS #11 Library

When the installation succeeds, the PKCS #11 libraries are `/opt/cloudhsm/lib`.

Install the PKCS #11 Library with Redis (Not applicable for SDK version 3.0 or higher)

Prior to version 3.0, AWS CloudHSM provided an optional software library for PKCS #11 that uses a Redis cache. The cache stores key handles and attributes locally so you can access them without calling into your HSMs.

**Note**
Redis is not necessary for version 3.0 or later. If you are using AWS CloudHSM version 2.0.4 or earlier, AWS CloudHSM provides an optional software library for PKCS #11 that uses a Redis cache. The cache stores key handles and attributes locally so you can access them without calling into your HSMs.
When you build the cache, you specify the crypto user (CU) that your PKCS #11 application uses to be authenticated (p. 220). The cache is preloaded with the keys that the CU owns and shares. It is automatically updated when your application uses functions in the PKCS #11 library to make changes in the HSMs. Examples include creating or deleting keys or changing keys attributes. The cache is not aware of any other keys on the HSM.

Caching can improve the performance of your PKCS #11 application, but it might not be the right choice for all applications. Consider the following:

- Redis caches all PKCS #11 library operations that run on the host, but it's not aware of operations that are performed outside the library. For example, if you use the command line tools (p. 75) or the software library for Java (p. 239) to manage keys in your HSMs, those operations do not update the cache. You can rebuild the cache to update it to the new state of the HSMs, but the cache is not synchronized with the HSMs automatically.

- If you have other applications that use Redis on the same host, don't use the PKCS #11 library with Redis. The PKCS #11 library configures Redis to recognize it as the only Redis consumer on the host.

To install the PKCS #11 Library with Redis use the Extra Packages for Enterprise Linux (EPEL) repository. Then enable and configure Redis to work with AWS CloudHSM and PKCS #11.

Some steps in this process are required only on selected operating systems.

**Step 1: Install the AWS CloudHSM PKCS #11 Library**

To install the PKCS #11 Library with Redis, you must first install the standard AWS CloudHSM PKCS #11 library (p. 215). This library is required.

**Required for Redis on:** All supported operating systems.

**Step 2: Install the EPEL Repository**

This step installs the Extra Packages for Enterprise Linux (EPEL) repository. It is required only on operating systems that do not include EPEL.

**Required for Redis only on:** Amazon Linux 2, CentOS 6, CentOS 7, Red Hat Enterprise Linux (RHEL) 6, Red Hat Enterprise Linux (RHEL) 7

Amazon Linux

  No action required.

Amazon Linux 2

1. Download the EPEL repository.

   ```
   ```

2. Install the EPEL repository.

   ```
   sudo yum install epel-release-latest-7.noarch.rpm
   ```

CentOS 6

1. Download the EPEL repository.
Installing the PKCS #11 Library

2. Install the EPEL repository.

```bash
sudo yum install epel-release-latest-6.noarch.rpm
```

CentOS 7

1. Download the EPEL repository.

```bash
```

2. Install the EPEL repository.

```bash
sudo yum install epel-release-latest-7.noarch.rpm
```

RHEL 6

1. Download the EPEL repository.

```bash
```

2. Install the EPEL repository.

```bash
sudo yum install epel-release-latest-6.noarch.rpm
```

RHEL 7

1. Download the EPEL repository.

```bash
```

2. Install the EPEL repository.

```bash
sudo yum install epel-release-latest-7.noarch.rpm
```

Ubuntu 16.04 LTS

No action required.

Step 3: Prepare for Redis

This step includes system-specific tasks that must be completed before you install and configure the PKCS #11 library for Redis.

**Required for Redis only on:** Amazon Linux, CentOS 7, Red Hat Enterprise Linux (RHEL) 6

Amazon Linux

This step enables the EPEL repository. It is required only on Amazon Linux, but you can use this procedure to verify that EPEL is enabled on any Linux operating system.
1. Open the `/etc/yum.repos.d/epel.repo` file in a text editor. This step requires administrative permissions (`sudo`).

2. In the `[epel]` configuration in the file, set the value of `enabled` to `1`, as shown in the following example. Then, save the file and close it.

```
[epel]
name=Extra Packages for Enterprise Linux 6 - $basearch
#baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
```

Amazon Linux 2

No action required.

CentOS 6

No action required.

CentOS 7

This command disables a Security-Enhanced Linux (SELinux) policy that prevents Redis from using AWS CloudHSM resources.

```
sudo semanage module -d redis
```

RHEL 6

This command eliminates the TTY requirement in the `sudoers` file. The `sudoers` file contains the rules for the `sudo` command.

1. Use `visudo` editor to edit the `/etc/sudoers` file.
2. Comment out the `Defaults requiretty` statement. Then, save the file and quit `visudo`.

```
# Defaults requiretty
```

RHEL 7

No action required.

Ubuntu 16.04 LTS

No action required.

**Step 4: Install, Configure, and Build the Redis Cache**

Use the following procedure to install and configure the Redis package for the AWS CloudHSM library for PKCS #11 and build the cache.

**Required for Redis on:** All supported operating systems.

1. Use the `setup_redis` script to install Redis and configure it to work with the AWS CloudHSM PKCS #11 library for Redis.
2. Start the Redis service.

3. Use the `build_keystore` command to build the Redis cache. Type the name and password of the crypto user (CU) (p. 11) that your PKCS #11 application uses for authentication (p. 220).

   The cache is preloaded with the keys that the specified CU owns and shares. It is updated automatically when your application makes changes in the HSMs on behalf of the CU. Examples include creating or deleting keys, or changing key attributes. The cache is not aware of any other keys on the HSMs.

   ```bash
   /opt/cloudhsm/bin/build_keystore -s <CU user name> -p <CU password>
   ```

**Authenticating to PKCS #11**

When you use PKCS #11 with AWS CloudHSM, your application runs as a particular crypto user (CU) (p. 10) in your HSMs. Your application can view and manage only the keys that the CU owns and shares. You can use an existing CU in your HSMs or create a new CU (p. 52) for your application.

To specify the CU to PKCS #11, use the pin parameter of the PKCS #11 C_Login function. For AWS CloudHSM, the pin parameter has the following format:

```
<CU_user_name>:<password>
```

For example, the following command sets the PKCS #11 pin to the CU with user name `CryptoUser` and password `CUPassword123!`.

```
CryptoUser:CUPassword123!
```

**Supported PKCS #11 Key Types**

The AWS CloudHSM software library for PKCS #11 supports the following key types.

- **RSA** – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.
- **ECDSA** – Generate keys with the P-224, P-256, P-384, P-521, and secp256k1 curves. Only the P-256, P-384, and secp256k1 curves are supported for sign and verify.
- **AES** – 128, 192, and 256-bit AES keys.
- **Triple DES (3DES)** – 192-bit keys.
- **GENERIC_SECRET** – 1 to 64 bytes.

**Supported PKCS #11 Mechanisms**

The AWS CloudHSM software library for PKCS #11 supports the following algorithms:

- **Encryption and decryption** – AES-CBC, AES-CTR, AES-ECB, AES-GCM, DES3-CBC, DES3-ECB, RSA-OAEP, and RSA-PKCS
• **Sign and verify** – RSA, HMAC, and ECDSA; with and without hashing
• **Hash/digest** – SHA1, SHA224, SHA256, SHA384, and SHA512
• **Key wrap** – AES Key Wrap, AES-GCM, RSA-AES, and RSA-OAEP
• **Key derivation** – ECDH

The AWS CloudHSM software library for PKCS #11 is compliant with PKCS #11 version 2.40. To invoke a cryptographic feature using PKCS#11, call a function with a given mechanism. The following table summarizes the combinations of functions and mechanisms supported by AWS CloudHSM.

**Interpreting the Supported PKCS #11 Mechanism-Function Table**

A ✔ mark indicates that CloudHSM supports the mechanism for the function. We do not support all possible functions listed in the PKCS #11 specification. A ✗ mark indicates that CloudHSM does not yet support the mechanism for the given function, even though the PKCS #11 standard allows it. Empty cells indicate that PKCS #11 standard does not support the mechanism for the given function.

**Supported PKCS #11 Mechanisms and Functions**

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Generate Key or Key Pair</th>
<th>Sign &amp; Verify</th>
<th>Digest</th>
<th>Encrypt &amp; Decrypt</th>
<th>Derive Key</th>
<th>Wrap &amp; UnWrap</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_RSA_X9_31_KEY_PAIR_GEN</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RSA_PKCS</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RSA_PKCS_OAEP</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA1_RSA_PKCS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>CKM_SHA224_RSA_PKCS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA256_RSA_PKCS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA384_RSA_PKCS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA512_RSA_PKCS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA1_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA224_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA256_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA384_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_SHA512_RSA_PKCS_PSS</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_EC_KEY_PAIR_GEN</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_ECDSA</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_ECDSA_SHA1</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_ECDSA_SHA224</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKM_ECDSA_SHA256</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table above lists the supported PKCS #11 mechanisms and functions supported by AWS CloudHSM. Each entry indicates whether the mechanism is supported for each function, with ✔ for support and ✗ for non-support. Empty cells indicate the PKCS #11 standard does not support the mechanism for the given function.
## Supported PKCS #11 Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_ECDSA_SHA384</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_ECDSA_SHA512</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_ECDH1_DERIVE</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_GENERIC_SECRET_KEY_GEN</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_KEY_GEN</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_ECB</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_CTR</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_CBC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_CBC_PAD</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_CBC_PAD</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_DES3_KEY_GEN</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_DES3_CBC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_DES3_CBC_PAD</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_DES3_ECB</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_GCM</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_CLOUDHSM_AES_GCM</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_CLOUDHSM_AES_GCM</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA_1</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA_1_HMAC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA224_HMAC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA256_HMAC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA384_HMAC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_SHA512_HMAC</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_AES_KEY_WRAP</td>
<td>✔</td>
</tr>
<tr>
<td>CKM_RSA_AES_KEY_WRAP</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Annotations:**

1. Single-part operations only.

2. Mechanism is functionally identical to the CKM_RSA_PKCS_KEY_PAIR_GEN mechanism, but offers stronger guarantees for $p$ and $q$ generation.
3.1 When hashing data using any of the following mechanisms, data buffer less than 16 KB in size is hashed on the HSM. Larger data buffer, which is between 16 KB and the maximum data size, is hashed locally in software. The following table lists the maximum data size set for each mechanism:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Maximum Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_SHA_1</td>
<td>16296</td>
</tr>
<tr>
<td>CKM_SHA224</td>
<td>16264</td>
</tr>
<tr>
<td>CKM_SHA256</td>
<td>16296</td>
</tr>
<tr>
<td>CKM_SHA384</td>
<td>16232</td>
</tr>
<tr>
<td>CKM_SHA512</td>
<td>16232</td>
</tr>
</tbody>
</table>

3.2 When operating on data by using any of the following mechanisms, if the data buffer exceeds the maximum data size, the operation results in an error. The following table lists maximum data size set for each mechanism:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Maximum Data Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKM_SHA_1_HMAC</td>
<td>16288</td>
</tr>
<tr>
<td>CKM_SHA224_HMAC</td>
<td>16256</td>
</tr>
<tr>
<td>CKM_SHA256_HMAC</td>
<td>16288</td>
</tr>
<tr>
<td>CKM_SHA384_HMAC</td>
<td>16224</td>
</tr>
<tr>
<td>CKM_SHA512_HMAC</td>
<td>16224</td>
</tr>
<tr>
<td>CKM_SHA1_RSA_PKCS</td>
<td>16296</td>
</tr>
<tr>
<td>CKM_SHA224_RSA_PKCS</td>
<td>16264</td>
</tr>
<tr>
<td>CKM_SHA256_RSA_PKCS</td>
<td>16296</td>
</tr>
<tr>
<td>CKM_SHA364_RSA_PKCS</td>
<td>16232</td>
</tr>
<tr>
<td>CKM_SHA512_RSA_PKCS</td>
<td>16232</td>
</tr>
<tr>
<td>CKM_AES_CBC</td>
<td>16272</td>
</tr>
<tr>
<td>CKM_AES_GCM</td>
<td>16224</td>
</tr>
<tr>
<td>CKM_DES3_CBC</td>
<td>16280</td>
</tr>
</tbody>
</table>

4 When performing AES-GCM encryption, the HSM does not accept initialization vector (IV) data from the application. You must use an IV that it generates. The 12-byte IV provided by the HSM is written into the memory reference pointed to by the pIV element of the CK_GCM_PARAMS parameters structure that you supply. To prevent user confusion, PKCS #11 SDK in version 1.1.1 and later ensures that pIV points to a zeroized buffer when AES-GCM encryption is initialized.

5 Mechanism is implemented to support SSL/TLS offload cases and is executed only partially within the HSM. Before using this mechanism, see "Issue: ECDH key derivation is executed only partially within the HSM" in Known Issues for the PKCS #11 SDK (p. 342).

6 The following CK_MECHANISM_TYPE and CK_RSA_PKCS_MGF_TYPE are supported as CK_RSA_PKCS_OAEP_PARAMS for CKM_RSA_PKCS_OAEP:
AWS CloudHSM User Guide
Supported PKCS #11 API Operations

- CKM_SHA_1 using CKG_MGF1_SHA1
- CKM_SHA224 using CKG_MGF1_SHA224
- CKM_SHA256 using CKG_MGF1_SHA256
- CKM_SHA384 using CKG_MGF1_SHA384
- CKM_SHA512 using CKG_MGF1_SHA512

Vendor-defined mechanism. In order to use the CloudHSM vendor defined mechanisms, PKCS#11 applications must include /opt/cloudhsm/include/pkcs11t.h during compilation. For more information about vendor-defined mechanism, see the following description:

CKM_CLOUDHSM_AES_GCM: This proprietary mechanism is a programmatically safer alternative to the standard CKM_AES_GCM. It prepends the IV generated by the HSM to the ciphertext instead of writing it back into the CK_GCM_PARAMS structure that is provided during cipher initialization. You can use this mechanism with C_Encrypt, C_WrapKey, C_Decrypt, and C_UnwrapKey functions. When using this mechanism, the pIV variable in the CK_GCM_PARAMS struct must be set to NULL. When using this mechanism with C_Decrypt and C_UnwrapKey, the IV is expected to be prepended to the ciphertext that is being unwrapped.

Supported PKCS #11 API Operations

The AWS CloudHSM software library for PKCS #11 supports the following PKCS #11 API operations.

- C_CreateObject
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DestroyObject
- C_DigestInit
- C_Digest
- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_Finalize
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetOperationState
- C_GetSessionInfo
AWS CloudHSM User Guide
Supported PKCS #11 Attributes

- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignRecover
- C_SignRecoverInit
- C_SignUpdate
- C_UnWrapKey
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyRecover
- C_VerifyRecoverInit
- C_VerifyUpdate
- C_WrapKey

Supported PKCS #11 Attributes

A key object can be a public, private, or secret key. Actions permitted on a key object are specified through attributes. Attributes are defined when the key object is created. When you use CloudHSM’s PKCS #11 SDK, we assign default values as specified by the PKCS #11 standard.

**Note**

CloudHSM does not support all attributes listed in the PKCS #11 specification. We are compliant with the specification for all attributes we support. These attributes are listed in the respective tables.

Cryptographic functions such as C_CreateObject, C_GenerateKey, C_GenerateKeyPair, C_UnWrapKey, and C_DeriveKey that create, modify, or copy objects take an attribute template as one of their parameters. For more information about passing an attribute template during object creation, see [Generate keys through PKCS #11 library sample](#).

Interpreting the PKCS #11 Attributes Table

The PKCS #11 table contains a list of attributes that differ by key types. It indicates whether a given attribute is supported for a particular key type when using a specific cryptographic function with AWS CloudHSM.

**Legend:**

- ✓ indicates that CloudHSM supports the attribute for the specific key type.
- ✗ indicates that CloudHSM does not support the attribute for the specific key type.
- R indicates that the attribute value is set to read-only for the specific key type.
- S indicates that the attribute cannot be read by the GetAttributeValue as it is sensitive.
A empty cell in the Default Value column indicates that there is no specific default value assigned to the attribute.

**GenerateKeyPair**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC private</td>
<td>EC public</td>
</tr>
<tr>
<td>CKA_CLASS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✓¹</td>
<td>✓¹</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✔¹</td>
<td>✔¹</td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_WRAP_TEMPLATE</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_TRUSTED</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_WRAP_WITH_TRUSTED</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP_TEMPLATE</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>R</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Supported PKCS #11 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>R</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
| CKA_MODULUS_BITS           | ✗        | ✗             | 2
| CKA_PRIME_1                | ✗        | ✗             |
| CKA_PRIME_2                | ✗        | ✗             |
| CKA_COEFFICIENT            | ✗        | ✗             |
| CKA_EXPONENT_1             | ✗        | ✗             |
| CKA_EXPONENT_2             | ✗        | ✗             |
| CKA_PRIVATE_EXPONENT       | ✗        | ✗             |
| CKA_PUBLIC_EXPONENT        | ✗        | ✗             | 2
| CKA_EC_PARAMS              | ✗        | ✗             | 2
| CKA_EC_POINT               | ✗        | ✗             |
| CKA_VALUE                  | ✗        | ✗             |
| CKA_VALUE_LEN              | ✗        | ✗             |
| CKA_CHECK_VALUE            | R        | ✗             |

### GenerateKey

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✓ 1</td>
<td>✓ 1</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Supported PKCS #11 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✔️</td>
<td>True</td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✔️</td>
<td>True</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_WRAP_TEMPLATE</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_TRUSTED</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_WRAP_WITH_TRUSTED</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_UNWRAP_TEMPLATE</td>
<td>✔️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔️</td>
<td>True</td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔️</td>
<td>True</td>
</tr>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>R R R</td>
<td>False</td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_PRIME_1</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_PRIME_2</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_COEFFICIENT</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_EXPONENT_1</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_EXPONENT_2</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_PRIVATE_EXPONENT</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_EC_PARAMS</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_EC_POINT</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>✖️</td>
<td>False</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>✔️²</td>
<td>False</td>
</tr>
<tr>
<td>CKA_CHECK_VALUE</td>
<td>R R R</td>
<td>False</td>
</tr>
</tbody>
</table>
### CreateObject

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>EC</td>
</tr>
<tr>
<td>CKA_CLASS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP_TEMPLATE</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_TRUSTED</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_WRAP_WITH_TRUSTED</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP_TEMPLATE</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>R</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>R</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Supported PKCS #11 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_MODULUS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIME_1</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIME_2</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_COEFFICIENT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EXPONENT_1</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EXPONENT_2</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIVATE_EXPONENT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EC_PARAMS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EC_POINT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_CHECK_VALUE</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

**UnwrapKey**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

230
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✖️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✖️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✖️ ✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>R R R R R</td>
<td></td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>R R R R R</td>
<td></td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_MODULE_BITS</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_PRIME_1</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_PRIME_2</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_COEFFICIENT</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_EXPONENT_1</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_EXPONENT_2</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_PRIVATE_EXPONENT</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_EC_PARAMS</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_EC_POINT</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>✖️ ✖️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CKA_CHECK_VALUE</td>
<td>R R R R R</td>
<td></td>
</tr>
</tbody>
</table>

**DeriveKey**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>DES3</td>
<td>Generic Secret</td>
</tr>
</tbody>
</table>

231
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>✔️ ²</td>
<td>✔️ ²</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✔️ ²</td>
<td>✔️ ²</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✔️ ¹</td>
<td>✔️ ¹</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✔️ ¹</td>
<td>✔️ ¹</td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✔️ ¹</td>
<td>✔️ ¹</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PRIME_1</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PRIME_2</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_COEFFICIENT</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXPONENT_1</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXPONENT_2</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PRIVATE_EXPONENT</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
### Supported PKCS #11 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EC_PARAMS</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EC_POINT</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>✓²</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_CHECK_VALUE</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

### GetAttributeValue

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>EC private</th>
<th>EC public</th>
<th>RSA private</th>
<th>RSA public</th>
<th>AES</th>
<th>DES3</th>
<th>Generic Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_LOCAL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
<td>✓¹</td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_MODIFIABLE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_DESTROYABLE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_WRAP_TEMPLATE</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_TRUSTED</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
### Supported PKCS #11 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Key Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_WRAP_WITH_TRUSTED</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_UNWRAP_TEMPLATE</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXTRACTABLE</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_NEVER_EXTRACTABLE</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_ALWAYS_SENSITIVE</td>
<td>R R; R R R R R R R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PRIME_1</td>
<td>✗</td>
<td>✗</td>
<td>S</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>S</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_PRIME_2</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_COEFFICIENT</td>
<td>✗</td>
<td>✗</td>
<td>S</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>S</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EXPONENT_1</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EXPONENT_2</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PRIVATE_EXPONENT</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>S</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_EC_PARAMS</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_EC_POINT</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>S</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_VALUE_LEN</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CKA_CHECK_VALUE</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Annotations:**

1. This attribute is partially supported by the firmware and must be explicitly set only to the default value.

2. Mandatory attribute.

3. The CKA_SIGN_RECOVER attribute is derived from the CKA_SIGN attribute. If being set, it can only be set to the same value that is set for CKA_SIGN. If not set, it derives the default value of CKA_SIGN. Since CloudHSM only supports RSA-based recoverable signature mechanisms, this attribute is currently applicable to RSA public keys only.

4. The CKA_VERIFY_RECOVER attribute is derived from the CKA_VERIFY attribute. If being set, it can only be set to the same value that is set for CKA_VERIFY. If not set, it derives the default value of CKA_VERIFY. Since CloudHSM only supports RSA-based recoverable signature mechanisms, this attribute is currently applicable to RSA public keys only.
Modifying Attributes

Some attributes of an object can be modified after the object has been created, whereas some cannot. To modify attributes, use the `setAttribute` (p. 112) command from `cloudhsm_mgmt_util`. You can also derive a list of attributes and the constants that represent them by using the `listAttribute` (p. 106) command from `cloudhsm_mgmt_util`.

The following list displays attributes that are allowed for modification after object creation:

- `CKA_LABEL`
- `CKA_TOKEN`

  **Note**
  Modification is allowed only for changing a session key to a token key. Use the `setAttribute` (p. 190) command from `key_mgmt_util` to change the attribute value.

- `CKA_ENCRYPT`
- `CKA_DECRYPT`
- `CKA_SIGN`
- `CKA_VERIFY`
- `CKA_WRAP`
- `CKA_UNWRAP`
- `CKA_LABEL`
- `CKA_SENSITIVE`
- `CKA_DERIVE`

  **Note**
  This attribute supports key derivation. It must be `False` for all public keys and cannot be set to `True`. For secret and EC private keys, it can be set to `True` or `False`.

- `CKA_TRUSTED`

  **Note**
  This attribute can be set to `True` or `False` by Crypto Officer (CO) only.

- `CKA_WRAP_WITH_TRUSTED`

  **Note**
  This attribute can be set to `True` if the key can only be wrapped with a wrapping key that has `CKA_TRUSTED` set to `True`. Modification is allowed for changing the attribute value from `False` to `True`. Once set to `True`, you cannot modify the attribute value.

Interpreting Error Codes

Specifying in the template an attribute that is not supported by a specific key results in an error. The following table contains error codes that are generated when you violate specifications:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKR_TEMPLATE_INCONSISTENT</td>
<td>You receive this error when you specify an attribute in the attribute template, where the attribute complies with the PKCS #11 specification, but is not supported by CloudHSM.</td>
</tr>
<tr>
<td>CKR_ATTRIBUTE_TYPE_INVALID</td>
<td>You receive this error when you retrieve value for an attribute, which complies with the PKCS #11 specification, but is not supported by CloudHSM.</td>
</tr>
</tbody>
</table>
Code Samples for AWS CloudHSM Software Library for PKCS#11

The PKCS#11 code samples on GitHub show you how to accomplish basic tasks using the AWS CloudHSM software library for PKCS#11.

Sample Code Prerequisites

Before running the samples, perform the following steps to set up your environment:

- Install and configure the AWS CloudHSM software library for PKCS#11 (p. 214) and the AWS CloudHSM client package (p. 34).
- Set up a valid HSM user name and password (p. 52). Cryptographic user (CU) permissions are sufficient for these tasks. Your application uses these credentials to log in to the HSM in each example.

Code Samples

Code Samples for the AWS CloudHSM Software Library for PKCS#11 are available on GitHub. This repository includes examples on how to do common operations using PKCS#11 including encryption, decryption, signing and verifying.

AWS CloudHSM Dynamic Engine for OpenSSL

The AWS CloudHSM dynamic engine for OpenSSL is an OpenSSL dynamic engine that supports the OpenSSL command line interface and EVP API operations. The dynamic engine allows applications that are integrated with OpenSSL, such as the NGINX and Apache web servers, to offload their cryptographic processing to the HSMs in your AWS CloudHSM cluster. The engine supports the following key types and ciphers:

- RSA key generation for 2048, 3072, and 4096-bit keys.
- RSA sign/verify.
- RSA encrypt/decrypt.
- Random number generation that is cryptographically secure and FIPS-validated.

For more information, see the following topic.

Topics

- Install and Use the AWS CloudHSM Dynamic Engine for OpenSSL (p. 237)
Install and Use the AWS CloudHSM Dynamic Engine for OpenSSL

Before you can use the AWS CloudHSM dynamic engine for OpenSSL, you need the AWS CloudHSM client. The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster, and the OpenSSL engine communicates locally with the client. If you haven't installed and configured the AWS CloudHSM client package, do that now by following the steps at Install the Client (Linux) (p. 34). After you install and configure the client, use the following command to start it.

The AWS CloudHSM dynamic engine for OpenSSL is supported only on Linux and compatible operating systems.

Amazon Linux

```
# sudo start cloudhsm-client
```

Amazon Linux 2

```
# sudo service cloudhsm-client start
```

CentOS 6

```
# sudo start cloudhsm-client
```

CentOS 7

```
# sudo service cloudhsm-client start
```

RHEL 6

```
# sudo start cloudhsm-client
```

RHEL 7

```
# sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
# sudo service cloudhsm-client start
```

Topics

- Install and Configure the OpenSSL Dynamic Engine (p. 237)
- Use the OpenSSL Dynamic Engine (p. 239)

Install and Configure the OpenSSL Dynamic Engine

Complete the following steps to install (or update) and configure the AWS CloudHSM dynamic engine for OpenSSL. It is supported only on Linux and compatible operating systems.
To install (or update) and configure the OpenSSL engine

1. Use the following commands to download and install the OpenSSL engine.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-
     client-dyn-latest.el6.x86_64.rpm

$ sudo yum install -y ./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 -y ./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 -y ./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 -y ./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 -y ./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 -y ./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
```

After you complete the preceding step, you can find the OpenSSL engine at /opt/cloudhsm/lib/libcloudhsm_openssl.so.

3. Use the following command to set an environment variable named n3fips_password that contains the credentials of a crypto user (CU).

```bash
$ export n3fips_password=<HSM user name>:<password>
```

**Use the OpenSSL Dynamic Engine**

To use the AWS CloudHSM dynamic engine for OpenSSL from the OpenSSL command line, use the -engine option to specify the OpenSSL dynamic engine named cloudhsm. For example:

```bash
$ openssl s_server -cert server.crt -key server.key -engine cloudhsm
```

To use the AWS CloudHSM dynamic engine for OpenSSL from an OpenSSL-integrated application, ensure that your application uses the OpenSSL dynamic engine named cloudhsm. The shared library for the dynamic engine is located at /opt/cloudhsm/lib/libcloudhsm_openssl.so.

**AWS CloudHSM Software Library for Java**

The AWS CloudHSM software library for Java is a provider implementation for the Sun Java JCE (Java Cryptography Extension) provider framework. It includes implementations for interfaces and engine classes in the JCA (Java Cryptography Architecture) standard. For more information about installing and using the Java library, see the following topics.

**Topics**

- Install and Use the AWS CloudHSM Software Library for Java (p. 239)
- Supported Mechanisms (p. 244)
- Code Samples for the AWS CloudHSM Software Library for Java (p. 247)
- Using CloudHSM KeyStore Java Class (p. 248)
- Using AWS CloudHSM Key Store with Third-Party Tools (p. 250)

**Install and Use the AWS CloudHSM Software Library for Java**

Before you can use the AWS CloudHSM software library for Java, you need the AWS CloudHSM client.

The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster. The Java library communicates locally with the client. If you haven't installed and configured the AWS CloudHSM client package, do that now by following the steps at Install the Client (Linux) (p. 34). After you install and configure the client, use the following command to start it.

Note that the AWS CloudHSM software library for Java is supported only on Linux and compatible operating systems.
Installing the Java Library

Use the following commands to download and install or update the AWS CloudHSM Java library. This library is supported only on Linux and compatible operating systems.

Amazon Linux

```bash
# wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-jce-latest.el6.x86_64.rpm

# sudo yum install -y ./cloudhsm-client-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

```bash
# wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm

# sudo yum install -y ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 6

```bash
# sudo start cloudhsm-client
```

CentOS 7

```bash
# sudo service cloudhsm-client start
```

RHEL 6

```bash
# sudo start cloudhsm-client
```

RHEL 7

```bash
# sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```bash
# sudo service cloudhsm-client start
```

Topics

- Installing the Java Library (p. 240)
- Validating the Installation (p. 242)
- Providing Credentials to the Java Library (p. 243)
- Key Management Basics in the Java Library (p. 244)
After you run the preceding commands, you can find the following Java library files:

- `/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-2.8.jar`
- `/opt/cloudhsm/java/log4j-core-2.8.jar`
- `/opt/cloudhsm/lib/libcaviumjca.so`
Validating the Installation

Perform basic operations on the HSM to validate the installation.

To validate Java library installation

1. (Optional) If you don't already have Java installed in your environment, use the following command to install it.

   **Linux (and compatible libraries)**
   
   ```bash
   $ sudo yum install -y java-1.8.0-openjdk
   ```

   **Ubuntu**
   
   ```bash
   $ sudo apt-get install openjdk-8-jre
   ```

2. Use the following commands to set the necessary environment variables. Replace `<HSM user name>` and `<password>` with the credentials of a crypto user (CU).

   ```bash
   $ export LD_LIBRARY_PATH=/opt/cloudhsm/lib
   $ export HSM_PARTITION=PARTITION_1
   $ export HSM_USER=<HSM user name>
   $ export HSM_PASSWORD=<password>
   ```

3. Use the following command to run the basic functionality test. If successful, the command's output should be similar to the one that follows.

   ```bash
   $ 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.
   (TestBasicFunctionality.java:92) - Logging out.
   
   Time: 0.205
Providing Credentials to the Java Library

HSMs need to authenticate your Java application before the application can use them. Each application can use one session. HSMs authenticate a session by using either explicit login or implicit login method.

Explicit login – This method lets you provide CloudHSM credentials directly in the application. It uses the LoginManager.login() method, where you pass the CU user name, password, and the HSM partition ID. For more information about using the explicit login method, see the Login to an HSM code example.

Implicit login – This method lets you set CloudHSM credentials either in a new property file, system properties, or as environment variables.

- **New property file** – Create a new file named HsmCredentials.properties and add it to your application's CLASSPATH. The file should contain the following:

```
HSM_PARTITION = PARTITION_1
HSM_USER = <HSM user name>
HSM_PASSWORD = <password>
```

- **System properties** – Set credentials through system properties when running your application. The following examples show two different ways that you can do this:

```
$ java -D HSM_PARTITION=PARTITION_1 -D HSM_USER=<HSM user name> -D HSM_PASSWORD=<password>
```

```java
System.setProperty("HSM_PARTITION","PARTITION_1");
System.setProperty("HSM_USER","<HSM user name>");
System.setProperty("HSM_PASSWORD","<password>");
```

- **Environment variables** – Set credentials as environment variables.

```
$ export HSM_PARTITION=PARTITION_1
$ export HSM_USER=<HSM user name>
$ export HSM_PASSWORD=<password>
```

Credentials might not be available if the application does not provide them or if you attempt an operation before the HSM authenticates session. In those cases, the CloudHSM software library for Java searches for the credentials in the following order:

1. HsmCredentials.properties
2. System properties
3. Environment variables

**Error handling**

The error handling is easier with the explicit login than the implicit login method. When you use the LoginManager class, you have more control over how your application deals with failures. The implicit login method makes error handling difficult to understand when the credentials are invalid or the HSMs are having problems in authenticating session.
Key Management Basics in the Java Library

The basics on key management in the Java library involve importing keys, exporting keys, loading keys by handle, or deleting keys. For more information on managing keys, see the Manage keys code example.

You can also find more Java library code examples at Java Samples (p. 247).

Supported Mechanisms

For information about the Java Cryptography Architecture (JCA) interfaces and engine classes supported by AWS CloudHSM, see the following topics.

Topics

- Supported Keys (p. 244)
- Supported Ciphers (p. 244)
- Supported Digests (p. 246)
- Supported Hash-Based Message Authentication Code (HMAC) Algorithms (p. 246)
- Supported Sign/Verify Mechanisms (p. 247)

Supported Keys

The AWS CloudHSM software library for Java enables you to generate the following key types.

- **RSA** – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.
- **AES** – 128, 192, and 256-bit AES keys.
- **ECC key pairs** for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1 (Blockchain).

In addition to standard parameters, we support the following parameters for each key that is generated.

- **Label**: A key label that you can use to search for keys.
- **isExtractable**: Indicates whether the key can be exported from the HSM.
- **isPersistent**: Indicates whether the key remains on the HSM when the current session ends.

Supported Ciphers

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mode</th>
<th>Padding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>CBC</td>
<td>AES/CBC/NoPadding&lt;br&gt;AES/CBC/PKCS5Padding</td>
<td>Implements Cipher.ENCRYPT_MODE, Cipher.DECRYPT_MODE, Cipher.WRAP_MODE, and Cipher.UNWRAP_MODE.</td>
</tr>
<tr>
<td>AES</td>
<td>ECB</td>
<td>AES/ECB/NoPadding&lt;br&gt;AES/ECB/PKCS5Padding</td>
<td>Implements Cipher.ENCRYPT_MODE, Cipher.DECRYPT_MODE, Cipher.WRAP_MODE, and Cipher.UNWRAP_MODE.</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Mode</td>
<td>Padding</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cipher.UNWRAP_MODE. Use Transformation AES.</td>
</tr>
<tr>
<td>AES</td>
<td>CTR</td>
<td>AES/CTR/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
<tr>
<td>AES</td>
<td>GCM</td>
<td>AES/GCM/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE. When performing AES-GCM encryption, the HSM ignores the initialization vector (IV) in the request and uses an IV that it generates. When the operation completes, you must call Cipher.getIV() to get the IV.</td>
</tr>
<tr>
<td>DESede (Triple DES)</td>
<td>CBC</td>
<td>DESede/CBC/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE. The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.</td>
</tr>
<tr>
<td>DESede (Triple DES)</td>
<td>ECB</td>
<td>DESede/ECB/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE. The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.</td>
</tr>
<tr>
<td>RSA</td>
<td>ECB</td>
<td>RSA/ECB/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
</tbody>
</table>
## Supported Mechanisms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mode</th>
<th>Padding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>ECB</td>
<td>RSA/ECB/OAEPPadding</td>
<td>Implements Cipher.ENCRYPT_MODE, Cipher.DECRYPT_MODE, Cipher.WRAP_MODE, Cipher.UNWRAP_MODE. OAEPPadding is OAEP with the SHA-1 padding type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-1ANDMGF1Padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-224ANDMGF1Padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-256ANDMGF1Padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-384ANDMGF1Padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-512ANDMGF1Padding</td>
<td></td>
</tr>
<tr>
<td>RSAAESWrap</td>
<td>ECB</td>
<td>OAEPPADDING</td>
<td>Implements Cipher.WRAP_Mode and Cipher.UNWRAP_MODE.</td>
</tr>
</tbody>
</table>

### Supported Digests

The AWS CloudHSM software library for Java supports the following message digests.

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

**Note**

Data under 16 KB in length are hashed on the HSM, while larger data are hashed locally in software.

### Supported Hash-Based Message Authentication Code (HMAC) Algorithms

The AWS CloudHSM software library for Java supports the following HMAC algorithms.

- HmacSHA1
- HmacSHA224
- HmacSHA256
- HmacSHA384
- HmacSHA512
Supported Sign/Verify Mechanisms

The AWS CloudHSM software library for Java supports the following types of signature and verification.

**RSA Signature Types**

- NONEwithRSA
- SHA1withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA
- SHA1withRSA/PSS
- SHA224withRSA/PSS
- SHA256withRSA/PSS
- SHA384withRSA/PSS
- SHA512withRSA/PSS

**ECDSA Signature Types**

- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA

Code Samples for the AWS CloudHSM Software Library for Java

**Sample Code Prerequisites**

Before running the samples, use the following steps to set up your environment:

- Install and configure the AWS CloudHSM software library for Java (p. 239) and the AWS CloudHSM client package (p. 34).
- Set up a valid HSM user name and password (p. 52). Cryptographic user (CU) permissions are sufficient for these tasks. Your application uses these credentials to log in to the HSM in each example.
- Decide how to specify the Cavium provider.

**Code Samples**

The following Java library code samples show you how to use the AWS CloudHSM software library for Java (p. 239) to perform basic tasks in AWS CloudHSM. More code samples are available on GitHub.

- Login to an HSM
- Manage keys
- Generate an AES key
• Encrypt and decrypt data with AES GCM
• Wrap and unwrap keys with AES
• Wrap and unwrap keys with RSA
• Enumerate through the KeyStore
• Sign messages in a multi-threaded sample
• Use JCE instead of OpenSSL to perform ECDH key derivation

Using CloudHSM KeyStore Java Class

The AWS CloudHSM KeyStore class provides a special-purpose PKCS12 key store that allows access to AWS CloudHSM keys through applications such as keytool and jarsigner. This key store can store certificates along with your key data and correlate them to key data stored on AWS CloudHSM.

Note
Because certificates are public information, and to maximize storage capacity for cryptographic keys, AWS CloudHSM does not support storing certificates on HSMs.

The AWS CloudHSM KeyStore class implements the KeyStore Service Provider Interface (SPI) of the Java Cryptography Extension (JCE). For more information about using KeyStore, see Class KeyStore.

Choosing the Appropriate Key Store

The AWS CloudHSM Java SDK comes with a default pass-through, read-only key store that passes all transactions to the HSM. This default key store is distinct from the special-purpose AWS CloudHSM KeyStore. In most situations, you will obtain better runtime performance and throughput by using the default. You should only use the AWS CloudHSM KeyStore for applications where you require support for certificates and certificate-based operations in addition to offloading key operations to the HSM.

Although both key stores use the Cavium JCE provider for operations, they are independent entities and do not exchange information with each other.

Load the default key store for your Java application as follows:

```java
KeyStore ks = KeyStore.getInstance("Cavium");
```

Load the special-purpose CloudHSM KeyStore as follows:

```java
KeyStore ks = KeyStore.getInstance("CloudHSM")
```

Initializing AWS CloudHSM KeyStore

Log into the AWS CloudHSM KeyStore the same way that you log into the JCE provider for AWS CloudHSM. You can use either environment variables or the system property file, and you should log in before you start using the CloudHSM KeyStore. For an example of logging into an HSM using the JCE, see Login to an HSM.

If desired, you can specify a password to encrypt the local PKCS12 file which holds key store data. When you create the AWS CloudHSM Keystore, you set the password and provide it when using the load, set and get methods.

Instantiate a new CloudHSM KeyStore object as follows:

```java
ks.load(null, null);
```

Write keystore data to a file using the store method. From that point on, you can load the existing keystore using the load method with the source file and password as follows:
Using CloudHSM KeyStore

A CloudHSM KeyStore object is generally used through a third-party application such as jarsigner or keytool. You can also access the object directly with code.

AWS CloudHSM KeyStore complies with the JCE Class KeyStore specification and provides the following functions.

- **load**

  Loads the key store from the given input stream. If a password was set when saving the key store, this same password must be provided for the load to succeed. Set both parameters to null to initialize a new empty key store.

  ```java
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  ks.load(inputStream, password);
  ```

- **aliases**

  Returns an enumeration of the alias names of all entries in the given key store instance. Results include objects stored locally in the PKCS12 file and objects resident on the HSM.

  **Sample code:**

  ```java
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  for(Enumeration<String> entry = ks.aliases(); entry.hasMoreElements();)
  {
    String label = entry.nextElement();
    System.out.println(label);
  }
  ```

- **ContainsAlias**

  Returns true if the key store has access to at least one object with the specified alias. The key store checks objects stored locally in the PKCS12 file and objects resident on the HSM.

- **DeleteEntry**

  Deletes a certificate entry from the local PKCS12 file. Deleting key data stored in an HSM is not supported using the AWS CloudHSM KeyStore. You can delete keys with CloudHSM's key_mgmt_util tool.

- **GetCertificate**

  Returns the certificate associated with an alias if available. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

  ```java
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  Certificate cert = ks.getCertificate(alias)
  ```

- **GetCertificateAlias**

  Returns the name (alias) of the first key store entry whose data matches the given certificate.

  ```java
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  String alias = ks.getCertificateAlias(cert)
  ```

- **GetCertificateChain**

  ```java
  ```
Returns the certificate chain associated with the given alias. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

- **GetCreationDate**

  Returns the creation date of the entry identified by the given alias. If a creation date is not available, the function returns the date on which the certificate became valid.

- **GetKey**

  GetKey is passed to the HSM and returns a key object corresponding to the given label. As `getKey` directly queries the HSM, it can be used for any key on the HSM regardless of whether it was generated by the KeyStore.

  ```java
  Key key = ks.getKey(keyLabel, null);
  ```

- **IsCertificateEntry**

  Checks if the entry with the given alias represents a certificate entry.

- **IsKeyEntry**

  Checks if the entry with the given alias represents a key entry. The action searches both the PKCS12 file and the HSM for the alias.

- **SetCertificateEntry**

  Assigns the given certificate to the given alias. If the given alias is already being used to identify a key or certificate, a `KeyStoreException` is thrown. You can use JCE code to get the key object and then use the KeyStore `setKeyEntry` method to associate the certificate to the key.

- **SetKeyEntry with byte[] key**

  Assigns the given byte array key to the given alias by storing it inside HSM as a generic key with the given alias.

- **SetKeyEntry with Key object**

  Assigns the given key to the given alias and stores it inside the HSM. If the `Key` object is not of type `CaviumKey`, the key is imported into the HSM as an extractable session key.

  If the `Key` object is of type `PrivateKey`, it must be accompanied by a corresponding certificate chain.

  If the alias already exists, the `setKeyEntry` call throws a `KeyStoreException` and prevents the key from being overwritten. If the key must be overwritten, use KMU or JCE for that purpose.

- **EngineSize**

  Returns the number of entries in the keystore.

- **Store**

  Stores the key store to the given output stream as a PKCS12 file and secures it with the given password. In addition, it persists all loaded keys (which are set using `setKey` calls).

### Using AWS CloudHSM Key Store with Third-Party Tools

AWS CloudHSM key store is a special-purpose JCE key store that utilizes certificates associated with keys on your HSM through third-party tools such as `keytool` and `jarsigner`. AWS CloudHSM does not store certificates on the HSM, as certificates are public, non-confidential data. The AWS CloudHSM key store stores the certificates in a local file and maps the certificates to corresponding keys on your HSM.
When you use the AWS CloudHSM key store to generate new keys, no entries are generated in the local key store file – the keys are created on the HSM. Similarly, when you use the AWS CloudHSM key store to search for keys, the search is passed on to the HSM. When you store certificates in the AWS CloudHSM key store, the provider verifies that a key pair with the corresponding alias exists on the HSM, and then associates the certificate provided with the corresponding key pair.

**Topics**
- Prerequisites (p. 251)
  - Using AWS CloudHSM Key Store with Keytool (p. 252)
  - Using AWS CloudHSM Key Store with Jarsigner (p. 254)
  - Known Issues (p. 255)
  - Registering Pre-existing Keys with AWS CloudHSM Key Store (p. 255)

**Prerequisites**
To use the AWS CloudHSM key store, you must first initialize and configure the AWS CloudHSM JCE SDK.

**Step 1: Install the JCE**
To install the JCE, including the AWS CloudHSM client prerequisites, follow the steps for installing the Java library (p. 239).

**Step 2: Add HSM login credentials to environment variables**
Set up environment variables to contain your HSM login credentials.

```bash
export HSM_PARTITION=PARTITION_1
export HSM_USER=<HSM user name>
export HSM_PASSWORD=<HSM password>
```

**Note**
The CloudHSM JCE offers various login options. To use the AWS CloudHSM key store with third-party applications, you must use implicit login with environment variables. If you want to use explicit login through application code, you must build your own application using the AWS CloudHSM key store. For additional information, see the article on Using AWS CloudHSM Key Store (p. 248).

**Step 3: Registering the JCE provider**
To register the JCE provider, in the Java CloudProvider configuration.

1. Open the java.security configuration file in your Java installation, for editing.
2. In the java.security configuration file, add `com.cavium.provider.CaviumProvider` as the last provider. For example, if there are nine providers in the java.security file, add the following provider as the last provider in the section. Adding the Cavium provider as a higher priority may negatively impact your system's performance.

   ```
   security.provider.10=com.cavium.provider.CaviumProvider
   ```

   **Note**
   Power users may be accustomed to specifying `-providerName`, `-providerclass`, and `-providerpath` command line options when using keytool, instead of updating the security configuration file. If you attempt to specify command line options when generating keys with AWS CloudHSM key store, it will cause errors.
Using AWS CloudHSM Key Store with Keytool

Keytool is a popular command line utility for common key and certificate tasks on Linux systems. A complete tutorial on keytool is out of scope for AWS CloudHSM documentation. This article explains the specific parameters you should use with various keytool functions when utilizing AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

When using keytool with the AWS CloudHSM key store, specify the following arguments to any keytool command:

```
-storetype CLOUDHSM \
-J-classpath '-J/opt/cloudhsm/java/*' \
-J-Djava.library.path=/opt/cloudhsm/lib
```

If you want to create a new key store file using AWS CloudHSM key store, see Using AWS CloudHSM Key Store. To use an existing key store, specify its name (including path) using the –keystore argument to keytool. If you specify a non-existent key store file in a keytool command, the AWS CloudHSM key store creates a new key store file.

Create New Keys with Keytool

You can use keytool to generate any type of key supported by AWS CloudHSM's JCE SDK. See a full list of keys and lengths in the Supported Keys article in the Java Library.

**Important**

A key generated through keytool is generated in software, and then imported into AWS CloudHSM as an extractable, persistent key.

Instructions for creating non-extractable keys directly on the HSM, and then using them with keytool or Jarsigner, are shown in the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store (p. 255). We strongly recommend generating non-exportable keys outside of keytool, and then importing corresponding certificates to the key store. If you use extractable RSA or EC keys through keytool and jarsigner, the providers export keys from the AWS CloudHSM and then use the key locally for signing operations.

If you have multiple client instances connected to your CloudHSM cluster, be aware that importing a certificate on one client instance's key store won't automatically make the certificates available on other client instances. To register the key and associated certificates on each client instance you need to run a Java application as described in Generate a CSR using Keytool (p. 253). Alternatively, you can make the necessary changes on one client and copy the resulting key store file to every other client instance.

**Example 1:** To generate a symmetric AES-256 key with label, "my_secret" and save it in a key store file named, "my_keystore.store", in the working directory.

```
keytool -genseckey -alias my_secret -keyalg aes \
-keystore 256 -keystore my_keystore.store \
-storetype CLOUDHSM -J-classpath '-J/opt/cloudhsm/java/*' \
-J-Djava.library.path=/opt/cloudhsm/lib
```

**Example 2:** To generate an RSA 2048 key pair with label "my_rsa_key_pair" and save it in a key store file named, "my_keystore.store" in the working directory.

```
keytool -genkeypair -alias my_rsa_key_pair \
-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/
```
Example 3: To generate a p256 ED key with label "my_ec_key_pair" and save it in a key store file named, "my_keystore.store" in the working directory.

```
keytool -genkeypair -alias my_ec_key_pair \
    -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/
```

You can find a list of supported signature algorithms in the Java library.

Delete a Key using Keytool

The AWS CloudHSM key store doesn't support deleting keys. To delete key, you must use the deleteKey function of AWS CloudHSM's command line tool, deleteKey (p. 130).

Generate a CSR using Keytool

You receive the greatest flexibility in generating a certificate signing request (CSR) if you use the AWS CloudHSM Dynamic Engine for OpenSSL (p. 236). The following command uses keytool to generate a CSR for a key pair with the alias, my_key_pair.

```
keytool -certreq -alias my_key_pair \
    -file my_csr.csr \
    -keystore my_keystore.store \
    -storetype CLOUDHSM \
    -J-classpath '-J/opt/cloudhsm/java/*' \
    -J-Djava.library.path=/opt/cloudhsm/lib/
```

Note

To use a key pair from keytool, that key pair must have an entry in the specified key store file. If you want to use a key pair that was generated outside of keytool, you must import the key and certificate metadata into the key store. For instructions on importing the keystore data see Importing Intermediate and root certificates into AWS CloudHSM Key Store using Keytool (p. 253).

Using Keytool to import intermediate and root certificates into AWS CloudHSM Key Store

To import a CA certificate you must enable verification of a full certificate chain on a newly imported certificate. The following command shows an example.

```
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/
```

If you connect multiple client instances to your AWS CloudHSM cluster, importing a certificate on one client instance's key store won't automatically make the certificate available on other client instances. You must import the certificate on each client instance.

Using Keytool to Delete Certificates from AWS CloudHSM Key Store

The following command shows an example of how to delete a certificate from a Java keytool key store.

```
keytool -delete -alias mydomain -keystore \
```
If you connect multiple client instances to your AWS CloudHSM cluster, deleting a certificate on one client instance's key store won't automatically remove the certificate from other client instances. You must delete the certificate on each client instance.

**Importing a Working Certificate into AWS CloudHSM Key Store using Keytool**

Once a certificate signing request (CSR) is signed, you can import it into the AWS CloudHSM key store and associate it with the appropriate key pair. The following command provides an example.

```bash
keytool -importcert -noprompt -alias my_key_pair \\
-file my_certificate.crt \\
-keystore my_keystore.store \\
-storetype CLOUDHSM \\
-J-classpath '-J/opt/cloudhsm/java/*' \\
-J-Djava.library.path=/opt/cloudhsm/lib/
```

The alias should be a key pair with an associated certificate in the key store. If the key is generated outside of keytool, or is generated on a different client instance, you must first import the key and certificate metadata into the key store. For instructions on importing the certificate metadata, see the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store (p. 255).

The certificate chain must be verifiable. If you can't verify the certificate, you might need to import the signing (certificate authority) certificate into the key store so the chain can be verified.

**Exporting a certificate using Keytool**

The following example generates a certificate in binary X.509 format. To export a human readable certificate, add `-rfc` to the `exportcert` command.

```bash
keytool -exportcert -alias my_key_pair \\
-file my_exported_certificate.crt \\
-keystore my_keystore.store \\
-storetype CLOUDHSM \\
-J-classpath '-J/opt/cloudhsm/java/*' \\
-J-Djava.library.path=/opt/cloudhsm/lib/
```

**Using AWS CloudHSM Key Store with Jarsigner**

Jarsigner is a popular command line utility for signing JAR files using a key securely stored on a HSM. A complete tutorial on Jarsigner is out of scope for the AWS CloudHSM documentation. This section explains the Jarsigner parameters you should use to sign and verify signatures with AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

**Setting up keys and certificates**

Before you can sign JAR files with Jarsigner, make sure you have set up or completed the following steps:

1. Follow the guidance in the AWS CloudHSM Key store prerequisites (p. 251).
2. Set up your signing keys and the associated certificates and certificate chain which should be stored in the AWS CloudHSM key store of the current server or client instance. Create the keys on the AWS CloudHSM and then import associated metadata into your AWS CloudHSM key store. Use the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store (p. 255) to import metadata into the key store. If you want to use keytool to set up the keys and certificates, see Create New Keys with Keytool (p. 252). If you use multiple client instances to sign your JARs, create the key
and import the certificate chain. Then copy the resulting key store file to each client instance. If you frequently generate new keys, you may find it easier to individually import certificates to each client instance.

3. The entire certificate chain should be verifiable. For the certificate chain to be verifiable, you may need to add the CA certificate and intermediate certificates to the AWS CloudHSM key store. See the code snippet in Sign a JAR file using AWS CloudHSM and Jarsigner (p. 255) for instruction on using Java code to verify the certificate chain. If you prefer, you can use keytool to import certificates. For instructions on using keytool, see Using Keytool to import intermediate and root certificates into AWS CloudHSM Key Store (p. 253).

Sign a JAR file using AWS CloudHSM and Jarsigner

Use the following command to sign a JAR file:

```bash
classpath='-J/opt/cloudhsm/java\*/usr/lib/jvm/java-1.8.0/lib/tools.jar' \n-keystore my_keystore.store \n-sigalg sha512withrsa \n-storetype CloudHSM \n-signedjar signthisclass_signed.jar \n-verify \
-keystore my_keystore.store \n-sigalg sha512withrsa \n-storetype CloudHSM \n-classpath '-J/opt/cloudhsm/java\*/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
my_key_pair
```

Use the following command to verify a signed JAR:

```bash
classpath='-J/opt/cloudhsm/java\*/usr/lib/jvm/java-1.8.0/lib/tools.jar' \n-keystore my_keystore.store \n-sigalg sha512withrsa \n-storetype CloudHSM \n-signedjar signthisclass_signed.jar \n-my_key_pair
```

Known Issues

The following list provides the current list of known issues.

- When generating keys using keytool, the first provider in provider configuration cannot be CaviumProvider.
- When generating keys using keytool, the first (supported) provider in the security configuration file is used to generate the key. This is generally a software provider. The generated key is then given an alias and imported into the AWS CloudHSM HSM as a persistent (token) key during the key addition process.
- When using keytool with AWS CloudHSM key store, do not specify -providerName, -providerclass, or -providerpath options on the command line. Specify these options in the security provider file as described in the Key store prerequisites (p. 251).
- When using non-extractable EC keys through keytool and Jarsigner, the SunEC provider needs to be removed/disabled from the list of providers in the java.security file. If you use extractable EC keys through keytool and Jarsigner, the providers export key bits from the AWS CloudHSM HSM and use the key locally for signing operations. We do not recommend you use exportable keys with keytool or Jarsigner.

Registering Pre-existing Keys with AWS CloudHSM Key Store

For maximum security and flexibility in attributes and labeling, we recommend you generate your signing keys using key_mgmt_util (p. 55). You can also use a Java application to generate the key in AWS CloudHSM.
The following section provides a code sample that demonstrates how to generate a new key pair on the HSM and register it using existing keys imported to the AWS CloudHSM key store. The imported keys are available for use with third-party tools such as keytool and Jarsigner.

To use a pre-existing key, modify the code sample to look up a key by label instead of generating a new key. Sample code for looking up a key by label is available in the KeyUtilitiesRunner.java sample on GitHub.

**Important**
Registering a key stored on AWS CloudHSM with a local key store does not export the key. When the key is registered, the key store registers the key's alias (or label) and correlates locally store certificate objects with a key pair on the AWS CloudHSM. As long as the key pair is created as non-exportable, the key bits won't leave the HSM.

```java
// 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.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" +
      "	--help			Display this message.%n" +
      "	--store <filename>		Path of the keystore.%n" +
      "	--password <password>		Password for the keystore (not your CU password).%n" +
      "$" +
      "	--label <label>		Label to store the key and certificate under.%n" +
      "$--list		List 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 (arg) {
          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 aliases easier to understand.
//
// public static KeyPair generateRSAKeyPair(int keySizeInBits, String publicKeyLabel, 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"), publicKeyLabel, 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("0x2A,0x86,0x48,0x86,0xF7,0x0D," +
(byte) 0x01,0x01,0x0D\n),
(byte) 0x01, (byte) 0x01, (byte) 0x0D );
    String sigAlgoName = "SHA512WithRSA";
    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(
    Encoder.encodeOid(COUNTRY_NAME_OID, "<Country>") +,
    Encoder.encodeOid(STATE_OR_PROVINCE_NAME_OID, "<State>") +,
    Encoder.encodeOid(LOCALITY_NAME_OID, "<City>") +,
    Encoder.encodeOid(ORGANIZATION_NAME_OID, "<Organization>") +,
    Encoder.encodeOid(ORGANIZATION_UNIT_OID, "<Unit>") +,
    Encoder.encodeOid(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");
    } catch (KeyStoreException ex) {
        System.err.println("Failed to load KeyStore");
    } catch (CertificateException ex) {
        System.err.println("Failed to load Certificate");
    } catch (IOException ex) {
        System.err.println("Failed to read from file");
    } catch (Exception ex) {
        System.err.println("Unknown error");
    }
KSP and CNG Providers for Windows

Cryptography API: Next Generation (CNG) is a cryptographic API specific to the Microsoft Windows operating system. CNG enables developers to use cryptographic techniques to secure Windows-based applications. At a high level, CNG provides the following functionality.

- **Cryptographic Primitives** - enable you to perform fundamental cryptographic operations.
- **Key Import and Export** - enables you to import and export symmetric and asymmetric keys.
- **Data Protection API (CNG DPAPI)** - enables you to easily encrypt and decrypt data.
- **Key Storage and Retrieval** - enables you to securely store and isolate the private key of an asymmetric key pair.

Key storage providers (KSPs) enable key storage and retrieval. For example, if you add the Microsoft Active Directory Certificate Services (AD CS) role to your Windows server and you choose to create a new private key for your certificate authority (CA), you can choose the KSP that will manage key storage. The Windows CloudHSM client includes KSPs created by Cavium specifically for AWS CloudHSM. When you configure the AD CS role, you can choose a Cavium KSP. For more information, see Create Windows Server CA (p. 300). The Windows CloudHSM client also installs a Cavium CNG provider.

**Topics**
- Install the KSP and CNG Providers for Windows (p. 261)
- Windows AWS CloudHSM Prerequisites (p. 263)
- Associate an AWS CloudHSM Key with a Certificate (p. 264)
- Code Sample for Cavium CNG Provider (p. 265)

Install the KSP and CNG Providers for Windows

The Cavium KSP and CNG providers are installed when you install the Windows AWS CloudHSM client. You can install the client by following the steps at Install the Client (Windows) (p. 36). You can choose the Cavium KSP when you add the AD CS role to your Windows Server. See Create Windows Server CA (p. 300).

Configure and Run the Windows AWS CloudHSM Client

To start the Windows CloudHSM client, you must first satisfy the Prerequisites (p. 263). Then, update the configuration files that the providers use and start the client by completing the steps below. You need to do these steps the first time you use the KSP and CNG providers and after you add or remove HSMs in your cluster. This way, AWS CloudHSM is able to synchronize data and maintain consistency across all HSMs in the cluster.
Step 1: Stop the AWS CloudHSM Client

Before you update the configuration files that the providers use, stop the AWS CloudHSM client. If the client is already stopped, running the stop command has no effect.

- For Windows client 1.1.2+:

  `C:\Program Files\Amazon\CloudHSM> net.exe stop AWSCloudHSMClient`

- For Windows clients 1.1.1 and older:

  Use Ctrl+C in the command window where you started the AWS CloudHSM client.

Step 2: Update the AWS CloudHSM Configuration Files

This step uses the `-a` parameter of the Configure tool (p. 207) to add the elastic network interface (ENI) IP address of one of the HSMs in the cluster to the configuration file.

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

To get the ENI IP address of an HSM in your cluster, navigate to the AWS CloudHSM console, choose `clusters`, and select the desired cluster. You can also use the `DescribeClusters` operation, the `describe-clusters` command, or the `Get-HSM2Cluster` PowerShell cmdlet. Type only one ENI IP address. It does not matter which ENI IP address you use.

Step 3: Start the AWS CloudHSM Client

Next, start or restart the AWS CloudHSM client. When the AWS CloudHSM client starts, it uses the ENI IP address in its configuration file to query the cluster. Then it adds the ENI IP addresses of all HSMs in the cluster to the cluster information file.

- For Windows client 1.1.2+:

  `C:\Program Files\Amazon\CloudHSM> net.exe start AWSCloudHSMClient`

- For Windows clients 1.1.1 and older:

  `C:\Program Files\Amazon\CloudHSM> start "cloudhsm_client" cloudhsm_client.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg`

Checking the KSP and CNG Providers

You can use either of the following commands to determine which providers are installed on your system. The commands list the registered KSP and CNG providers. The AWS CloudHSM client does not need to be running.

`C:\Program Files\Amazon\CloudHSM>ksp_config.exe -enum`

`C:\Program Files\Amazon\CloudHSM>cng_config.exe -enum`

Verify that the Cavium KSP and CNG providers are installed on your Windows Server EC2 instance. If the CNG provider is missing, run the following command.
If the Cavium KSP is missing, run the following command.

**C:\Program Files\Amazon\CloudHSM>ksp_config.exe -register**

### Windows AWS CloudHSM Prerequisites

Before you can start the Windows AWS CloudHSM client and use the KSP and CNG providers, you must set the login credentials for the HSM on your system. You can set credentials through either Windows Credentials Manager or system environment variable. We recommend you use Windows Credential Manager for storing credentials. This option is available with AWS CloudHSM client version 2.0.4 and later. Using environment variable is easier to set up, but less secure than using Windows Credential Manager.

#### Windows Credential Manager

You can use either the `set_cloudhsm_credentials` utility or the Windows Credentials Manager interface.

- **Using the `set_cloudhsm_credentials` utility:**

  The `set_cloudhsm_credentials` utility is included in your Windows installer. You can use this utility to conveniently pass HSM login credentials to Windows Credential Manager. If you want to compile this utility from source, you can use the Python code that is included in the installer.

  1. Go to the `C:\Program Files\Amazon\CloudHSM\tools\` folder.
  2. Run the `set_cloudhsm_credentials.exe` file with the CU username and password parameters.


```bash
set_cloudhsm_credentials.exe --username <cu-user> --password <cu-pwd>
```

- **Using the Credential Manager interface:**

  You can use the Credential Manager interface to manually manage your credentials.

  1. To open Credential Manager, type `credential manager` in the search box on the taskbar and select **Credential Manager**.
  2. Select **Windows Credentials** to manage Windows credentials.
  3. Select **Add a generic credential** and fill out the details as follows:

     - In **Internet or Network Address**, enter the target name as `cloudhsm_client`.
     - In **Username** and **Password** enter the CU credentials.
     - Click **OK**.

#### System Environment Variables

You can set system environment variables that identify an HSM and a crypto user (p. 11) (CU) for your Windows application. You can use the `setx` command to set system environment variables, or set permanent system environment variables programmatically or in the **Advanced** tab of the Windows System Properties Control Panel.

**Warning**

When you set credentials through system environment variables, the password is available in plaintext on a user's system. To overcome this problem, use Windows Credential Manager.

Set the following system environment variables:
n3fips_password=CU-username:CU-password

Identifies a crypto user (p. 11) (CU) in the HSM and provides all required login information. Your application authenticates and runs as this CU. The application has the permissions of this CU and can view and manage only the keys that the CU owns and shares. This CU must be available in the HSM specified by the n3fips_partition environment variable. To create a new CU, use createUser (p. 87). To find existing CUs, use listUsers (p. 108).

For example:

```
setx /m n3fips_password test_user:password123
```

Associate a AWS CloudHSM Key with a Certificate

Before you can use AWS CloudHSM keys with third-party tools, such as Microsoft's SignTool, you must import the key's metadata into the local certificate store and associate the metadata with a certificate. To import the key's metadata, use the import_key.exe utility which is included in CloudHSM version 3.0 and higher. The following steps provide additional information, and sample output.

**Step 1: Import your certificate**

On Windows, you should be able to double-click the certificate to import it to your local certificate store. However, if double-clicking doesn't work, use the Microsoft Certreq tool to import the certificate into the certificate manager. For example:

```
certreq -accept certificatename
```

If this action fails and you receive the error, Key not found, continue to Step 2. If the certificate appears in your key store, you've completed the task and no further action is necessary.

**Step 2: Gather certificate-identifying information**

If the previous step wasn't successful, you'll need to associate your private key with a certificate. However, before you can create the association, you must first find the certificate's Unique Container Name and Serial Number. Use a utility, such as certutil, to display the needed certificate information. The following sample output from certutil shows the container name and the serial number.

```
================ Certificate 1 ================
Serial Number: 72000000047f7f7a9d41851b4e000000000004
Issuer: CN=Enterprise-CANotBefore: 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 Certificate
CertHash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45 75 bc 65
No key provider information Simple container name: CertReq-39c04db0-6aa9-4310-93db-db0d966f42c
Unique container name: CertReq-39c04db0-6aa9-4310-93db-db0d966f42c
```

**Step 3: Associate the AWS CloudHSM private key with the certificate**

To associate the key with the certificate, first be sure to start the AWS CloudHSM client daemon (p. 122). Then, use import_key.exe (which is included in CloudHSM version 3.0 and higher) to associate the private key with the certificate. When specifying the certificate, use its simple container name. The following example shows the command and the response. This action only copies the key's metadata; the key remains on the HSM.
Step 4: Update the certificate store

Be certain the AWS CloudHSM client daemon is still running. Then, use the certutil verb, -repairstore, to update the certificate serial number. The following sample shows the command and output. See the Microsoft documentation for information about the -repairstore verb.

C:\Program Files\Amazon\CloudHSM>certutil -f -csp "Cavium Key Storage Provider" -repairstore my "72000000047f7f7a9d41851b4e00000000004"my "Personal"

Certificate 1

Serial Number: 72000000047f7f7a9d41851b4e00000000004
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 Certificate
Cert Hash (sha1): 7f d8 5c 00 27 bf 37 74 3d 71 54 4e c0 94 20 45 bc 65
SDK Version: 3.0
Key Container = CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
Provider = Cavium Key Storage Provider
Private key is NOT exportable
Encryption test passed
CertUtil: -repairstore command completed successfully.

After updating the certificate serial number you can use this certificate and the corresponding AWS CloudHSM private key with any third-party signing tool on Windows.

Code Sample for Cavium CNG Provider

** Example code only - Not for production use **

This page includes example code that has not been fully tested. It is designed for test environments. Do not run this code in production.

The following sample shows how to enumerate the registered cryptographic providers on your system to find the Cavium CNG provider. The sample also shows how to create an asymmetric key pair and how to use the key pair to sign data.

Important

Before you run this example, you must set up the HSM credentials as explained in the prerequisites. For details, see Windows AWS CloudHSM Prerequisites (p. 263).

```cpp
// 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.
```
```c
#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;
}

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...
");
    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;
Integrating Third-Party Applications with AWS CloudHSM

Some of the use cases (p. 1) for AWS CloudHSM involve integrating third-party software applications with the HSM in your AWS CloudHSM cluster. By integrating third-party software with AWS CloudHSM, you can accomplish a variety of security-related goals. The following topics describe how to accomplish some of these goals.

Topics
- Improve Your Web Server's Security with SSL/TLS Offload in AWS CloudHSM (p. 270)
- Configure Windows Server as a Certificate Authority (CA) with AWS CloudHSM (p. 298)
- Oracle Database Transparent Data Encryption (TDE) with AWS CloudHSM (p. 302)
- Use Microsoft SignTool with AWS CloudHSM to Sign Files (p. 306)
- Other Third-party Vendor Integrations (p. 309)

Improve Your Web Server's Security with SSL/TLS Offload in AWS CloudHSM

Web servers and their clients (web browsers) can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS). These protocols confirm the identity of the web server and establish a secure connection to send and receive webpages or other data over the internet. This is commonly known as HTTPS. The web server uses a public–private key pair and an SSL/TLS public key certificate to establish an HTTPS session with each client. This process involves a lot of computation for the web server, but you can offload some of this to the HSMs in your AWS CloudHSM cluster. This is sometimes known as SSL acceleration. Offloading reduces the computational burden on your web server and provides extra security by storing the server’s private key in the HSMs.

The following topics provide an overview of how SSL/TLS offload with AWS CloudHSM works and tutorials for setting up SSL/TLS offload with AWS CloudHSM on the following platforms:
- Linux – Using the NGINX or Apache HTTP Server web server software
- Windows – Using the Internet Information Services (IIS) for Windows Server web server software

Topics
- How SSL/TLS Offload with AWS CloudHSM Works (p. 270)
- Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Linux (p. 271)
- Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Windows (p. 287)

How SSL/TLS Offload with AWS CloudHSM Works

To establish an HTTPS connection, your web server performs a handshake process with clients. As part of this process, the server offloads some of the cryptographic processing to the HSMs, as shown in the following figure. Each step of the process is explained below the figure.

Note
The following image and process assumes that RSA is used for server verification and key exchange. The process is slightly different when Diffie–Hellman is used instead of RSA.
1. The client sends a hello message to the server.
2. The server responds with a hello message and sends the server's certificate.
3. The client performs the following actions:
   a. Verifies that the SSL/TLS server certificate is signed by a root certificate that the client trusts.
   b. Extracts the public key from the server certificate.
   c. Generates a premaster secret and encrypts it with the server's public key.
   d. Sends the encrypted premaster secret to the server.
4. To decrypt the client's premaster secret, the server sends it to the HSM. The HSM uses the private key in the HSM to decrypt the premaster secret and then it sends the premaster secret to the server. Independently, the client and server each use the premaster secret and some information from the hello messages to calculate a master secret.
5. The handshake process ends. For the rest of the session, all messages sent between the client and the server are encrypted with derivatives of the master secret.

To learn how to configure SSL/TLS offload with AWS CloudHSM, see one of the following topics:

- Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Linux (p. 271)
- Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Windows (p. 287)

**Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Linux**

This tutorial provides step-by-step instructions for setting up SSL/TLS offload with AWS CloudHSM on a Linux web server.

**Topics**

- Overview (p. 272)
- Step 1: Set Up the Prerequisites (p. 272)
- Step 2: Generate or Import a Private Key and SSL/TLS Certificate (p. 273)
- Step 3: Configure the Web Server (p. 276)
• Step 4: Enable HTTPS Traffic and Verify the Certificate (p. 281)
• (Optional) Step 5: Add a Load Balancer with Elastic Load Balancing (p. 282)

Overview

On Linux, the NGINX and Apache HTTP Server web server software integrate with OpenSSL to support HTTPS. The AWS CloudHSM dynamic engine for OpenSSL (p. 236) provides an interface that enables the web server software to use the HSMs in your cluster for cryptographic offloading and key storage. The OpenSSL engine is the bridge that connects the web server to your AWS CloudHSM cluster.

To complete this tutorial, you must first choose whether to use the NGINX or Apache web server software on Linux. Then the tutorial shows you how to do the following:

• Install the web server software on an Amazon EC2 instance.
• Configure the web server software to support HTTPS with a private key stored in your AWS CloudHSM cluster.
• (Optional) Use Amazon EC2 to create a second web server instance and Elastic Load Balancing to create a load balancer. Using a load balancer can increase performance by distributing the load across multiple servers. It can also provide redundancy and higher availability if one or more servers fail.

When you're ready to get started, go to Step 1: Set Up the Prerequisites (p. 272).

Step 1: Set Up the Prerequisites

To set up web server SSL/TLS offload with AWS CloudHSM, you need the following:

• An active AWS CloudHSM cluster with at least one HSM.
• An Amazon EC2 instance running a Linux operating system with the following software installed:
  • The AWS CloudHSM client and command line tools.
  • The NGINX or Apache web server application.
  • The AWS CloudHSM dynamic engine for OpenSSL.
• A crypto user (p. 11) (CU) to own and manage the web server's private key on the HSM.

To set up a Linux web server instance and create a CU on the HSM

1. Complete the steps in Getting Started (p. 15). You will then have an active cluster with one HSM and an Amazon EC2 client instance. Your EC2 instance will be configured with the command line tools. Use this client instance as your web server.
2. Connect to your client instance. For more information, see Connecting to Your Linux Instance Using SSH or Connecting to Your Linux Instance from Windows Using PuTTY in the Amazon EC2 documentation. Then do the following:
   a. Choose whether to install the NGINX or Apache web server application. Then complete one of the following steps:
      • To install NGINX, run the following command.

```
sudo yum install -y nginx
```
      • To install Apache, run the following command.

```
sudo yum install -y httpd24 mod24_ssl
```
   b. Install and configure the OpenSSL engine (p. 237).
3. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 42).

4. To create a crypto user (p. 11) (CU) on your HSM, do the following:
   a. Start the AWS CloudHSM client (p. 78).
   b. Update the cloudhsm_mgmt_util configuration file (p. 79).
   c. Use cloudhsm_mgmt_util to create a CU. For more information, see Managing HSM Users (p. 52). Keep track of the CU user name and password. You will need them later when you generate or import the HTTPS private key and certificate for your web server.

After you complete these steps, go to Step 2: Generate or Import a Private Key and SSL/TLS Certificate (p. 273).

Step 2: Generate or Import a Private Key and SSL/TLS Certificate

To enable HTTPS, your web server application (NGINX or Apache) needs a private key and a corresponding SSL/TLS certificate. To use web server SSL/TLS offload with AWS CloudHSM, you must store the private key in an HSM in your AWS CloudHSM cluster. You can accomplish this in one of the following ways:

- If you don't yet have a private key and a corresponding certificate, you can generate a private key in an HSM (p. 273). You can then use the private key to create a certificate signing request (CSR). Use the CSR to create the SSL/TLS certificate.

- If you already have a private key and corresponding certificate, you can import the private key into an HSM (p. 274).

Regardless of which method you choose, you then export a fake PEM private key from the HSM and save it to a file. This file doesn't contain the actual private key. It contains a reference to the private key that is stored on the HSM. Your web server uses the fake PEM private key file and the AWS CloudHSM dynamic engine for OpenSSL to offload SSL/TLS processing to an HSM.

Topics (choose only one)
- Generate a Private Key and Certificate (p. 273)
- Import an Existing Private Key (p. 274)

Generate a Private Key and Certificate

If you don't have a private key and a corresponding SSL/TLS certificate to use for HTTPS, you can generate a private key on an HSM. You can then use the private key to create a certificate signing request (CSR). Sign the CSR to create the certificate.

To generate a private key on an HSM

1. Connect to your client instance.
2. Run the following command to set an environment variable named n3fips_password that contains the user name and password of the cryptographic user (CU). Replace `<CU user name>` with the user name of the cryptographic user. Replace `<password>` with the CU password.

   ```bash
   export n3fips_password=<CU user name>:<password>
   ```

3. Run the following command to use the AWS CloudHSM dynamic engine for OpenSSL to generate a private key on an HSM. This command also exports the fake PEM private key and saves it in a file.
Replace `<web_server_fake_PEM.key>` with the file name you want to use for the exported fake PEM private key.

```
openssl genrsa -engine cloudhsm -out <web_server_fake_PEM.key> 2048
```

**To create a CSR**

Run the following command to use the AWS CloudHSM dynamic engine for OpenSSL to create a certificate signing request (CSR). Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key. Replace `<web_server.csr>` with the name of the file that contains your CSR.

The `req` command is interactive. Respond to each field. The field information is copied into your SSL/TLS certificate.

```
openssl req -engine cloudhsm -new -key <web_server_fake_PEM.key> -out <web_server.csr>
```

In a production environment, you typically use a certificate authority (CA) to create a certificate from a CSR. A CA is not necessary for a test environment. If you do use a CA, send the CSR file (`<web_server.csr>`) to it and use the CA create a signed SSL/TLS certificate. Your web server uses the signed certificate for HTTPS.

As an alternative to using a CA, you can use the AWS CloudHSM dynamic engine for OpenSSL to create a self-signed certificate. Self-signed certificates are not trusted by browsers and should not be used in production environments. They can be used in test environments.

**Warning**

Self-signed certificates should be used in a test environment only. For a production environment, use a more secure method such as a certificate authority to create a certificate.

**To create a self-signed certificate**

Run the following command to use the AWS CloudHSM dynamic engine for OpenSSL to sign your CSR with your private key on your HSM. This creates a self-signed certificate. Replace the following values in the command with your own.

- `<web_server.csr>` – Name of the file that contains the CSR.
- `<web_server_fake_PEM.key>` – Name of the file that contains the fake PEM private key.
- `<web_server.crt>` – Name of the file that will contain your web server certificate.

```
openssl x509 -engine cloudhsm -req -days 365 -in <web_server.csr> -signkey <web_server_fake_PEM.key> -out <web_server.crt>
```

After you complete these steps, go to Step 3: Configure the Web Server (p. 276).

**Import an Existing Private Key**

You might already have a private key and a corresponding SSL/TLS certificate that you use for HTTPS on your web server. If so, you can import that key into an HSM by doing the following:

**To import an existing private key into an HSM**

1. Connect to your Amazon EC2 client instance. If necessary, copy your existing private key and certificate to the instance.
2. Run the following command to start the AWS CloudHSM client.
3. Run the following command to start the key_mgmt_util command line tool.

```
/opt/cloudhsm/bin/key_mgmt_util
```

4. Run the following command to log in to the HSM. Replace `<user name>` and `<password>` with the user name and password of the cryptographic user (CU).

```
loginHSM -u CU -s <user name> -p <password>
```

5. Run the following commands to import your private key into an HSM.

   a. Run the following command to create a symmetric wrapping key that is valid for the current session only. The command and output are shown. Replace the following values with your own:

```
genSymKey -t 31 -s 16 -sess -l wrapping_key_for_import
```

   b. Run the following command to import your existing private key into an HSM. The command and output are shown. Replace the following values with your own:

```
<web_server_existing.key> – Name of the file that contains your private key.
<web_server_imported_key> – Label for your imported private key.
• `<wrapping_key_handle>` – Wrapping key handle generated by the preceding command. In the previous example, the wrapping key handle is 6.

```bash
importPrivateKey -f <web_server_existing.key> -l <web_server_imported_key> -w <wrapping_key_handle>
```

<table>
<thead>
<tr>
<th>BER encoded key length is 1219</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Private Key Unwrapped. Key Handle: 8</td>
</tr>
<tr>
<td>Cluster Error Status</td>
</tr>
<tr>
<td>Node id 0 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

6. Run the following command to export the private key in fake PEM format and save it to a file. Replace the following values with your own.

• `<private_key_handle>` – Handle of the imported private key. This handle was generated by the second command in the preceding step. In the preceding example, the handle of the private key is 8.

• `<web_server_fake_PEM.key>` – Name of the file that contains your exported fake PEM private key.

```bash
getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
```

7. Run the following command to stop key_mgmt_util.

```bash
exit
```

After you complete these steps, go to Step 3: Configure the Web Server (p. 276).

**Step 3: Configure the Web Server**

Update your web server software’s configuration to use the HTTPS certificate and corresponding fake PEM private key that you created in the previous step (p. 273). This will finish setting up your Linux web server software for SSL/TLS offload with AWS CloudHSM.

To update your web server configuration, complete the steps from one of the following procedures. Choose the procedure that corresponds to your web server software.

• Update the configuration for NGINX (p. 276)
• Update the configuration for Apache HTTP Server (p. 280)

**To update the web server configuration for NGINX**

1. Connect to your client instance.

2. Run the following command to create the required directories for the web server certificate and the fake PEM private key.

```bash
sudo mkdir -p /etc/pki/nginx/private
```

3. Run the following command to copy your web server certificate to the required location. Replace `<web_server.crt>` with the name of your web server certificate.
sudo cp <web_server.crt> /etc/pki/nginx/server.crt

4. Run the following command to copy your fake PEM private key to the required location. Replace <web_server_fake_PEM.key> with the name of the file that contains your fake PEM private key.

```
sudo cp <web_server_fake_PEM.key> /etc/pki/nginx/private/server.key
```

5. Run the following command to change the file ownership so that the user named nginx can read them.

```
sudo chown nginx /etc/pki/nginx/server.crt /etc/pki/nginx/private/server.key
```

6. Run the following command to back up the /etc/nginx/nginx.conf file.

```
sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

7. Use a text editor to edit the /etc/nginx/nginx.conf file. At the top of the file, add the following command:

```bash
ssl_engine cloudhsm;
env n3fips_password;
```

Then uncomment the TLS section of the file so that it looks like the following:

```bash
# 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 TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:SEED:!aNULL:!eNULL:!EXPORT:!RC4:!MD5:!PSK:!RSAPSK:!aDH:!aECDH:!EDH-DSS-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA:!SRP;
    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 {
    }
}
```
8. Back up the systemd configuration file, and then set the EnvironmentFile path.

**Amazon Linux**

No action required.

**Amazon Linux 2**

1. Back up the nginx.service file.

```bash
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/nginx.service.backup
```

2. Open the /lib/systemd/system/nginx.service file in a text editor, and then under the [Service] section, add the following path:

   ```
   EnvironmentFile=/etc/sysconfig/nginx
   ```

**CentOS 6**

No action required.

**CentOS 7**

No action required.

**RHEL 6**

No action required.

**RHEL 7**

No action required.

**Ubuntu 16.04**

1. Back up the nginx.service file.

```bash
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/nginx.service.backup
```

2. Open the /lib/systemd/system/nginx.service file in a text editor, and then under the [Service] section, add the following path:

   ```
   EnvironmentFile=/etc/sysconfig/nginx
   ```

9. Check if the /etc/sysconfig/nginx file exists, and then do one of the following:
   - If the file exists, back up the file by running the following command:

   ```bash
   $ sudo cp /etc/sysconfig/nginx /etc/sysconfig/nginx.backup
   ```
   - If the file doesn't exist, open a text editor, and then create a file named nginx in the /etc/sysconfig/ folder.

**Tip**

There is no need to back up the newly created file.
10. Open the /etc/sysconfig/nginx file in a text editor, and then add the Cryptography User (CU) credentials:

```
$ n3fips_password=<CU user name>:<password>
```

Replace the `<CU user name>` and `<password>` with the cryptography user credentials.

Save the file. This requires Linux root permissions.

11. Start the NGINX web server.

Amazon Linux

```
$ sudo service nginx start
```

Amazon Linux 2

```
$ sudo systemctl start nginx
```

CentOS 6

No action required.

CentOS 7

No action required.

RHEL 6

No action required.

RHEL 7

No action required.

Ubuntu 16.04

```
$ sudo systemctl start nginx
```

12. Configure your server to start NGINX when the server starts, if needed.

Amazon Linux

```
$ sudo chkconfig nginx on
```

Amazon Linux 2

```
$ sudo systemctl enable nginx
```

CentOS 6

No action required.

CentOS 7

No action required.

RHEL 6

No action required.
RHEL 7

No action required.

Ubuntu 16.04

$ sudo systemctl enable nginx

After you update your web server configuration, go to Step 4: Enable HTTPS Traffic and Verify the Certificate (p. 281).

To update the web server configuration for Apache

1. Connect to your Amazon EC2 client instance.
2. Run the following command to make a backup copy of the default certificate.

   ```
   sudo cp /etc/pki/tls/certs/localhost.crt /etc/pki/tls/certs/localhost.crt.backup
   ```

3. Run the following command to make a backup copy of the default private key.

   ```
   sudo cp /etc/pki/tls/private/localhost.key /etc/pki/tls/private/localhost.key.backup
   ```

4. Run the following command to copy your web server certificate to the required location. Replace `<web_server.crt>` with the name of your web server certificate.

   ```
   sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
   ```

5. Run the following command to copy your fake PEM private key to the required location. Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

   ```
   sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
   ```

6. Run the following command to change the ownership of these files so that the user named `apache` can read them.

   ```
   sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
   ```

7. Run the following command to make a backup copy of the file named `/etc/httpd/conf.d/ssl.conf`.

   ```
   sudo cp /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.conf.backup
   ```

8. Use a text editor to edit the file named `/etc/httpd/conf.d/ssl.conf`. Replace the line that starts with `SSLCryptoDevice` so that it looks like the following:

   ```
   SSLCryptoDevice cloudhsm
   ```

   Save the file. This requires Linux root permissions.

9. Run the following command to back up the `/etc/apache2/envvars` file.

   ```
   sudo cp /etc/apache2/envvars /etc/apache2/envvars.backup
   ```

10. Use a text editor to edit the `/etc/apache2/envvars` file. Add the following command, specifying the user name and password of the crypto user (CU). Replace `<CU user name>` with the name of the crypto user. Replace `<password>` with the CU password.
export n3fips_password=<CU user name>:<password>

Save the file. This requires Linux root permissions.

11. Run the following command to start the Apache HTTP Server.

```
sudo service httpd start
```

12. Run the following command to configure your server to start Apache when the server starts.

```
sudo chkconfig httpd on
```

After you update your web server configuration, go to Step 4: Enable HTTPS Traffic and Verify the Certificate (p. 281).

**Step 4: Enable HTTPS Traffic and Verify the Certificate**

After you configure your web server for SSL/TLS offload with AWS CloudHSM, add your web server instance to a security group that allows inbound HTTPS traffic. This allows clients, such as web browsers, to establish an HTTPS connection with your web server. Then make an HTTPS connection to your web server and verify that it's using the certificate that you configured for SSL/TLS offload with AWS CloudHSM.

** Topics**
- Enable Inbound HTTPS Connections (p. 281)
- Verify That HTTPS Uses the Certificate That You Configured (p. 282)

**Enable Inbound HTTPS Connections**

To connect to your web server from a client (such as a web browser), create a security group that allows inbound HTTPS connections. Specifically, it should allow inbound TCP connections on port 443. Assign this security group to your web server.

**To create a security group for HTTPS and assign it to your web server**

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Security Groups in the navigation pane.
4. For Create Security Group, do the following:
   a. For Security group name, type a name for the security group that you are creating.
   b. (Optional) Type a description of the security group that you are creating.
   c. For VPC, choose the VPC that contains your web server Amazon EC2 instance.
   d. Choose Add Rule.
   e. For Type, choose HTTPS.
5. Choose Create.
6. In the navigation pane, choose Instances.
7. Select the check box next to your web server instance. Then choose Actions, Networking, and Change Security Groups.
8. Select the check box next to the security group that you created for HTTPS. Then choose Assign Security Groups.
Verify That HTTPS Uses the Certificate That You Configured

After you add the web server to a security group, you can verify that SSL/TLS offload with AWS CloudHSM is working. You can do this with a web browser or with a tool such as OpenSSL s_client.

**To verify SSL/TLS offload with a web browser**

1. Use a web browser to connect to your web server using the public DNS name or IP address of the server. Ensure that the URL in the address bar begins with https://. For example, https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/.

   **Tip**
   You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

2. Use your web browser to view the web server certificate. For more information, see the following:
   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

3. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

**To verify SSL/TLS offload with OpenSSL s_client**

1. Run the following OpenSSL command to connect to your web server using HTTPS. Replace `<server name>` with the public DNS name or IP address of your web server.

   ```bash
   openssl s_client -connect <server name>:443
   ```

   **Tip**
   You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

2. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

You now have a website that is secured with HTTPS. The private key for the web server is stored in an HSM in your AWS CloudHSM cluster. However, you have only one web server. To set up a second web server and a load balancer for higher availability, go to **Optional Step 5: Add a Load Balancer with Elastic Load Balancing (p. 282)**.

**Optional Step 5: Add a Load Balancer with Elastic Load Balancing**

After you set up SSL/TLS offload with one web server, you can create more web servers and an Elastic Load Balancing load balancer that routes HTTPS traffic to the web servers. A load balancer can reduce the load on your individual web servers by balancing traffic across two or more servers. It can also increase the availability of your website because the load balancer monitors the health of your web servers and only routes traffic to healthy servers. If a web server fails, the load balancer automatically stops routing traffic to it.
Create a Subnet for a Second Web Server

Before you can create another web server, you need to create a new subnet in the same VPC that contains your existing web server and AWS CloudHSM cluster.

To create a new subnet

1. Open the Subnets section of the Amazon VPC console at https://console.aws.amazon.com/vpc/home#subnets:
2. Choose Create Subnet.
3. In the Create Subnet dialog box, do the following:
   a. For Name tag, type a name for your subnet.
   b. For VPC, choose the AWS CloudHSM VPC that contains your existing web server and AWS CloudHSM cluster.
   c. For Availability Zone, choose an Availability Zone that is different from the one that contains your existing web server.
   d. For IPv4 CIDR block, type the CIDR block to use for the subnet. For example, type 10.0.10.0/24.
   e. Choose Yes, Create.
4. Select the check box next to the public subnet that contains your existing web server. This is different from the public subnet that you created in the previous step.
5. In the content pane, choose the Route Table tab. Then choose the link for the route table.
6. Select the check box next to the route table.
7. Choose the Subnet Associations tab. Then choose Edit.
8. Select the check box next to the public subnet that you created earlier in this procedure. Then choose Save.
Create the Second Web Server

Complete the following steps to create a second web server with the same configuration as your existing web server.

To create a second web server

1. Open the **Instances** section of the Amazon EC2 console at https://console.aws.amazon.com/ec2/v2/home#Instances:.
2. Select the check box next to your existing web server instance.
3. Choose **Actions**, **Image**, and then **Create Image**.
4. In the **Create Image** dialog box, do the following:
   a. For **Image name**, type a name for the image.
   b. For **Image description**, type a description for the image.
   c. Choose **Create Image**. This action reboots your existing web server.
   d. Choose the **View pending image ami-*<AMI ID>* link.

   ![Create Image request received.
   View pending image ami-ca6d57aa](image)
   Any snapshots backing your new EBS image.

   In the **Status** column, note your image status. When your image status is **available** (this might take several minutes), go to the next step.

5. In the navigation pane, choose **Instances**.
6. Select the check box next to your existing web server.
7. Choose **Actions** and choose **Launch More Like This**.
8. Choose **Edit AMI**.
9. In the left navigation pane, choose **My AMIs**. Then clear the text in the search box.
10. Next to your web server image, choose **Select**.
11. Choose **Yes, I want to continue with this AMI (<image name> - ami-*<AMI ID>*).**
12. Choose **Next**.
13. Select an instance type, and then choose **Next: Configure Instance Details**.
14. For **Step 3: Configure Instance Details**, do the following:
   a. For **Network**, choose the VPC that contains your existing web server.
   b. For **Subnet**, choose the public subnet that you created for the second web server.
   c. For **Auto-assign Public IP**, choose **Enable**.
   d. Change the remaining instance details as preferred. Then choose **Next: Add Storage**.
15. Change the storage settings as preferred. Then choose **Next: Add Tags**.
16. Add or edit tags as preferred. Then choose **Next: Configure Security Group**.

17. For **Step 6: Configure Security Group**, do the following:
   
   a. For **Assign a security group**, choose **Select an existing security group**. AWS CloudHSM created this security group on your behalf when you created the cluster (p. 17). You must choose this security group to allow the web server instance to connect to the HSMs in the cluster.
   
   b. Select the check box next to the security group named `cloudhsm-<cluster ID>-sg`. AWS CloudHSM created this security group on your behalf when you created the cluster (p. 17). You must choose this security group to allow the web server instance to connect to the HSMs in the cluster.
   
   c. Select the check box next to the security group that allows inbound HTTPS traffic. You created this security group previously (p. 281).
   
   d. (Optional) Select the check box next to a security group that allows inbound SSH (for Linux) or RDP (for Windows) traffic from your network. That is, the security group must allow inbound TCP traffic on port 22 (for SSH on Linux) or port 3389 (for RDP on Windows). Otherwise, you cannot connect to your client instance. If you don't have a security group like this, you must create one and then assign it to your client instance later.

   Choose **Review and Launch**.

18. Review your instance details, and then choose **Launch**.

19. Choose whether to launch your instance with an existing key pair, create a new key pair, or launch your instance without a key pair.

   • To use an existing key pair, do the following:
     1. Choose **Choose an existing key pair**.
     2. For **Select a key pair**, choose the key pair to use.
     3. Select the check box next to **I acknowledge that I have access to the selected private key file (<private key file name>.pem), and that without this file, I won't be able to log into my instance**.

   • To create a new key pair, do the following:
     1. Choose **Create a new key pair**.
     2. For **Key pair name**, type a key pair name.
     3. Choose **Download Key Pair** and save the private key file in a secure and accessible location.

       **Warning**
       You cannot download the private key file again after this point. If you do not download the private key file now, you will be unable to access the client instance.

   • To launch your instance without a key pair, do the following:
     1. Choose **Proceed without a key pair**.
     2. Select the check box next to **I acknowledge that I will not be able to connect to this instance unless I already know the password built into this AMI**.

   Choose **Launch Instances**.

**Create the Load Balancer**

Complete the following steps to create an Elastic Load Balancing load balancer that routes HTTPS traffic to your web servers.

**To create a load balancer**

1. Open the **Load Balancers** section of the Amazon EC2 console at https://console.aws.amazon.com/ec2/v2/home#LoadBalancers:
2. Choose **Create Load Balancer**.
3. In the **Network Load Balancer** section, choose **Create**.

4. For **Step 1: Configure Load Balancer**, do the following:
   a. For **Name**, type a name for the load balancer that you are creating.
   b. In the **Listeners** section, for **Load Balancer Port**, change the value to **443**.
   c. In the **Availability Zones** section, for **VPC**, choose the VPC that contains your web servers.
   d. In the **Availability Zones** section, choose the subnets that contain your web servers.
   e. Choose **Next: Configure Routing**.

5. For **Step 2: Configure Routing**, do the following:
   a. For **Name**, type a name for the target group that you are creating.
   b. For **Port**, change the value to **443**.
   c. Choose **Next: Register Targets**.

6. For **Step 3: Register Targets**, do the following:
   a. In the **Instances** section, select the check boxes next to your web server instances. Then choose **Add to registered**.
   b. Choose **Next: Review**.

7. Review your load balancer details, then choose **Create**.

8. When the load balancer has been successfully created, choose **Close**.

After you complete the preceding steps, the Amazon EC2 console shows your Elastic Load Balancing load balancer.

When your load balancer's state is active, you can verify that the load balancer is working. That is, you can verify that it's sending HTTPS traffic to your web servers with SSL/TLS offload with AWS CloudHSM. You can do this with a web browser or a tool such as OpenSSL s_client.

**To verify that your load balancer is working with a web browser**

1. In the Amazon EC2 console, find the **DNS name** for the load balancer that you just created. Then select the DNS name and copy it.
2. Use a web browser such as Mozilla Firefox or Google Chrome to connect to your load balancer using the load balancer's DNS name. Ensure that the URL in the address bar begins with https://.

   **Tip**
   You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

3. Use your web browser to view the web server certificate. For more information, see the following:
   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

4. Ensure that the certificate is the one that you configured the web server to use.

**To verify that your load balancer is working with OpenSSL s_client**

1. Use the following OpenSSL command to connect to your load balancer using HTTPS. Replace `<DNS name>` with the DNS name of your load balancer.
Tip
You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

2. Ensure that the certificate is the one that you configured the web server to use.

You now have a website that is secured with HTTPS, with the web server's private key stored in an HSM in your AWS CloudHSM cluster. Your website has two web servers and a load balancer to help improve efficiency and availability.

Tutorial: Using SSL/TLS Offload with AWS CloudHSM on Windows

This tutorial provides step-by-step instructions for setting up SSL/TLS offload with AWS CloudHSM on a Windows web server.

Topics
- Overview (p. 287)
- Step 1: Set Up the Prerequisites (p. 287)
- Step 2: Create a Certificate Signing Request (CSR) and Certificate (p. 289)
- Step 3: Configure the Web Server (p. 291)
- Step 4: Enable HTTPS Traffic and Verify the Certificate (p. 292)
- (Optional) Step 5: Add a Load Balancer with Elastic Load Balancing (p. 294)

Overview

On Windows, the Internet Information Services (IIS) for Windows Server web server application natively supports HTTPS. The AWS CloudHSM key storage provider (KSP) for Microsoft's Cryptography API: Next Generation (CNG) (p. 261) provides the interface that allows IIS to use the HSMs in your cluster for cryptographic offloading and key storage. The AWS CloudHSM KSP is the bridge that connects IIS to your AWS CloudHSM cluster.

This tutorial shows you how to do the following:

- Install the web server software on an Amazon EC2 instance.
- Configure the web server software to support HTTPS with a private key stored in your AWS CloudHSM cluster.
- (Optional) Use Amazon EC2 to create a second web server instance and Elastic Load Balancing to create a load balancer. Using a load balancer can increase performance by distributing the load across multiple servers. It can also provide redundancy and higher availability if one or more servers fail.

When you're ready to get started, go to Step 1: Set Up the Prerequisites (p. 287).

Step 1: Set Up the Prerequisites

To set up web server SSL/TLS offload with AWS CloudHSM, you need the following:
• An active AWS CloudHSM cluster with at least one HSM.
• An Amazon EC2 instance running a Windows operating system with the following software installed:
  • The AWS CloudHSM client software for Windows.
  • Internet Information Services (IIS) for Windows Server.
• A crypto user (p. 11) (CU) to own and manage the web server's private key on the HSM.

**Note**
This tutorial uses Microsoft Windows Server 2016. Microsoft Windows Server 2012 is also supported, but Microsoft Windows Server 2012 R2 is not.

**To set up a Windows Server instance and create a CU on the HSM**

1. Complete the steps in Getting Started (p. 15). When you launch the Amazon EC2 client, choose a Windows Server 2016 or Windows Server 2012 AMI. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 42).
3. Connect to your Windows server. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
4. To create a cryptographic user (CU) on your HSM, do the following:
   a. Start the AWS CloudHSM client (p. 78).
   b. Update the cloudhsm_mgmt_util configuration file (p. 79).
   c. Start cloudhsm_mgmt_util (p. 80).
   d. Enable end-to-end encryption (p. 81).
   e. Log in to the HSMs (p. 81) with the user name and password of a crypto officer (CO).
   f. Create a crypto user (CU) (p. 52). Keep track of the CU user name and password. You will need them to complete the next step.
5. Set the login credentials for the HSM (p. 263), using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download psexec.exe from SysInternals to run the following command as NT Authority\SYSTEM:

   ```bash
   psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe" --username <username> --password <password>
   ```

Replace `<username>` and `<password>` with the HSM credentials.

**To install IIS on your Windows Server**

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
2. On your Windows server, start Server Manager.
3. In the Server Manager dashboard, choose Add roles and features.
4. Read the Before you begin information, and then choose Next.
5. For Installation Type, choose Role-based or feature-based installation. Then choose Next.
6. For Server Selection, choose Select a server from the server pool. Then choose Next.
7. For Server Roles, do the following:
   a. Select Web Server (IIS).
b. For **Add features that are required for Web Server (IIS)**, choose **Add Features**.

c. Choose **Next** to finish selecting server roles.

8. For **Features**, accept the defaults. Then choose **Next**.

9. Read the **Web Server Role (IIS)** information. Then choose **Next**.

10. For **Select role services**, accept the defaults or change the settings as preferred. Then choose **Next**.

11. For **Confirmation**, read the confirmation information. Then choose **Install**.

12. After the installation is complete, choose **Close**.

After you complete these steps, go to **Step 2: Create a Certificate Signing Request (CSR) and Certificate** (p. 289).

### Step 2: Create a Certificate Signing Request (CSR) and Certificate

To enable HTTPS, your web server needs an SSL/TLS certificate and a corresponding private key. To use SSL/TLS offload with AWS CloudHSM, you store the private key in the HSM in your AWS CloudHSM cluster. To do this, you use the **AWS CloudHSM key storage provider (KSP) for Microsoft's Cryptography API: Next Generation (CNG)** (p. 261) to create a certificate signing request (CSR). Then you give the CSR to a certificate authority (CA), which signs the CSR to produce a certificate.

**Topics**

- Create a CSR (p. 289)
- Get a Signed Certificate and Import It (p. 290)

### Create a CSR

Use the AWS CloudHSM KSP on your Windows Server to create a CSR.

**To create a CSR**

1. If you haven't already done so, connect to your Windows server. For more information, see **Connect to Your Instance** in the **Amazon EC2 User Guide for Windows Instances**.

2. Start the AWS CloudHSM client (p. 78).

3. On your Windows Server, use a text editor to create a certificate request file named `IISCertRequest.inf`. The following shows the contents of an example `IISCertRequest.inf` file. For more information about the sections, keys, and values that you can specify in the file, see Microsoft's documentation. Do not change the `ProviderName` value.

   ```
   [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. Use the **Windows certreq command** to create a CSR from the `IISCertRequest.inf` file that you created in the previous step. The following example saves the CSR to a file named `IISCertRequest.csr`. If you used a different file name for your certificate request file,
replace IISCertRequest.inf with the appropriate file name. You can optionally replace IISCertRequest.csr with a different file name for your CSR file.

```plaintext
C:\>certreq -new IISCertRequest.inf IISCertRequest.csr
SDK Version: 2.03
CertReq: Request Created
```

The IISCertRequest.csr file contains your CSR. You need this CSR to get a signed certificate.

### Get a Signed Certificate and Import It

In a production environment, you typically use a certificate authority (CA) to create a certificate from a CSR. A CA is not necessary for a test environment. If you do use a CA, send the CSR file (IISCertRequest.csr) to it and use the CA to create a signed SSL/TLS certificate.

As an alternative to using a CA, you can use a tool like OpenSSL to create a self-signed certificate.

**Warning**

Self-signed certificates are not trusted by browsers and should not be used in production environments. They can be used in test environments.

The following procedures show how to create a self-signed certificate and use it to sign your web server’s CSR.

### To create a self-signed certificate

1. Use the following OpenSSL command to create a private key. You can optionally replace SelfSignedCA.key with the file name to contain your private key.

   ```plaintext
   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. Use the following OpenSSL command to create a self-signed certificate using the private key that you created in the previous step. This is an interactive command. Read the on-screen instructions and follow the prompts. Replace SelfSignedCA.key with the name of the file that contains your private key (if different). You can optionally replace SelfSignedCA.crt with the file name to contain your self-signed certificate.

   ```plaintext
   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 [:]:
   ```
To use your self-signed certificate to sign your web server's CSR

- Use the following OpenSSL command to use your private key and self-signed certificate to sign the CSR. Replace the following with the names of the files that contain the corresponding data (if different).
  - `IISCertRequest.csr` – The name of the file that contains your web server's CSR
  - `SelfSignedCA.crt` – The name of the file that contains your self-signed certificate
  - `SelfSignedCA.key` – The name of the file that contains your private key
  - `IISCert.crt` – The name of the file to contain your web server's signed certificate

```
openssl x509 -req -days 365 -in IISCertRequest.csr
  -CA SelfSignedCA.crt
  -CAkey SelfSignedCA.key
  -CAcreateserial
  -out IISCert.crt
```

After you complete the previous step, you have a signed certificate for your web server (`IISCert.crt`) and a self-signed certificate (`SelfSignedCA.crt`). When you have these files, go to Step 3: Configure the Web Server (p. 291).

Step 3: Configure the Web Server

Update your IIS website's configuration to use the HTTPS certificate that you created at the end of the previous step (p. 289). This will finish setting up your Windows web server software (IIS) for SSL/TLS offload with AWS CloudHSM.

If you used a self-signed certificate to sign your CSR, you must first import the self-signed certificate into the Windows Trusted Root Certification Authorities.

To import your self-signed certificate into the Windows Trusted Root Certification Authorities

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
2. Copy your self-signed certificate to your Windows server.
3. On your Windows Server, open the Control Panel.
4. For Search Control Panel, type certificates. Then choose Manage computer certificates.
5. In the Certificates - Local Computer window, double-click Trusted Root Certification Authorities.
6. Right-click on Certificates and then choose All Tasks, Import.
7. In the Certificate Import Wizard, choose Next.
8. Choose Browse, then find and select your self-signed certificate. If you created your self-signed certificate by following the instructions in the previous step of this tutorial (p. 289), your self-signed certificate is named `SelfSignedCA.crt`. Choose Open.
9. Choose Next.
10. For Certificate Store, choose Place all certificates in the following store. Then ensure that Trusted Root Certification Authorities is selected for Certificate store.
11. Choose Next and then choose Finish.
To update the IIS website's configuration

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
2. Start the AWS CloudHSM client (p. 78).
3. Copy your web server's signed certificate—the one that you created at the end of this tutorial's previous step (p. 289)—to your Windows server.
4. On your Windows Server, use the Windows certreq command to accept the signed certificate, as in the following example. Replace IISCert.crt with the name of the file that contains your web server's signed certificate.

   C:\>certreq -accept IISCert.crt

   SDK Version: 2.03

5. On your Windows server, start Server Manager.
6. In the Server Manager dashboard, in the top right corner, choose Tools, Internet Information Services (IIS) Manager.
7. In the Internet Information Services (IIS) Manager window, double-click your server name. Then double-click Sites. Select your website.
8. Select SSL Settings. Then, on the right side of the window, choose Bindings.
9. In the Site Bindings window, choose Add.
10. For Type, choose https. For SSL certificate, choose the HTTPS certificate that you created at the end of this tutorial's previous step (p. 289).

   Note
   If you encounter an error during this certificate binding, restart your server and retry this step.
11. Choose OK.

After you update your website's configuration, go to Step 4: Enable HTTPS Traffic and Verify the Certificate (p. 292).

Step 4: Enable HTTPS Traffic and Verify the Certificate

After you configure your web server for SSL/TLS offload with AWS CloudHSM, add your web server instance to a security group that allows inbound HTTPS traffic. This allows clients, such as web browsers, to establish an HTTPS connection with your web server. Then make an HTTPS connection to your web server and verify that it's using the certificate that you configured for SSL/TLS offload with AWS CloudHSM.

Topics
- Enable Inbound HTTPS Connections (p. 292)
- Verify That HTTPS Uses the Certificate That You Configured (p. 293)

Enable Inbound HTTPS Connections

To connect to your web server from a client (such as a web browser), create a security group that allows inbound HTTPS connections. Specifically, it should allow inbound TCP connections on port 443. Assign this security group to your web server.

To create a security group for HTTPS and assign it to your web server

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Security Groups in the navigation pane.
3. Choose **Create Security Group**.

4. For **Create Security Group**, do the following:
   a. For **Security group name**, type a name for the security group that you are creating.
   b. (Optional) Type a description of the security group that you are creating.
   c. For **VPC**, choose the VPC that contains your web server Amazon EC2 instance.
   d. Choose **Add Rule**.
   e. For **Type**, choose **HTTPS**.

5. Choose **Create**.

6. In the navigation pane, choose **Instances**.

7. Select the check box next to your web server instance. Then choose **Actions**, **Networking**, and **Change Security Groups**.

8. Select the check box next to the security group that you created for HTTPS. Then choose **Assign Security Groups**.

**Verify That HTTPS Uses the Certificate That You Configured**

After you add the web server to a security group, you can verify that SSL/TLS offload with AWS CloudHSM is working. You can do this with a web browser or with a tool such as **OpenSSL s_client**.

**To verify SSL/TLS offload with a web browser**

1. Use a web browser to connect to your web server using the public DNS name or IP address of the server. Ensure that the URL in the address bar begins with https://. For example, `https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`.

   **Tip**
   You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, `https://www.example.com/`) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

2. Use your web browser to view the web server certificate. For more information, see the following:
   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

3. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

**To verify SSL/TLS offload with OpenSSL s_client**

1. Run the following OpenSSL command to connect to your web server using HTTPS. Replace `<server name>` with the public DNS name or IP address of your web server.

   ```bash
   openssl s_client -connect <server name>:443
   ```

   **Tip**
   You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, `https://www.example.com/`) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

293
2. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

You now have a website that is secured with HTTPS. The private key for the web server is stored in an HSM in your AWS CloudHSM cluster. However, you have only one web server. To set up a second web server and a load balancer for higher availability, go to (Optional) Step 5: Add a Load Balancer with Elastic Load Balancing (p. 294).

(Optional) Step 5: Add a Load Balancer with Elastic Load Balancing

After you set up SSL/TLS offload with one web server, you can create more web servers and an Elastic Load Balancing load balancer that routes HTTPS traffic to the web servers. A load balancer can reduce the load on your individual web servers by balancing traffic across two or more servers. It can also increase the availability of your website because the load balancer monitors the health of your web servers and only routes traffic to healthy servers. If a web server fails, the load balancer automatically stops routing traffic to it.

Topics
• Create a Subnet for a Second Web Server (p. 294)
• Create the Second Web Server (p. 295)
• Create the Load Balancer (p. 297)

Create a Subnet for a Second Web Server

Before you can create another web server, you need to create a new subnet in the same VPC that contains your existing web server and AWS CloudHSM cluster.

To create a new subnet

1. Open the Subnets section of the Amazon VPC console at https://console.aws.amazon.com/vpc/home#subnets.
2. Choose Create Subnet.
3. In the Create Subnet dialog box, do the following:
   a. For Name tag, type a name for your subnet.
   b. For VPC, choose the AWS CloudHSM VPC that contains your existing web server and AWS CloudHSM cluster.
   c. For Availability Zone, choose an Availability Zone that is different from the one that contains your existing web server.
   d. For IPv4 CIDR block, type the CIDR block to use for the subnet. For example, type 10.0.10.0/24.
   e. Choose Yes, Create.
4. Select the check box next to the public subnet that contains your existing web server. This is different from the public subnet that you created in the previous step.
5. In the content pane, choose the Route Table tab. Then choose the link for the route table.
6. Select the check box next to the route table.
7. Choose the **Subnet Associations** tab. Then choose **Edit**.
8. Select the check box next to the public subnet that you created earlier in this procedure. Then choose **Save**.

### Create the Second Web Server

Complete the following steps to create a second web server with the same configuration as your existing web server.

**To create a second web server**

1. Open the **Instances** section of the Amazon EC2 console at https://console.aws.amazon.com/ec2/v2/home#Instances:.
2. Select the check box next to your existing web server instance.
3. Choose **Actions, Image, and then Create Image**.
4. In the **Create Image** dialog box, do the following:
   a. For **Image name**, type a name for the image.
   b. For **Image description**, type a description for the image.
   c. Choose **Create Image**. This action reboots your existing web server.
   d. Choose the **View pending image ami-<AMI ID>** link.

In the **Status** column, note your image status. When your image status is **available** (this might take several minutes), go to the next step.

5. In the navigation pane, choose **Instances**.
6. Select the check box next to your existing web server.
7. Choose **Actions and choose Launch More Like This**.
8. Choose **Edit AMI**.
9. In the left navigation pane, choose **My AMIs**. Then clear the text in the search box.
10. Next to your web server image, choose **Select**.
11. Choose **Yes, I want to continue with this AMI** (<image name> - ami-<AMI ID>).
12. Choose **Next**.
13. Select an instance type, and then choose **Next: Configure Instance Details**.
14. For **Step 3: Configure Instance Details**, do the following:
   a. For **Network**, choose the VPC that contains your existing web server.
   b. For **Subnet**, choose the public subnet that you created for the second web server.
   c. For **Auto-assign Public IP**, choose **Enable**.
   d. Change the remaining instance details as preferred. Then choose **Next: Add Storage**.
15. Change the storage settings as preferred. Then choose **Next: Add Tags**.
16. Add or edit tags as preferred. Then choose **Next: Configure Security Group**.
17. For **Step 6: Configure Security Group**, do the following:
   a. For **Assign a security group**, choose **Select an existing security group**.
   b. Select the check box next to the security group named **cloudhsm-<cluster ID>-sg**. AWS CloudHSM created this security group on your behalf when you created the cluster (p. 17). You must choose this security group to allow the web server instance to connect to the HSMs in the cluster.
   c. Select the check box next to the security group that allows inbound HTTPS traffic. You created this security group previously (p. 292).
   d. (Optional) Select the check box next to a security group that allows inbound SSH (for Linux) or RDP (for Windows) traffic from your network. That is, the security group must allow inbound TCP traffic on port 22 (for SSH on Linux) or port 3389 (for RDP on Windows). Otherwise, you cannot connect to your client instance. If you don't have a security group like this, you must create one and then assign it to your client instance later.

Choose **Review and Launch**.
18. Review your instance details, and then choose **Launch**.
19. Choose whether to launch your instance with an existing key pair, create a new key pair, or launch your instance without a key pair.
   • To use an existing key pair, do the following:
     1. Choose **Choose an existing key pair**.
     2. For **Select a key pair**, choose the key pair to use.
     3. Select the check box next to **I acknowledge that I have access to the selected private key file (<private key file name>.pem), and that without this file, I won't be able to log into my instance**.
   • To create a new key pair, do the following:
     1. Choose **Create a new key pair**.
     2. For **Key pair name**, type a key pair name.
3. Choose **Download Key Pair** and save the private key file in a secure and accessible location.

   **Warning**
   You cannot download the private key file again after this point. If you do not download the private key file now, you will be unable to access the client instance.

   • To launch your instance without a key pair, do the following:
     1. Choose **Proceed without a key pair**.
     2. Select the check box next to **I acknowledge that I will not be able to connect to this instance unless I already know the password built into this AMI**.

   Choose **Launch Instances**.

**Create the Load Balancer**

Complete the following steps to create an Elastic Load Balancing load balancer that routes HTTPS traffic to your web servers.

**To create a load balancer**

1. Open the **Load Balancers** section of the Amazon EC2 console at https://console.aws.amazon.com/ec2/v2/home#LoadBalancers:

2. Choose **Create Load Balancer**.

3. In the **Network Load Balancer** section, choose **Create**.

4. For **Step 1: Configure Load Balancer**, do the following:
   a. For **Name**, type a name for the load balancer that you are creating.
   b. In the **Listeners** section, for **Load Balancer Port**, change the value to **443**.
   c. In the **Availability Zones** section, for **VPC**, choose the VPC that contains your web servers.
   d. In the **Availability Zones** section, choose the subnets that contain your web servers.
   e. Choose **Next: Configure Routing**.

5. For **Step 2: Configure Routing**, do the following:
   a. For **Name**, type a name for the target group that you are creating.
   b. For **Port**, change the value to **443**.
   c. Choose **Next: Register Targets**.

6. For **Step 3: Register Targets**, do the following:
   a. In the **Instances** section, select the check boxes next to your web server instances. Then choose **Add to registered**.
   b. Choose **Next: Review**.

7. Review your load balancer details, then choose **Create**.

8. When the load balancer has been successfully created, choose **Close**.

After you complete the preceding steps, the Amazon EC2 console shows your Elastic Load Balancing load balancer.

When your load balancer's state is active, you can verify that the load balancer is working. That is, you can verify that it's sending HTTPS traffic to your web servers with SSL/TLS offload with AWS CloudHSM. You can do this with a web browser or a tool such as **OpenSSL s_client**.
To verify that your load balancer is working with a web browser

1. In the Amazon EC2 console, find the DNS name for the load balancer that you just created. Then select the DNS name and copy it.

2. Use a web browser such as Mozilla Firefox or Google Chrome to connect to your load balancer using the load balancer’s DNS name. Ensure that the URL in the address bar begins with https://.

   Tip: You can use a DNS service such as Amazon Route 53 to route your website’s domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

3. Use your web browser to view the web server certificate. For more information, see the following:
   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

4. Ensure that the certificate is the one that you configured the web server to use.

To verify that your load balancer is working with OpenSSL s_client

1. Use the following OpenSSL command to connect to your load balancer using HTTPS. Replace `<DNS name>` with the DNS name of your load balancer.

   ```bash
   openssl s_client -connect <DNS name>:443
   ```

   Tip: You can use a DNS service such as Amazon Route 53 to route your website’s domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the Amazon Route 53 Developer Guide or in the documentation for your DNS service.

2. Ensure that the certificate is the one that you configured the web server to use.

You now have a website that is secured with HTTPS, with the web server’s private key stored in an HSM in your AWS CloudHSM cluster. Your website has two web servers and a load balancer to help improve efficiency and availability.

Configure Windows Server as a Certificate Authority (CA) with AWS CloudHSM

In a public key infrastructure (PKI), a certificate authority (CA) is a trusted entity that issues digital certificates. These digital certificates bind a public key to an identity (a person or organization) by means of public key cryptography and digital signatures. To operate a CA, you must maintain trust by protecting the private key that signs the certificates issued by your CA. You can store the private key in the HSM in your AWS CloudHSM cluster, and use the HSM to perform the cryptographic signing operations.

In this tutorial, you use Windows Server and AWS CloudHSM to configure a CA. You install the AWS CloudHSM client software for Windows on your Windows server, then add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you configure this role, you use an AWS
CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster. The KSP is the bridge that connects your Windows server to your AWS CloudHSM cluster. In the last step, you sign a certificate signing request (CSR) with your Windows Server CA.

For more information, see the following topics:

**Topics**
- Windows Server CA Step 1: Set Up the Prerequisites (p. 299)
- Windows Server CA Step 2: Create a Windows Server CA with AWS CloudHSM (p. 300)
- Windows Server CA Step 3: Sign a Certificate Signing Request (CSR) with Your Windows Server CA with AWS CloudHSM (p. 301)

**Windows Server CA Step 1: Set Up the Prerequisites**

To set up Windows Server as a certificate authority (CA) with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Windows Server operating system with the AWS CloudHSM client software for Windows installed. This tutorial uses Microsoft Windows Server 2016.
- A cryptographic user (CU) to own and manage the CA's private key on the HSM.

**To set up the prerequisites for a Windows Server CA with AWS CloudHSM**

1. Complete the steps in Getting Started (p. 15). When you launch the Amazon EC2 client, choose a Windows Server AMI. This tutorial uses Microsoft Windows Server 2016. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 42).
3. Connect to your client instance. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
4. To create a cryptographic user (CU) on your HSM, do the following:
   a. Start the AWS CloudHSM client (p. 78).
   b. Update the cloudhsm_mgmt_util configuration file (p. 79).
   c. Start cloudhsm_mgmt_util (p. 80).
   d. Enable end-to-end encryption (p. 81).
   e. Log in to the HSMs (p. 81) with the user name and password of a crypto officer (CO).
   f. Create a crypto user (CU) (p. 52). Keep track of the CU user name and password. You will need them to complete the next step.
5. Set the login credentials for the HSM (p. 263), using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download psexec.exe from SysInternals to run the following command as NT Authority\SYSTEM:
   ```
   psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe" --username <username> --password <password>
   ```
   Replace <username> and <password> with the HSM credentials.

To create a Windows Server CA with AWS CloudHSM, go to Create Windows Server CA (p. 300).
Windows Server CA Step 2: Create a Windows Server CA with AWS CloudHSM

To create a Windows Server CA, you add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you add this role, you use an AWS CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster.

Note
When you create your Windows Server CA, you can choose to create a root CA or a subordinate CA. You typically make this decision based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.

To add the AD CS role to your Windows Server and create the CA's private key

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
2. On your Windows server, start Server Manager.
3. In the Server Manager dashboard, choose Add roles and features.
4. Read the Before you begin information, and then choose Next.
5. For Installation Type, choose Role-based or feature-based installation. Then choose Next.
6. For Server Selection, choose Select a server from the server pool. Then choose Next.
7. For Server Roles, do the following:
   b. For Add features that are required for Active Directory Certificate Services, choose Add Features.
   c. Choose Next to finish selecting server roles.
8. For Features, accept the defaults, and then choose Next.
9. For AD CS, do the following:
   a. Choose Next.
   b. Select Certification Authority, and then choose Next.
10. For Confirmation, read the confirmation information, and then choose Install. Do not close the window.
11. Choose the highlighted Configure Active Directory Certificate Services on the destination server link.
12. For Credentials, verify or change the credentials displayed. Then choose Next.
13. For Role Services, select Certification Authority. Then choose Next.
14. For Setup Type, select Standalone CA. Then choose Next.
15. For CA Type, select Root CA. Then choose Next.

   Note
   You can choose to create a root CA or a subordinate CA based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.
16. For Private Key, select Create a new private key. Then choose Next.
17. For Cryptography, do the following:
   a. For Select a cryptographic provider, choose one of the Cavium Key Storage Provider options from the menu. These are the AWS CloudHSM key storage providers. For example, you can choose RSA#Cavium Key Storage Provider.
b. For **Key length**, choose one of the key length options.

c. For **Select the hash algorithm for signing certificates issued by this CA**, choose one of the hash algorithm options.

Choose **Next**.

18. For **CA Name**, do the following:

   a. (Optional) Edit the common name.

   b. (Optional) Type a distinguished name suffix.

   Choose **Next**.

19. For **Validity Period**, specify a time period in years, months, weeks, or days. Then choose **Next**.

20. For **Certificate Database**, you can accept the default values, or optionally change the location for the database and the database log. Then choose **Next**.

21. For **Confirmation**, review the information about your CA; Then choose **Configure**.

22. Choose **Close**, and then choose **Close** again.

You now have a Windows Server CA with AWS CloudHSM. To learn how to sign a certificate signing request (CSR) with your CA, go to **Sign a CSR (p. 301)**.

### Windows Server CA Step 3: Sign a Certificate Signing Request (CSR) with Your Windows Server CA with AWS CloudHSM

You can use your Windows Server CA with AWS CloudHSM to sign a certificate signing request (CSR). To complete these steps, you need a valid CSR. You can create a CSR in several ways, including the following:

- Using OpenSSL
- Using the Windows Server Internet Information Services (IIS) Manager
- Using the certificates snap-in in the Microsoft Management Console
- Using the `certreq` command line utility on Windows

The steps for creating a CSR are outside the scope of this tutorial. When you have a CSR, you can sign it with your Windows Server CA.

### To sign a CSR with your Windows Server CA

1. If you haven't already done so, connect to your Windows server. For more information, see [Connect to Your Instance](#) in the Amazon EC2 User Guide for Windows Instances.

2. On your Windows server, start **Server Manager**.

3. In the **Server Manager** dashboard, in the top right corner, choose **Tools**, **Certification Authority**.

4. In the **Certification Authority** window, choose your computer name.

5. From the **Action** menu, choose **All Tasks**, **Submit new request**.

6. Select your CSR file, and then choose **Open**.

7. In the **Certification Authority** window, double-click **Pending Requests**.

8. Select the pending request. Then, from the **Action** menu, choose **All Tasks**, **Issue**.
9. In the **Certification Authority** window, double-click **Issued Requests** to view the signed certificate.

10. (Optional) To export the signed certificate to a file, complete the following steps:
   a. In the **Certification Authority** window, double-click the certificate.
   b. Choose the **Details** tab, and then choose **Copy to File**.
   c. Follow the instructions in the **Certificate Export Wizard**.

You now have a Windows Server CA with AWS CloudHSM, and a valid certificate signed by the Windows Server CA.

Oracle Database Transparent Data Encryption (TDE) with AWS CloudHSM

Some versions of Oracle's database software offer a feature called Transparent Data Encryption (TDE). With TDE, the database software encrypts data before storing it on disk. The data in the database's table columns or table spaces is encrypted with a table key or table space key. These keys are encrypted with the TDE master encryption key. You can store the TDE master encryption key in the HSMs in your AWS CloudHSM cluster, which provides additional security.

In this solution, you use Oracle Database installed on an Amazon EC2 instance. Oracle Database integrates with the [AWS CloudHSM software library for PKCS #11](#) to store the TDE master key in the HSMs in your cluster.
**Important**
You cannot use an Oracle instance in Amazon Relational Database Service (Amazon RDS) to integrate with AWS CloudHSM. You must install Oracle Database on an Amazon EC2 instance.

Complete the following steps to accomplish Oracle TDE integration with AWS CloudHSM.

**To configure Oracle TDE integration with AWS CloudHSM**
1. Follow the steps in Set Up Prerequisites (p. 303) to prepare your environment.
2. Follow the steps in Configure the Database (p. 304) to configure Oracle Database to integrate with your AWS CloudHSM cluster.

**Oracle TDE with AWS CloudHSM: Set Up the Prerequisites**

To accomplish Oracle TDE integration with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running the Amazon Linux operating system with the following software installed:
  - The AWS CloudHSM client and command line tools.
  - The AWS CloudHSM software library for PKCS #11.
  - Oracle Database. AWS CloudHSM supports Oracle TDE integration with Oracle Database versions 11 and 12.
  - A cryptographic user (CU) to own and manage the TDE master encryption key on the HSMs in your cluster.

Complete the following steps to set up all of the prerequisites.

**To set up the prerequisites for Oracle TDE integration with AWS CloudHSM**
1. Complete the steps in Getting Started (p. 15). After you do so, you'll have an active cluster with one HSM. You will also have an Amazon EC2 instance running the Amazon Linux operating system. The AWS CloudHSM client and command line tools will also be installed and configured.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 42).
3. Connect to your Amazon EC2 client instance and do the following:
   a. Install the AWS CloudHSM software library for PKCS #11 (p. 214).
   b. Install Oracle Database. For more information, see the Oracle Database documentation. AWS CloudHSM supports Oracle TDE integration with Oracle Database versions 11 and 12.
   c. Start the AWS CloudHSM client (p. 78).
   d. Update the configuration file for the cloudhsm_mgmt_util command line tool (p. 79).
   e. Use the cloudhsm_mgmt_util command line tool to create a cryptographic user (CU) on your cluster. For more information, see Managing HSM Users (p. 52).

After you complete these steps, you can Configure the Database (p. 304).
Oracle TDE with AWS CloudHSM: Configure the Database and Generate the Master Encryption Key

To integrate Oracle TDE with your AWS CloudHSM cluster, see the following topics:

1. Update the Oracle Database Configuration (p. 304) to use the HSMs in your cluster as the external security module. For information about external security modules, see Introduction to Transparent Data Encryption in the Oracle Database Advanced Security Guide.

2. Generate the Oracle TDE Master Encryption Key (p. 305) on the HSMs in your cluster.

Topics
- Update the Oracle Database Configuration (p. 304)
- Generate the Oracle TDE Master Encryption Key (p. 305)

Update the Oracle Database Configuration

To update the Oracle Database configuration to use an HSM in your cluster as the external security module, complete the following steps. For information about external security modules, see  Introduction to Transparent Data Encryption in the Oracle Database Advanced Security Guide.

To update the Oracle configuration

1. Connect to your Amazon EC2 client instance. This is the instance where you installed Oracle Database.

2. Make a backup copy of the file named sqlnet.ora. For the location of this file, see the Oracle documentation.

3. Use a text editor to edit the file named sqlnet.ora. Add the following line. If an existing line in the file begins with encryption_wallet_location, replace the existing line with the following one.

   ```
   encryption_wallet_location=(source=(method=hsm))
   ```

   Save the file.

4. Run the following command to create the directory where Oracle Database expects to find the library file for the AWS CloudHSM PKCS #11 software library.

   ```
   sudo mkdir -p /opt/oracle/ extapi/64/hsm
   ```

5. Use one of the following commands to copy the AWS CloudHSM software library for PKCS #11 file to the directory that you created in the previous step.

   - If you installed the PKCS #11 library without Redis, run the following command.

     ```
     sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11_standard.so /opt/oracle/ extapi/64/hsm/
     ```

   - If you installed the PKCS #11 library with Redis, run the following command.

     ```
     sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11_redis.so /opt/oracle/ extapi/64/hsm/
     ```

Note
The /opt/oracle/ extapi/64/hsm directory must contain only one library file. Copy only the library file that corresponds to the way you installed the PKCS #11 library (p. 215). If additional files exist in that directory, remove them.
6. Run the following command to change the ownership of the `/opt/oracle` directory and everything inside it.

```
sudo chown -R oracle:dba /opt/oracle
```

7. Start the Oracle Database.

**Generate the Oracle TDE Master Encryption Key**

To generate the Oracle TDE master key on the HSMs in your cluster, complete the steps in the following procedure.

**To generate the master key**

1. Use the `sqlplus` command to open Oracle SQL*Plus. When prompted, type the system password that you set when you installed Oracle Database.

2. Run the SQL statement that creates the master encryption key, as shown in the following examples. Use the statement that corresponds to your version of Oracle Database. Replace `<CU user name>` with the user name of the cryptographic user (CU). Replace `<password>` with the CU password.

   **Important**
   Run the following command only once. Each time the command is run, it creates a new master encryption key.

   - For Oracle Database version 11, run the following SQL statement.

     ```sql
     SQL> alter system set encryption key identified by "<CU user name>:<password>";
     ```

   - For Oracle Database version 12, run the following SQL statement.

     ```sql
     SQL> administer key management set key identified by "<CU user name>:<password>";
     ```

   If the response is `System altered` or `keystore altered`, then you successfully generated and set the master key for Oracle TDE.

3. (Optional) Run the following command to verify the status of the Oracle wallet.

   ```sql
   SQL> select * from v$encryption_wallet;
   ```

   If the wallet is not open, use one of the following commands to open it. Replace `<CU user name>` with the name of the cryptographic user (CU). Replace `<password>` with the CU password.

   - For Oracle 11, run the following command to open the wallet.

     ```sql
     SQL> alter system set encryption wallet open identified by "<CU user name>:<password>";
     ```

   To manually close the wallet, run the following command.

     ```sql
     SQL> alter system set encryption wallet close identified by "<CU user name>:<password>";
     ```

   - For Oracle 12, run the following command to open the wallet.

     ```sql
     SQL> administer key management set keystore open identified by "<CU user name>:<password>";
     ```
To manually close the wallet, run the following command.

```sql
SQL> administer key management set keystore close identified by "<CU user name>:<password>";
```

### Use Microsoft SignTool with AWS CloudHSM to Sign Files

In cryptography and public key infrastructure (PKI), digital signatures are used to confirm that data has been sent by a trusted entity. Signatures also indicate that the data has not been tampered with in transit. A signature is an encrypted hash that is generated with the sender's private key. The receiver can verify the data's integrity by decrypting its hash signature with the sender's public key. In turn, it is the sender's responsibility to maintain a digital certificate. The digital certificate demonstrates the sender's ownership of the private key and provides the recipient with the public key that is needed for decryption. As long as the private key is owned by the sender, the signature can be trusted. AWS CloudHSM provides secure FIPS 140-2 level 3 validated hardware for you to secure these keys with exclusive single-tenant access.

Many organizations use Microsoft SignTool, a command line tool that signs, verifies, and timestamps files to simplify the code signing process. You can use AWS CloudHSM to securely store your key pairs until they are needed by SignTool, thus creating an easily automatable workflow for signing data.

The following topics provide an overview of how to use SignTool with AWS CloudHSM:

**Topics**
- Microsoft SignTool with AWS CloudHSM Step 1: Set Up the Prerequisites (p. 306)
- Microsoft SignTool with AWS CloudHSM Step 2: Create a Signing Certificate (p. 307)
- Microsoft SignTool with AWS CloudHSM Step 3: Sign a File (p. 308)

### Microsoft SignTool with AWS CloudHSM Step 1: Set Up the Prerequisites

To use Microsoft SignTool with AWS CloudHSM, you need the following:

- An Amazon EC2 client instance running a Windows operating system.
- A certificate authority (CA), either self-maintained or established by a third-party provider.
- An active AWS CloudHSM cluster in the same virtual public cloud (VPC) as your EC2 instance. The cluster must contain at least one HSM.
- A crypto user (CU) to own and manage keys in the AWS CloudHSM cluster.
- An unsigned file or executable.
- The Microsoft Windows Software Development Kit (SDK).

**To set up the prerequisites for using AWS CloudHSM with Windows SignTool**

1. Follow the instructions in the Getting Started (p. 15) section of this guide to launch a Windows EC2 instance and an AWS CloudHSM cluster.
2. If you would like to host your own Windows Server CA, follow steps 1 and 2 in Configuring Windows Server as a Certificate Authority with AWS CloudHSM (p. 298). Otherwise, continue to use your publicly trusted third-party CA.

3. Download and install one of the following versions of the Microsoft Windows SDK on your Windows EC2 instance:
   - Microsoft Windows SDK 10
   - Microsoft Windows SDK 8.1
   - Microsoft Windows SDK 7

The SignTool executable is part of the Windows SDK Signing Tools for Desktop Apps installation feature. You can omit the other features to be installed if you don’t need them. The default installation location is:

```
C:\Program Files (x86)\Windows Kits\<SDK version>\bin\<version number>\<CPU architecture>\signtool.exe
```

You can now use the Microsoft Windows SDK, your AWS CloudHSM cluster, and your CA to Create a Signing Certificate (p. 307).

**Microsoft SignTool with AWS CloudHSM Step 2: Create a Signing Certificate**

Now that you've downloaded the Windows SDK on to your EC2 instance, you can use it to generate a certificate signing request (CSR). The CSR is an unsigned certificate that is eventually passed to your CA for signing. In this example, we use the `certreq` executable that's included with the Windows SDK to generate the CSR.

**To generate a CSR using the certreq executable**

1. If you haven’t already done so, connect to your Windows EC2 instance. For more information, see Connect to Your Instance in the Amazon EC2 User Guide for Windows Instances.
2. Create a file called `request.inf` that contains the lines below. Replace the `Subject` information with that of your organization. For an explanation of each parameter, see Microsoft's documentation.

```ini
[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. Run `certreq.exe`. For this example, we save the CSR as `request.csr`.

```
certreq.exe -new request.inf request.csr
```
Internally, a new key pair is generated on your AWS CloudHSM cluster, and the pair's private key is used to create the CSR.

4. Submit the CSR to your CA. If you are using a Windows Server CA, follow these steps:
   a. Enter the following command to open the CA tool:
   
   ```
   certsrv.msc
   ```
   
   b. In the new window, right-click the CA server's name. Choose **All Tasks**, and then choose **Submit new request**.
   
   c. Navigate to request.csr's location and choose **Open**.
   
   d. Navigate to the **Pending Requests** folder by expanding the **Server CA** menu. Right-click on the request you just created, and under **All Tasks** choose **Issue**.
   
   e. Now navigate to the **Issued Certificates** folder (above the **Pending Requests** folder).
   
   f. Choose **Open** to view the certificate, and then choose the **Details** tab.
   
   g. Choose **Copy to File** to start the Certificate Export Wizard. Save the DER-encoded X.509 file to a secure location as signedCertificate.cer.
   
   h. Exit the CA tool and use the following command, which moves the certificate file to the Personal Certificate Store in Windows. It can then be used by other applications.
   
   ```
   certreq.exe -accept signedCertificate.cer
   ```
   
   You can now use your imported certificate to **Sign a File** (p. 308).

**Microsoft SignTool with AWS CloudHSM Step 3: Sign a File**

You are now ready to use SignTool and your imported certificate to sign your example file. In order to do so, you need to know the certificate's SHA-1 hash, or **thumbprint**. The thumbprint is used to ensure that SignTool only uses certificates that are verified by AWS CloudHSM. In this example, we use PowerShell to get the certificate's hash. You can also use the CA's GUI or the Windows SDK's certutil executable.

**To obtain a certificate's thumbprint and use it to sign a file**

1. Open PowerShell as an administrator and run the following command:

   ```
   Get-ChildItem -path cert:LocalMachine\My
   ```

   Copy the **Thumbprint** that is returned.

   ![Thumbprint](image)

   2. Navigate to the directory within PowerShell that contains SignTool.exe. The default location is `C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64`. 
   
   308
3. Finally, sign your file by running the following command. If the command is successful, PowerShell
returns a success message.

```
signtool.exe sign /v /fd sha256 /sha1 <thumbprint> /sm /as C:\Users\Administrator
\Desktop\<test>.ps1
```

4. (Optional) To verify the signature on the file, use the following command:

```
signtool.exe verify /v /pa C:\Users\Administrator\Desktop\<test>.ps1
```

### Other Third-party Vendor Integrations

Several third-party vendors support AWS CloudHSM as a root of trust. This means that you can utilize
a software solution of your choice while creating and storing the underlying keys in your CloudHSM
cluster. As a result, your workload in AWS can rely on the latency, availability, reliability, and elasticity
benefits of CloudHSM. The following list includes third-party vendors that support CloudHSM.

**Note**
AWS does not endorse or vouch for any third-party vendor.

- **Hashicorp Vault** is a secrets management tool designed to enable collaboration and governance
  across organizations. It supports AWS Key Management Service and AWS CloudHSM as roots of trust
  for additional protection.
- **Thycotic Secrets Server** helps customers manage sensitive credentials across privileged accounts. It
  supports AWS CloudHSM as a root of trust.
- **P6R's KMIP adapter** allows you to utilize your AWS CloudHSM instances through a standard KMIP
  interface.
- **PrimeKey EJBCA** is a popular open source solution for PKI. It allows you to create and store key pairs
  securely with AWS CloudHSM.
- **Box KeySafe** provides encryption key management for cloud content to many organizations with strict
  security, privacy, and regulatory compliance requirements. Customers can further secure KeySafe keys
directly in AWS Key Management Service or indirectly in AWS CloudHSM via AWS KMS Custom Key
  Store.
- **Gemalto KeySecure** is a centralized key management platform, and supports AWS CloudHSM as a root
  of trust.
- **Vormetric Data Security Platform** enables encryption, tokenization, and key management, and
  supports AWS CloudHSM as a root of trust.
- **Insyde Software** supports AWS CloudHSM as a root of trust for firmware signing.
- **F5 BIG-IP LTM** supports AWS CloudHSM as a root of trust.
- **Cloudera Navigator Key HSM** allows you to use your CloudHSM cluster to create and store keys for
  Cloudera Navigator Key Trustee Server.
Monitoring AWS CloudHSM Logs

AWS CloudHSM is integrated with the following AWS services to provide different kinds of logs.

**AWS CloudTrail for API logs**

AWS CloudHSM is integrated with AWS CloudTrail, a service that records all AWS CloudHSM API calls in your AWS account. CloudTrail records these calls in log files that are delivered to an Amazon Simple Storage Service (Amazon S3) bucket of your choice. For example, when you create and delete AWS CloudHSM clusters, create and delete HSMs in a cluster, tag AWS CloudHSM resources, and more, the corresponding API calls are recorded in CloudTrail log files.

**Amazon CloudWatch Logs for HSM Audit Logs**

AWS CloudHSM sends the audit logs recorded by your HSM instances to Amazon CloudWatch Logs, a service that stores, organizes, and displays log data from multiple sources. For example, when you create and delete HSM users, change user passwords, create and delete keys, and more, these events are collected and stored in CloudWatch Logs.

For more information, see the following topics.

**Topics**

- Getting AWS CloudHSM Client Logs (p. 310)
- Logging AWS CloudHSM API Calls with AWS CloudTrail (p. 311)
- Monitoring AWS CloudHSM Audit Logs in Amazon CloudWatch Logs (p. 313)

Getting AWS CloudHSM Client Logs

You can retrieve the logs generated by the AWS CloudHSM client. These logs contain detailed information from the AWS CloudHSM client daemon. The location of the logs depends on the operating system of the Amazon EC2 client instance where you run the AWS CloudHSM client.

**Amazon Linux**

In Amazon Linux, the AWS CloudHSM client logs are written to the file named `/opt/cloudhsm/run/cloudhsm_client.log`. You can use `logrotate` or a similar tool to rotate and manage these logs.

**Amazon Linux 2**

In Amazon Linux 2, the AWS CloudHSM Client logs are collected and stored in the `journal`. You can use `journalctl` to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

**CentOS 6**

In CentOS 6, the AWS CloudHSM client logs are written to the file named `/opt/cloudhsm/run/cloudhsm_client.log`. You can use `logrotate` or a similar tool to rotate and manage these logs.
CentOS 7

In CentOS 7, the AWS CloudHSM Client logs are collected and stored in the `journal`. You can use `journalctl` to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

RHEL 6

In Red Hat Enterprise Linux 6, the AWS CloudHSM client logs are written to the file named `/opt/cloudhsm/run/cloudhsm_client.log`. You can use `logrotate` or a similar tool to rotate and manage these logs.

RHEL 7

In Red Hat Enterprise Linux 7, the AWS CloudHSM Client logs are collected and stored in the `journal`. You can use `journalctl` to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

Ubuntu 16.04

In Ubuntu, the AWS CloudHSM Client logs are collected and stored in the `journal`. You can use `journalctl` to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

Windows

- For Windows client 1.1.2+:
  
  AWS CloudHSM client logs are written to a `cloudhsm.log` file in the AWS CloudHSM program files folder (`C:\Program Files\Amazon\CloudHSM\`). Each log file name is suffixed with a timestamp indicating when the AWS CloudHSM client was started.

- For Windows client 1.1.1 and older:
  
  The client logs are not written to a file. The logs are displayed at the command prompt or in the PowerShell window where you started the AWS CloudHSM client.

Logging AWS CloudHSM API Calls with AWS CloudTrail

AWS CloudHSM is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS CloudHSM. CloudTrail captures all API calls for AWS CloudHSM as events. The calls captured include calls from the AWS CloudHSM console and code calls to the AWS CloudHSM API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS CloudHSM. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to AWS CloudHSM, the IP address from which the request was made, who made the request, when it was made, and additional details.
AWS CloudHSM Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS CloudHSM, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS CloudHSM, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

CloudTrail logs all AWS CloudHSM operations, including read-only operations, such as DescribeClusters and ListTags, and management operations, such as InitializeCluster, CreateHsm, and DeleteBackup.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding AWS CloudHSM Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the AWS CloudHSM CreateHsm action.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAJZVM5NEGZSTCITAMM:ExampleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/AdminRole/ExampleSession",
```
Monitoring AWS CloudHSM Audit Logs in Amazon CloudWatch Logs

When an HSM in your account receives a command from the AWS CloudHSM command line tools (p. 75) or software libraries (p. 214), it records its execution of the command in audit log form. The HSM audit logs include all client-initiated management commands (p. 327), including those that create and delete the HSM, log into and out of the HSM, and manage users and keys. These logs provide a reliable record of actions that have changed the state of the HSM.

AWS CloudHSM collects your HSM audit logs and sends them to Amazon CloudWatch Logs on your behalf. You can use the features of CloudWatch Logs to manage your AWS CloudHSM audit logs, including searching and filtering the logs and exporting log data to Amazon S3. You can work with your HSM audit logs in the Amazon CloudWatch console or use the CloudWatch Logs commands in the AWS CLI and CloudWatch Logs SDKs.

Topics
How Audit Logging Works

Audit logging is automatically enabled in all AWS CloudHSM clusters. It cannot be disabled or turned off, and no settings can prevent AWS CloudHSM from exporting the logs to CloudWatch Logs. Each log event has a time stamp and sequence number that indicate the order of events and help you detect any log tampering.

Each HSM instance generates its own log. The audit logs of various HSMs, even those in the same cluster, are likely to differ. For example, only the first HSM in each cluster records initialization of the HSM. Initialization events do not appear in the logs of HSMs that are cloned from backups. Similarly, when you create a key, the HSM that generates the key records a key generation event. The other HSMs in the cluster record an event when they receive the key via synchronization.

AWS CloudHSM collects the logs and posts them to CloudWatch Logs in your account. To communicate with the CloudWatch Logs service on your behalf, AWS CloudHSM uses a service-linked role (p. 336). The IAM policy that is associated with the role allows AWS CloudHSM to perform only the tasks required to send the audit logs to CloudWatch Logs.

**Important**
If you created a cluster before January 20, 2018, and have not yet created an attached service-linked role, you must manually create one. This is necessary for CloudWatch to receive audit logs from your AWS CloudHSM cluster. For more information about service-linked role creation, see Understanding Service-Linked Roles (p. 336), as well as Creating a Service-Linked Role in the IAM User Guide.

Viewing Audit Logs in CloudWatch Logs

Amazon CloudWatch Logs organizes the audit logs into log groups and, within a log group, into log streams. Each log entry is an event. AWS CloudHSM creates one log group for each cluster and one log stream for each HSM in the cluster. You do not have to create any CloudWatch Logs components or change any settings.

- The log group name is `/aws/cloudhsm/<cluster ID>`; for example `/aws/cloudhsm/cluster-likphkxygsn`. When you use the log group name in a AWS CLI or PowerShell command, be sure to enclose it in double quotation marks.
- The log stream name is the HSM ID; for example, `hsm-nwbbigbj4jk`.

In general, there is one log stream for each HSM. However, any action that changes the HSM ID, such as when an HSM fails and is replaced, creates a new log stream.

For more information about CloudWatch Logs concepts, see Concepts in the Amazon CloudWatch Logs User Guide.

You can view the audit logs for an HSM from the CloudWatch Logs page in the AWS Management Console, the CloudWatch Logs commands in the AWS CLI, the CloudWatch Logs PowerShell cmdlets, or the CloudWatch Logs SDKs. For instructions, see View Log Data in the Amazon CloudWatch Logs User Guide.

For example, the following image shows the log group for the `cluster-likphkxygsn` cluster in the AWS Management Console.
When you choose the cluster log group name, you can view the log stream for each of the HSMs in the cluster. The following image shows the log streams for the HSMs in the `cluster-likphkxygsn` cluster.

When you choose an HSM log stream name, you can view the events in the audit log. For example, this event, which has a sequence number of 0x0 and an Opcode of `CN_INIT_TOKEN`, is typically the first event for the first HSM in each cluster. It records the initialization of the HSM in the cluster.
You can use all the many features in CloudWatch Logs to manage your audit logs. For example, you can use the Filter events feature to find particular text in an event, such as the `CN_CREATE_USER` Opcode.

To find all events that do not include the specified text, add a minus sign (-) before the text. For example, to find events that do not include `CN_CREATE_USER`, enter `-CN_CREATE_USER`. 
Interpreting HSM Audit Logs

The events in the HSM audit logs have standard fields. Some event types have additional fields that capture useful information about the event. For example, user login and user management events include the user name and user type of the user. Key management commands include the key handle.

Several of the fields provide particularly important information. The Opcode identifies the management command that is being recorded. The Sequence No identifies an event in the log stream and indicates the order in which it was recorded.

For example, the following example event is the second event (Sequence No: 0x1) in the log stream for an HSM. It shows the HSM generating a password encryption key, which is part of its startup routine.

```
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 : 0x10140006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : tester
User Type : CN_CRYPTO_USER (1)
```

The following fields are common to every AWS CloudHSM event in the audit log.
Time
The time that the event occurred in the UTC time zone. The time is displayed as a human-readable

time and Unix time in microseconds.

Reboot counter
A 32-bit persistent ordinal counter that is incremented when the HSM hardware is rebooted.

All events in a log stream have the same reboot counter value. However, the reboot counter might
not be unique to a log stream, as it can differ across different HSM instances in the same cluster.

Sequence No
A 64-bit ordinal counter that is incremented for each log event. The first event in each log stream
has a sequence number of 0x0. There should be no gaps in the Sequence No values. The sequence
number is unique only within a log stream.

Command type
A hexadecimal value that represents the category of the command. Commands in the AWS
CloudHSM log streams have a command type of CN_MGMT_CMD (0x0) or CN_CERT_AUTH_CMD (0x9).

Opcode
Identifies the management command that was executed. For a list of Opcode values in the AWS
CloudHSM audit logs, see Audit Log Reference (p. 327).

Session handle
Identifies the session in which the command was run and the event was logged.

Response
Records the response to the management command. You can search the Response field for
SUCCESS and ERROR values.

Log type
Indicates the log type of the AWS CloudHSM log that recorded the command.

• MINIMAL_LOG_ENTRY (0)
• MGMT_KEY_DETAILS_LOG (1)
• MGMT_USER_DETAILS_LOG (2)
• GENERIC_LOG

Examples of Audit Log Events
The events in a log stream record the history of the HSM from its creation to deletion. You can use the
log to review the lifecycle of your HSMs and gain insight into its operation. When you interpret the
events, note the Opcode, which indicates the management command or action, and the Sequence No,
which indicates the order of events.

Topics
• Example: Initialize the First HSM in a Cluster (p. 319)
• Login and Logout Events (p. 320)
• Example: Create and Delete Users (p. 319)
• Example: Create and Delete a Key Pair (p. 322)
• Example: Generate and Synchronize a Key (p. 323)
• Example: Export a Key (p. 325)
• Example: Import a Key (p. 326)
• Example: Share and Unshare a Key (p. 327)
Example: Initialize the First HSM in a Cluster

The audit log stream for the first HSM in each cluster differs significantly from the log streams of other HSMs in the cluster. The audit log for the first HSM in each cluster records its creation and initialization. The logs of additional HSMs in the cluster, which are generated from backups, begin with a login event.

**Important**

The following initialization entries will not appear in the CloudWatch logs of clusters initialized before the release of the CloudHSM audit logging feature (August 30, 2018). For more information, see Document History (p. 379).

The following example events appear in the log stream for the first HSM in a cluster. The first event in the log — the one with Sequence No 0x0 — represents the command to initialize the HSM (CN_INIT_TOKEN). The response indicates that the command was successful (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) |

The second event in this example log stream (Sequence No 0x1) records the command to create the password encryption key that the HSM uses (CN_GEN_PSWD_ENC_KEY).

This is a typical startup sequence for the first HSM in each cluster. Because subsequent HSMs in the same cluster are clones of the first one, they use the same password encryption key.

| 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) |

The third event in this example log stream (Sequence No 0x2) is the creation of the *appliance user (AU)* (p. 11), which is the AWS CloudHSM service. Events that involve HSM users include extra fields for the user name and user type.

| 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) |

The fourth event in this example log stream (Sequence No 0x3) records the CN_INIT_DONE event, which completes the initialization of the HSM.

| Time: 12/19/17 21:01:17.298914, usecs:1513717277298914 |
| Sequence No : 0x3 |
| Reboot counter : 0xe8 |
You can follow the remaining events in the startup sequence. These events might include several login and logout events, and the generation of the key encryption key (KEK). The following event records the command that changes the password of the **precrypto officer (PRECO)** (p. 11). This command activates the cluster.

```
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)
```

**Login and Logout Events**

When interpreting your audit log, note events that record users logging and in and out of the HSM. These events help you to determine which user is responsible for management commands that appear in sequence between the login and logout commands.

For example, this log entry records a login by a crypto officer named **admin**. The sequence number, 0x0, indicates that this is the first event in this log stream.

```
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)
```

The following example event records the **admin** crypto officer logging out. The sequence number, 0x2, indicates that this is the third event in the log stream.

```
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)
```

If the logged in user closes the session without logging out, the log stream includes an **CN_APP_FINALIZE** or close session event (**CN_SESSION_CLOSE**), instead of a **CN_LOGOUT** event. Unlike the login event, this logout event typically is recorded only by the HSM that executes the command.

```
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)

If a login attempt fails because the user name is invalid, the HSM records a CN_LOGIN event with the user name and type provided in the login command. The response displays error message 157, which explains that the user name does not exist.

Time: 01/24/18 17:41:39.037255, 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)

Example: Create and Delete Users

This example shows the log events that are recorded when a crypto officer (CO) creates and deletes users.

The first event records a CO, admin, logging into the HSM. The sequence number of 0x0 indicates that this is the first event in the log stream. The name and type of the user who logged in are included in the event.

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 : 0xc07014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USERDETAILS_LOG (2)
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)

The next event in the log stream (sequence 0x1) records the CO creating a new crypto user (CU). The name and type of the new user are included in the event.

Time: 01/16/18 01:49:39.437708, usecs:1516067379437708
Sequence No : 0x1
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Then, the CO creates another crypto officer, alice. The sequence number indicates that this action followed the previous one with no intervening actions.

Later, the CO named admin logs in and deletes the crypto officer named alice. The HSM records a CN_DELETE_USER event. The name and type of the deleted user are included in the event.

Example: Create and Delete a Key Pair

This example shows the events that are recorded in an HSM audit log when you create and delete a key pair.

The following event records the crypto user (CU) named crypto_user logging in to the HSM.

Next, the CU generates a key pair (CN_GENERATE_KEY_PAIR). The private key has key handle 131079. The public key has key handle 131078.
Interpreting HSM Audit Logs

The CU immediately deletes the key pair. A CN_DESTROY_OBJECT event records the deletion of the public key (131078).

```
Time: 12/13/17 23:09:04.813977, usecs:1513206544813977
Sequence No: 0x2a
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x2a)
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
```

Then, a second CN_DESTROY_OBJECT event records the deletion of the private key (131079).

```
Time: 12/13/17 23:09:04.815530, usecs:1513206544815530
Sequence No: 0x2b
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x2b)
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
```

Finally, the CU logs out.

```
Time: 12/13/17 23:09:04.817222, usecs:1513206544817222
Sequence No: 0x2c
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x2c)
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)
```

Example: Generate and Synchronize a Key

This example shows the effect of creating a key in a cluster with multiple HSMs. The key is generated on one HSM, extracted from the HSM as a masked object, and inserted in the other HSMs as a masked object.

Note

The client tools might fail to synchronize the key. Or the command might include the `min_srv` parameter, which synchronizes the key only to the specified number of HSMs. In either case, the AWS CloudHSM service synchronizes the key to the other HSMs in the cluster. Because the HSMs record only client-side management commands in their logs, the server-side synchronization is not recorded in the HSM log.

First consider the log stream of the HSM that receives and executes the commands. The log stream is named for HSM ID, `hsm-abcde123456`, but the HSM ID does not appear in the log events.
First, the testuser crypto user (CU) logs in to the hsm-abcde123456 HSM.

| 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) |

The CU runs an exSymKey (p. 158) command to generate a symmetric key. The hsm-abcde123456 HSM generates a symmetric key with a key handle of 262152. The HSM records a CN_GENERATE_KEY event in its log.

| 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 |

The next event in the log stream for hsm-abcde123456 records the first step in the key synchronization process. The new key (key handle 262152) is extracted from the HSM as a masked object.

| 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 |

Now consider the log stream for HSM hsm-zyxwv987654, another HSM in the same cluster. This log stream also includes a login event for the testuser CU. The time value shows that occurs shortly after the user logs in to the hsm-abcde123456 HSM.

| 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) |

This log stream for this HSM does not have a CN_GENERATE_KEY event. But it does have an event that records synchronization of the key to this HSM. The CN_INSERT_MASKED_OBJECT_USER event records the receipt of key 262152 as a masked object. Now key 262152 exists on both HSMs in the cluster.
When the CU user logs out, this CN_LOGOUT event appears only in the log stream of the HSM that received the commands.

**Example: Export a Key**

This example shows the audit log events that are recorded when a crypto user (CU) exports keys from a cluster with multiple HSMs.

The following event records the CU (testuser) logging into key_mgmt_util (p. 122).

```
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)
```

The CU runs an exSymKey (p. 136) command to export key 7, a 256-bit AES key. The command uses key 6, a 256-bit AES key on the HSMs, as the wrapping key.

The HSM that receives the command records a CN_WRAP_KEY event for key 7, the key that is being exported.

```
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
```

Then, the HSM records a CN_NIST_AES_WRAP event for the wrapping key, key 6. The key is wrapped and then immediately unwrapped, but the HSM records only one event.

```
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

The `exSymKey` command writes the exported key to a file but does not change the key on the HSM. Consequently, there are no corresponding events in the logs of other HSMs in the cluster.

**Example: Import a Key**

This example shows the audit log events that are recorded when you import keys into the HSMs in a cluster. In this example, the crypto user (CU) uses the `imSymKey` command to import an AES key into the HSMs. The command uses key 6 as the wrapping key.

The HSM that receives the commands first records a `CN_NIST_AES_WRAP` event for key 6, the wrapping key.

```
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
```

Then, the HSM records a `CN_UNWRAP_KEY` event that represents the import operation. The imported key is assigned a key handle of 11.

```
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
```

When a new key is generated or imported, the client tools automatically attempt to synchronize the new key to other HSMs in the cluster. In this case, the HSM records a `CN_EXTRACT_MASKED_OBJECT_USER` event when key 11 is extracted from the HSM as a masked object.

```
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
```

The log streams of other HSMs in the cluster reflect the arrival of the newly imported key.

For example, this event was recorded in the log stream of a different HSM in the same cluster. This `CN_INSERT_MASKED_OBJECT_USER` event records the arrival of a masked object that represents key 11.

```
Time: 01/24/18 19:58:23.286793, usecs:1516823903286793
```
Example: Share and Unshare a Key

This example shows the audit log event that is recorded when a crypto user (CU) shares or unshares ECC private key with other crypto users. The CU uses the `shareKey` (p. 116) command and provides the key handle, the user ID, and the value 1 to share or value 0 to unshare the key.

In the following example, the HSM that receives the command, records a CM_SHARE_OBJECT event that represents the share operation.

```
Time: 02/08/19 19:35:39.480168, usecs:154965439480168
Sequence No : 0x3f
Reboot counter : 0x38
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_MGMT_CMD (0x0)
Session Handle : 0x3014007
Response : 0:HSM Return: SUCCESS
Log type : UNKNOWN_LOG_TYPE (5)
```

Audit Log Reference

AWS CloudHSM records HSM management commands in audit log events. Each event has an operation code (Opcode) value that identifies the action that occurred and its response. You can use the Opcode values to search, sort, and filter the logs.

The following table defines the Opcode values in an AWS CloudHSM audit log.

<table>
<thead>
<tr>
<th>Operation Code (Opcode)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Login:</strong></td>
<td>These events include the user name and user type.</td>
</tr>
<tr>
<td>CN_LOGIN (0xd)</td>
<td>User login (p. 109) (excludes appliance user [AU]).</td>
</tr>
<tr>
<td>CN_LOGOUT (0xe)</td>
<td>User logout (p. 109) (excludes appliance user [AU]).</td>
</tr>
<tr>
<td>CN_APP_FINALIZE</td>
<td>App finalize (logged only when user did not explicitly log out)</td>
</tr>
<tr>
<td>CN_CLOSE_SESSION</td>
<td>Close session (logged only when user did not explicitly log out)</td>
</tr>
<tr>
<td><strong>User Management:</strong></td>
<td>These events include the user name and user type.</td>
</tr>
<tr>
<td>CN_CREATE_USER (0x3)</td>
<td>Create a crypto user (CU) (p. 87)</td>
</tr>
<tr>
<td>CN_CREATE_CO</td>
<td>Create a crypto officer (CO) (p. 87)</td>
</tr>
<tr>
<td>CN_CREATE_APPLIANCE_USER</td>
<td>Create an appliance user (AU) (p. 87)</td>
</tr>
<tr>
<td>CN_DELETE_USER</td>
<td>Delete a user (p. 91)</td>
</tr>
<tr>
<td>Operation Code (Opcode)</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CN_CHANGE_PSWD</td>
<td>Change a user password (p. 84)</td>
</tr>
<tr>
<td>CN_SET_M_VALUE</td>
<td>Set quorum authentication (M of N) for a user action.</td>
</tr>
<tr>
<td>CN_APPROVE_TOKEN</td>
<td>Approve a quorum authentication token for a user action.</td>
</tr>
</tbody>
</table>

**Key Management**: These events include the key handle.

<table>
<thead>
<tr>
<th>Operation Code (Opcode)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN_GENERATE_KEY</td>
<td>Generate a symmetric key (p. 158)</td>
</tr>
<tr>
<td>CN_GENERATE_KEY_PAIR (0x19)</td>
<td>Generate a key pair (DSA (p. 145), ECC (p. 150), or RSA (p. 154))</td>
</tr>
<tr>
<td>CN_CREATE_OBJECT</td>
<td>Import a public key (without wrapping)</td>
</tr>
<tr>
<td>CN_MODIFY_OBJECT</td>
<td>Set a key attribute in key_mgmt_util (p. 190) or cloudhsm_mgmt_util (p. 112).</td>
</tr>
<tr>
<td>CN_DESTROY_OBJECT (0x11)</td>
<td>Delete a key (p. 130)</td>
</tr>
<tr>
<td>CN_TOMBSTONE_OBJECT</td>
<td>Mark the key for deletion, but do not remove it</td>
</tr>
<tr>
<td>CN_SHARE_OBJECT</td>
<td>Share or unshare a key (p. 116)</td>
</tr>
<tr>
<td>CN_WRAP_KEY</td>
<td>Export an encrypted copy of a key (wrapKey (p. 201))</td>
</tr>
<tr>
<td>CN_UNWRAP_KEY</td>
<td>Import an encrypted copy of a key (unwrapKey (p. 195))</td>
</tr>
<tr>
<td>CN_NIST_AES_WRAP</td>
<td>Encrypt or decrypt a file (aesWrapUnwrap (p. 127))</td>
</tr>
<tr>
<td>CN_INSERT_MASKED_OBJECT_USER</td>
<td>Receive a key (as a masked object) from another HSM in the cluster; this event is recorded when a client action synchronizes the key</td>
</tr>
<tr>
<td>CN_EXTRACT_MASKED_OBJECT_USER</td>
<td>Send a key (as a masked object) to other HSMs in the cluster; this event is recorded when a client action synchronizes the key</td>
</tr>
</tbody>
</table>

**Clone HSMs**

<table>
<thead>
<tr>
<th>Operation Code (Opcode)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN_CLONE_SOURCE_INIT</td>
<td>Clone source start</td>
</tr>
<tr>
<td>CN_CLONE_SOURCE_STAGE1</td>
<td>Clone source end</td>
</tr>
<tr>
<td>CN_CLONE_TARGET_INIT</td>
<td>Clone target start</td>
</tr>
<tr>
<td>CN_CLONE_TARGET_STAGE1</td>
<td>Clone target end</td>
</tr>
</tbody>
</table>

**Certificate-Based Authentication**

<table>
<thead>
<tr>
<th>Operation Code (Opcode)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN_CERT_AUTH_STORE_CERT</td>
<td>Store a certificate</td>
</tr>
<tr>
<td>CN_CERT_AUTH_VALIDATE_PEER_CERTS</td>
<td>Validate a certificate</td>
</tr>
<tr>
<td>CN_CERT_AUTH_SOURCE_KEY_EXCHANGE</td>
<td>Source key exchange</td>
</tr>
<tr>
<td>Operation Code (Opcode)</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CN_CERT_AUTH_TARGET_KEY_EXCHANGE</td>
<td>Target key exchange</td>
</tr>
<tr>
<td><strong>HSM Instance Commands</strong></td>
<td></td>
</tr>
<tr>
<td>CN_INIT_TOKEN (0x1)</td>
<td>Initialize the HSM: Start</td>
</tr>
<tr>
<td>CN_INIT_DONE</td>
<td>Initialize the HSM: Complete</td>
</tr>
<tr>
<td>CN_GEN_KEY_ENC_KEY</td>
<td>Generate a key encryption key (KEK)</td>
</tr>
<tr>
<td>CN_GEN_PSWD_ENC_KEY (0x1d)</td>
<td>Generate a password encryption key (PEK)</td>
</tr>
<tr>
<td>CN_CLOSE_PARTITION_SESSIONS</td>
<td>Close a session on the HSM</td>
</tr>
<tr>
<td>CN_STORE_KBK_SHARE</td>
<td>Store the key backup key (KBK)</td>
</tr>
<tr>
<td>CN_SET_NODEID</td>
<td>Set the node ID of the HSM in the cluster</td>
</tr>
<tr>
<td>CN_ZEROIZE</td>
<td>Zeroize the HSM</td>
</tr>
</tbody>
</table>
Getting Metrics

You can retrieve metrics from your AWS CloudHSM environment by getting CloudWatch metrics (p. 330).

Topics
- Getting CloudWatch Metrics (p. 330)

Getting CloudWatch Metrics

AWS CloudHSM publishes metrics about your HSM instances to your CloudWatch dashboard. The metrics can be grouped by region, by cluster ID, and by HSM ID. Note, however, that the HSM ID will change if AWS CloudHSM replaces a failed HSM. We therefore recommend that you alarm and measure on the regional or cluster ID level rather than on the HSM ID. The following metrics are available:

- **HsmUnhealthy**: The HSM instance is not performing properly. AWS CloudHSM automatically replaces unhealthy instances for you. You may choose to proactively expand cluster size to reduce performance impact while we are replacing the HSM.
- **HsmTemperature**: Junction temperature of the hardware processor. The system shuts down if temperature reaches 110 degrees Centigrade.
- **HsmKeysSessionOccupied**: Number of session keys being used by the HSM instance.
- **HsmKeysTokenOccupied**: Number of token keys being used by the HSM instance and the cluster.
- **HsmSslCtxsOccupied**: Number of end-to-end encrypted channels currently established for the HSM instance. Up to 2048 channels are allowed.
- **HsmSessionCount**: Number of open connections to the HSM instance. Up to 2048 are allowed. By default, the client daemon is configured to open two sessions with each HSM instance under one end-to-end encrypted channel.
- **HsmUsersAvailable**: Number of additional users that can be created. This equals the maximum number of users, **HsmUsersMax**, minus the users created to date.
- **HsmUsersMax**: Maximum number of users that can be created on the HSM instance. Currently this is 1024.
- **InterfaceEth2OctetsInput**: Cumulative sum of traffic to the HSM to date. We recommend that you also examine Amazon EC2 instance metrics.
- **InterfaceEth2OctetsOutput**: see the preceding metric - **InterfaceEth2OctetsInput**.
Security in AWS CloudHSM

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS CloudHSM, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS CloudHSM. The following topics show you how to configure AWS CloudHSM to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS CloudHSM resources.

Contents
- Data Protection in AWS CloudHSM (p. 331)
- Identity and Access Management for AWS CloudHSM (p. 332)
- FIPS Validation (p. 337)
- Resilience in AWS CloudHSM (p. 338)
- Infrastructure Security in AWS CloudHSM (p. 338)
- Update Management in AWS CloudHSM (p. 339)

Data Protection in AWS CloudHSM

AWS CloudHSM conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
• Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers’ account numbers, into free-form fields such as a Name field. This includes when you work with AWS CloudHSM or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into AWS CloudHSM or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don’t include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

**Encryption at Rest**

When AWS CloudHSM makes a backup from an HSM, the HSM encrypts its data before sending it to AWS CloudHSM. The data is encrypted using a unique, ephemeral encryption key. For more information, see Security of Backups (p. 7).

**Encryption in Transit**

Communication between the AWS CloudHSM client and the HSMs in your cluster is encrypted from end to end. This communication can be decrypted only by your client and your HSMs.

**Identity and Access Management for AWS CloudHSM**

AWS uses security credentials to identify you and to grant you access to your AWS resources. You can use features of AWS Identity and Access Management (IAM) to allow other users, services, and applications to use your AWS resources fully or in a limited way. You can do this without sharing your security credentials.

By default, IAM users don’t have permission to create, view, or modify AWS resources. To allow an IAM user to access resources such as a load balancer, and to perform tasks, you:

1. Create an IAM policy that grants the IAM user permission to use the specific resources and API actions they need.
2. Attach the policy to the IAM user or the group that the IAM user belongs to.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources.

For example, you can use IAM to create users and groups under your AWS account. An IAM user can be a person, a system, or an application. Then you grant permissions to the users and groups to perform specific actions on the specified resources using an IAM policy.

**Grant Permissions Using IAM Policies**

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources.
An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as shown in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "effect",
        "Action": "action",
        "Resource": "resource-arn",
        "Condition": {
            "condition": {
                "key": "value"
            }
        }
    }]
}
```

- **Effect**— The effect can be Allow or Deny. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action**— The action is the specific API action for which you are granting or denying permission. For more information about specifying action, see API Actions for AWS CloudHSM (p. 333).
- **Resource**— The resource that's affected by the action. AWS CloudHSM does not support resource-level permissions. You must use the * wildcard to specify all AWS CloudHSM resources.
- **Condition**— You can optionally use conditions to control when your policy is in effect. For more information, see Condition Keys for AWS CloudHSM (p. 334).

For more information, see the IAM User Guide.

### API Actions for AWS CloudHSM

In the **Action** element of your IAM policy statement, you can specify any API action that AWS CloudHSM offers. You must prefix the action name with the lowercase string cloudhsm:, as shown in the following example.

```
"Action": "cloudhsm:DescribeClusters"
```

To specify multiple actions in a single statement, enclose them in square brackets and separate them with a comma, as shown in the following example.

```
"Action": [ "cloudhsm:DescribeClusters", "cloudhsm:DescribeHsm" ]
```

You can also specify multiple actions using the * wildcard. The following example specifies all API action names for AWS CloudHSM that start with List.

```
"Action": "cloudhsm:List*"
```

To specify all API actions for AWS CloudHSM, use the * wildcard, as shown in the following example.

```
"Action": "cloudhsm:*"
```

For the list of API actions for AWS CloudHSM, see AWS CloudHSM Actions.
Condition Keys for AWS CloudHSM

When you create a policy, you can specify the conditions that control when the policy is in effect. Each condition contains one or more key-value pairs. There are global condition keys and service-specific condition keys.

AWS CloudHSM has no service-specific context keys.

For more information about global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

Predefined AWS Managed Policies for AWS CloudHSM

The managed policies created by AWS grant the required permissions for common use cases. You can attach these policies to your IAM users, based on the access to AWS CloudHSM that they require:

- **AWSCloudHSMFullAccess** — Grants full access required to use AWS CloudHSM features.
- **AWSCloudHSMReadOnlyAccess** — Grants read-only access to AWS CloudHSM features.

Customer Managed Policies for AWS CloudHSM

We recommend that you create an IAM administrators group for AWS CloudHSM that contains only the permissions required to run AWS CloudHSM. Attach the policy with the appropriate permissions to this group. Add IAM users to the group as needed. Each user that you add inherits the policy from the administrators group.

Also, we recommend that you create additional user groups based on the permissions that your users need. This ensures that only trusted users have access to critical API actions. For example, you could create a user group that you use to grant read-only access to clusters and HSMs. Because this group does not allow a user to delete clusters or HSMs, an untrusted user cannot affect the availability of a production workload.

As new AWS CloudHSM management features are added over time, you can ensure that only trusted users are given immediate access. By assigning limited permissions to policies at creation, you can manually assign new feature permissions to them later.

The following are example policies for AWS CloudHSM. For information about how to create a policy and attach it to an IAM user group, see Creating Policies on the JSON Tab in the IAM User Guide.

Examples

- **Read Only Permissions** (p. 334)
- **Power User Permissions** (p. 335)
- **Admin Permissions** (p. 335)

Example Example: Read-Only Permissions

This policy allows access to the DescribeClusters and DescribeBackups API actions. It also includes additional permissions for specific Amazon EC2 API actions. It does not allow the user to delete clusters or HSMs.

```
Example Example: Power User Permissions

This policy allows access to a subset of the AWS CloudHSM API actions. It also includes additional permissions for specific Amazon EC2 actions. It does not allow the user to delete clusters or HSMs. You must include the `iam:CreateServiceLinkedRole` action to allow AWS CloudHSM to automatically create the `AWSServiceRoleForCloudHSM` service-linked role in your account. This role allows AWS CloudHSM to log events. For more information, see Service-Linked Roles for AWS CloudHSM (p. 336).

```json
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
         "cloudhsm:DescribeBackups",
         "cloudhsm:ListTags",
         "cloudhsm:CreateCluster",
         "cloudhsm:CreateHsm",
         "cloudhsm:RestoreBackup",
         "cloudhsm:CopyBackupToRegion",
         "cloudhsm:InitializeCluster",
         "cloudhsm:ListTags",
         "cloudhsm:TagResource",
         "cloudhsm:UntagResource",
         "ec2:CreateNetworkInterface",
         "ec2:DescribeNetworkInterfaces",
         "ec2:DescribeNetworkInterfaceAttribute",
         "ec2:AttachNetworkInterface",
         "ec2:DeleteNetworkInterface",
         "ec2:CreateSecurityGroup",
         "ec2:AuthorizeSecurityGroupIngress",
         "ec2:AuthorizeSecurityGroupEgress",
         "ec2:RevokeSecurityGroupEgress",
         "ec2:DescribeSecurityGroups",
         "ec2:DeleteSecurityGroup",
         "ec2:CreateTags",
         "ec2:DescribeVpc",
         "ec2:DescribeSubnets",
         "iam:CreateServiceLinkedRole"
      ],
      "Resource": "*
   }
}
```

Example Example: Admin Permissions

This policy allows access to all AWS CloudHSM API actions, including the actions to delete HSMs and clusters. It also includes additional permissions for specific Amazon EC2 actions. You must include the `iam:CreateServiceLinkedRole` action to allow AWS CloudHSM to automatically create the `AWSServiceRoleForCloudHSM` service-linked role in your account. This role allows AWS CloudHSM to log events. For more information, see Service-Linked Roles for AWS CloudHSM (p. 336).

```json
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
         "cloudhsm:DescribeBackups",
         "cloudhsm:ListTags",
         "cloudhsm:CreateCluster",
         "cloudhsm:CreateHsm",
         "cloudhsm:RestoreBackup",
         "cloudhsm:CopyBackupToRegion",
         "cloudhsm:InitializeCluster",
         "cloudhsm:ListTags",
         "cloudhsm:TagResource",
         "cloudhsm:UntagResource",
         "ec2:CreateNetworkInterface",
         "ec2:DescribeNetworkInterfaces",
         "ec2:DescribeNetworkInterfaceAttribute",
         "ec2:AttachNetworkInterface",
         "ec2:DeleteNetworkInterface",
         "ec2:CreateSecurityGroup",
         "ec2:AuthorizeSecurityGroupIngress",
         "ec2:AuthorizeSecurityGroupEgress",
         "ec2:RevokeSecurityGroupEgress",
         "ec2:DescribeSecurityGroups",
         "ec2:DeleteSecurityGroup",
         "ec2:CreateTags",
         "ec2:DescribeVpc",
         "ec2:DescribeSubnets",
         "iam:CreateServiceLinkedRole"
      ],
      "Resource": "*
   }
}
```
Service-Linked Roles for AWS CloudHSM

The IAM policy that you created previously to Customer Managed Policies for AWS CloudHSM (p. 334) includes the iam:CreateServiceLinkedRole action. AWS CloudHSM defines a service-linked role named AWSServiceRoleForCloudHSM. The role is predefined by AWS CloudHSM and includes permissions that AWS CloudHSM requires to call other AWS services on your behalf. The role makes setting up your service easier because you don’t need to manually add the role policy and trust policy permissions.

The role policy allows AWS CloudHSM to create Amazon CloudWatch Logs log groups and log streams and write log events on your behalf. You can view it below and in the IAM console.

```json
{
"Version": "2018-06-12",
"Statement": [
{
"Effect": "Allow",
"Action": [
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:PutLogEvents",
"logs:DescribeLogStreams"
],
"Resource": [
"arn:aws:logs:*:*:*"
]
}
}
```

The trust policy for the AWSServiceRoleForCloudHSM role allows AWS CloudHSM to assume the role.
Creating a Service-Linked Role (Automatic)

AWS CloudHSM creates the `AWSServiceRoleForCloudHSM` role when you create a cluster if you include the `iam:CreateServiceLinkedRole` action in the permissions that you defined when you created the AWS CloudHSM administrators group. See Customer Managed Policies for AWS CloudHSM (p. 334).

If you already have one or more clusters and just want to add the `AWSServiceRoleForCloudHSM` role, you can use the console, the `create-cluster` command, or the `CreateCluster` API operation to create a cluster. Then use the console, the `delete-cluster` command, or the `DeleteCluster` API operation to delete it. Creating the new cluster creates the service-linked role and applies it to all clusters in your account. Alternatively, you can create the role manually. See the following section for more information.

**Note**
You do not need to perform all of the steps outlined in Getting Started with AWS CloudHSM (p. 15) to create a cluster if you are only creating it to add the `AWSServiceRoleForCloudHSM` role.

Creating a Service-Linked Role (Manual)

You can use the IAM console, AWS CLI, or API to create the `AWSServiceRoleForCloudHSM` role. For more information, see Creating a Service-Linked Role in the IAM User Guide.

Editing the Service-Linked Role

AWS CloudHSM does not allow you to edit the `AWSServiceRoleForCloudHSM` role. After the role is created, for example, you cannot change its name because various entities might reference the role by name. Also, you cannot change the role policy. You can, however, use IAM to edit the role description. For more information, see Editing a Service–Linked Role in the IAM User Guide.

Deleting the Service-Linked Role

You cannot delete a service-linked role as long as a cluster to which it has been applied still exists. To delete the role, you must first delete each HSM in your cluster and then delete the cluster. Every cluster in your account must be deleted. You can then use the IAM console, AWS CLI, or API to delete the role. For more information about deleting a cluster, see Deleting an AWS CloudHSM Cluster (p. 48). For more information, see Deleting a Service-Linked Role in the IAM User Guide.

FIPS Validation

Relying on a FIPS-validated HSM can help you meet corporate, contractual, and regulatory compliance requirements for data security in the AWS Cloud. You can review the FIPS-approved security policies for the HSMs provided by AWS CloudHSM below.
FIPS Validation for Hardware Used by CloudHSM

- Certificate #3254 was issued on August 2, 2018
- Certificate #2850 was issued on February 27, 2017

FIPS 140-2 Compliance

The Federal Information Processing Standard (FIPS) Publication 140-2 is a US government security standard that specifies security requirements for cryptographic modules that protect sensitive information. The HSMs provided by AWS CloudHSM comply with FIPS 140-2 level 3.

PCI DSS Compliance

The Payment Card Industry Data Security Standard (PCI DSS) is a proprietary information security standard administered by the PCI Security Standards Council. The HSMs provided by AWS CloudHSM comply with PCI DSS.

Resilience in AWS CloudHSM

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure. For more information about AWS CloudHSM features to support resiliency, see Cluster High Availability and Load Balancing (p. 5).

Infrastructure Security in AWS CloudHSM

As a managed service, AWS CloudHSM is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS CloudHSM through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Network Isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the AWS cloud. You can create a cluster in a private subnet in your VPC. For more information, see Create a Private Subnet (p. 17).

When you create an HSM, AWS CloudHSM put an elastic network interface (ENI) in your subnet so that you can interact with your HSMs. For more information, see Cluster Architecture (p. 3).

AWS CloudHSM creates a security group that allows inbound and outbound communication between HSMs in your cluster. You can use this security group to enable your EC2 instances to communicate with
the HSMs in your cluster. For more information, see Connect Amazon EC2 Instance to AWS CloudHSM Cluster (p. 20).

Authorization of Users

With AWS CloudHSM, operations performed on the HSM require the credentials of an authenticated HSM user. For more information, see HSM Users (p. 10).

Update Management in AWS CloudHSM

AWS manages the firmware. Firmware is maintained by a third party, and must be evaluated by NIST for FIPS 140-2 Level 3 compliance. Only firmware that has been cryptographically signed by the FIPS key, which AWS does not have access to, can be installed.
Troubleshooting AWS CloudHSM

If you encounter problems with AWS CloudHSM, the following topics can help you resolve them.

**Topics**

- Known Issues (p. 340)
- Lost Connection to the Cluster (p. 346)
- Keep HSM Users In Sync Across HSMs In The Cluster (p. 348)
- Verify the Performance of the HSM (p. 348)
- Resolving Cluster Creation Failures (p. 351)
- Missing AWS CloudHSM Audit Logs in CloudWatch (p. 353)
- Retrieving Client Configuration Logs (p. 353)

**Known Issues**

The following issues are currently known for AWS CloudHSM.

**Topics**

- Known Issues for all HSM instances (p. 340)
- Known Issues for Amazon EC2 Instances Running Amazon Linux 2 (p. 342)
- Known Issues for the PKCS #11 SDK (p. 342)
- Known Issues for the JCE SDK (p. 344)
- Known Issues for the OpenSSL SDK (p. 345)

**Known Issues for all HSM instances**

The following issues impact all AWS CloudHSM users regardless of whether they use the key_mgmt_util command line tool, the PKCS #11 SDK, the JCE SDK, or the OpenSSL SDK.

**Issue:** AES key wrapping uses PKCS#5 padding instead of providing a standards-compliant implementation of key wrap with zero padding. Additionally, key wrap with no padding is not supported.

**Impact:** There is no impact if you wrap and unwrap within AWS CloudHSM. Keys wrapped with AWS CloudHSM cannot be unwrapped within other HSMs or software that expects compliance to the no-padding specification. This is because 8 bytes of padding data might be suffixed to your key data following a standards-compliant unwrap. Externally wrapped keys cannot be properly unwrapped into an AWS CloudHSM instance.

**Workaround:** To externally unwrap a key that was wrapped with AES Key Wrapping on a AWS CloudHSM instance, strip the extra padding before you attempt to use the key. You can do this by trimming the extra bytes in a file editor or copying only the key bytes into a new buffer in your code.

**Resolution status:** We are fixing the client and SDKs to provide SP800-38F compliant AES key wrapping. Updates will be announced in the AWS CloudHSM forum and on the version history page.
The update will include mechanisms to assist you in rewrapping any existing wrapped keys in a standards-compliant way.

**Issue:** The client daemon requires at least one valid IP address in its configuration file to successfully connect to the cluster.

- **Impact:** If you delete every HSM in your cluster and then add another HSM, which gets a new IP address, the client daemon continues to search for your HSMs at their original IP addresses.
- **Workaround:** If you run an intermittent workload, we recommend that you use the `IpAddress` argument in the `CreateHsm` function to set the elastic network interface (ENI) to its original value. Note that an ENI is specific to an Availability Zone (AZ). The alternative is to delete the `/opt/cloudhsm/daemon/1/cluster.info` file and then reset the client configuration to the IP address of your new HSM. You can use the `client -a <IP address>` command. For more information, see [Install and Configure the AWS CloudHSM Client (Linux)](p. 34) or [Install and Configure the AWS CloudHSM Client (Windows)](p. 36).

**Issue:** There was an upper limit of 16 KB on data that can be hashed and signed by AWS CloudHSM.

- **Resolution status:** Data less than 16KB in size continues to be sent to the HSM for hashing. We have added capability to hash locally, in software, data between 16KB and 64KB in size. The client and the SDKs will explicitly fail if the data buffer is larger than 64KB. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

**Issue:** Imported keys could not be specified as nonexportable.

- **Resolution Status:** This issue is fixed. No action is required on your part to benefit from the fix.

**Issue:** The default mechanism for the `wrapKey` and `unWrapKey` commands in the `key_mgmt_util` has been removed.

- **Resolution:** When using the `wrapKey` or `unWrapKey` commands, you must use the `-m` option to specify the mechanism. See the examples in the [wrapKey](p. 201) or [unWrapKey](p. 195) articles for more information.

**Issue:** If you have a single HSM in your cluster, HSM failover does not work correctly.

- **Impact:** If the single HSM instance in your cluster loses connectivity, the client will not reconnect with it even if the HSM instance is later restored.
- **Workaround:** We recommend at least two HSM instances in any production cluster. If you use this configuration, you will not be impacted by this issue. For single-HSM clusters, bounce the client daemon to restore connectivity.
- **Resolution status:** This issue has been resolved in the [AWS CloudHSM client 1.1.2](p. 367) release. You must upgrade to this client to benefit from the fix.

**Issue:** If you exceed the key capacity of the HSMs in your cluster within a short period of time, the client enters an unhandled error state.

- **Impact:** When the client encounters the unhandled error state, it freezes and must be restarted.
- **Workaround:** Test your throughput to ensure you are not creating session keys at a rate that the client is unable to handle. You can lower your rate by adding an HSM to the cluster or slowing down the session key creation.
- **Resolution status:** This issue has been resolved in the [AWS CloudHSM client 1.1.2](p. 367) release. You must upgrade to this client to benefit from the fix.
Known Issues for Amazon EC2 Instances Running Amazon Linux 2

**Issue:** Amazon Linux 2 version 2018.07 uses an updated `ncurses` package (version 6) that is currently incompatible with the AWS CloudHSM SDKs. The following error will be returned upon running the AWS CloudHSM `cloudhsm_mgmt_util` (p. 75) or `key_mgmt_util` (p. 122):

```
/opt/cloudhsm/bin/cloudhsm_mgmt_util: error while loading shared libraries: libncurses.so.5: cannot open shared object file: No such file or directory
```

- **Impact:** Instances running on Amazon Linux 2 version 2018.07 will be unable to use all AWS CloudHSM utilities.

- **Workaround:** Issue the following command on your Amazon Linux 2 EC2 instances to install the supported `ncurses` package (version 5):

  ```
sudo yum install ncurses-compat-libs
  ```

- **Resolution status:** This issue has been resolved in the AWS CloudHSM client 1.1.2 (p. 367) release. You must upgrade to this client to benefit from the fix.

Known Issues for the PKCS #11 SDK

**Issue:** The `CKA_DERIVE` attribute was not supported and was not handled.

- **Resolution status:** We have implemented fixes to accept `CKA_DERIVE` if it is set to `FALSE`. `CKA_DERIVE` set to `TRUE` will not be supported until we begin to add key derivation function support to AWS CloudHSM. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

**Issue:** The `CKA_SENSITIVE` attribute was not supported and was not handled.

- **Resolution status:** We have implemented fixes to accept and properly honor the `CKA_SENSITIVE` attribute. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

**Issue:** Multipart hashing and signing are not supported.

- **Impact:** `C_DigestUpdate` and `C_DigestFinal` are not implemented. `C_SignFinal` is also not implemented and will fail with `CKR_ARGUMENTS_BAD` for a non-NULL buffer.

- **Workaround:** Hash your data within your application and use AWS CloudHSM only for signing the hash.

- **Resolution status:** We are fixing the client and the SDKs to correctly implement multipart hashing. Updates will be announced in the AWS CloudHSM forum and on the version history page.
**Issue:** `C_GenerateKeyPair` does not handle `CKA_MODULUS_BITS` or `CKA_PUBLIC_EXPONENT` in the private template in a manner that is compliant with standards.

- **Impact:** `C_GenerateKeyPair` should return `CKA_TEMPLATE_INCONSISTENT` when the private template contains `CKA_MODULUS_BITS` or `CKA_PUBLIC_EXPONENT`. It instead generates a private key for which all usage fields are set to `FALSE`. The key cannot be used.
- **Workaround:** We recommend that your application check the usage field values in addition to the error code.
- **Resolution status:** We are implementing fixes to return the proper error message when an incorrect private key template is used. The updated PKCS #11 library will be announced on the version history page.

**Issue:** You could not hash more than 16KB of data. For larger buffers, only the first 16KB will be hashed and returned. The excess data would have been silently ignored.

- **Resolution status:** Data less than 16KB in size continues to be sent to the HSM for hashing. We have added capability to hash locally, in software, data between 16KB and 64KB in size. The client and the SDKs will explicitly fail if the data buffer is larger than 64KB. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

**Issue:** Buffers for the `C_Encrypt` and `C_Decrypt` API operations cannot exceed 16 KB when using the `CKM_AES_GCM` mechanism. Also, AWS CloudHSM does not support multipart AES-GCM encryption.

- **Impact:** You cannot use the `CKM_AES_GCM` mechanism to encrypt data larger than 16 KB.
- **Workaround:** You can use an alternative mechanism such as `CKM_AES_CBC` or you can divide your data into pieces and encrypt each piece individually. You must manage the division of your data and subsequent encryption. AWS CloudHSM does not perform multipart AES-GCM encryption for you. Note that FIPS requires that the initialization vector (IV) for AES-GCM be generated on the HSM. Therefore, the IV for each piece of your AES-GCM encrypted data will be different.
- **Resolution status:** We are fixing the SDK to fail explicitly if the data buffer is too large. We return `CKR_MECHANISM_INVALID` for the `C_EncryptUpdate` and `C_DecryptUpdate` API operations. We are evaluating alternatives to support larger buffers without relying on multipart encryption. Updates will be announced in the AWS CloudHSM forum and on the version history page.

**Issue:** Elliptic-curve Diffie-Hellman (ECDH) key derivation is executed partially within the HSM. Your EC private key remains within the HSM at all times, but the key derivation process is performed in multiple steps. As a result, intermediate results from each step are available on the client.

- **Impact:** The key derived using the `CKM_ECDH1_DERIVE` mechanism is first available on the client and is then imported into the HSM. A key handle is then returned to your application.
- **Workaround:** If you are implementing SSL/TLS Offload in AWS CloudHSM, this limitation may not be an issue. If your application requires your key to remain within an FIPS boundary at all times, consider using an alternative protocol that does not rely on ECDH key derivation.
- **Resolution status:** We are developing the option to perform ECDH key derivation entirely within the HSM. The updated implementation will be announced on the version history page once available.

**Issue:** Verification of secp256k1 signatures fails on EL6 platforms such as CentOS6 and RHEL6. This is because the CloudHSM PKCS#11 library avoids a network call during initialization of the verification operation by using OpenSSL to verify EC curve data. Since Secp256k1 is not supported by the default OpenSSL package on EL6 platforms, the initialization fails.

- **Impact:** Secp256k1 signature verification will fail on EL6 platforms. The verify call will fail with a `CKR_HOST_MEMORY` error.
Known Issues for the JCE SDK

**Issue:** When working with asymmetric key pairs, you see occupied key capacity even when you are not explicitly creating or importing keys

**Impact:** This issue can cause your HSMs to unexpectedly run out of key space and occurs when your application uses a standard JCE key object for crypto operations instead of a CaviumKey object. When you use a standard JCE key object, CaviumProvider implicitly imports that key into the HSM as a session key does not delete this key until the application exits. As a result, keys build up while the application is running and can cause your HSMs to run out of free key space, thus freezing your application.

**Workaround:** When using the CaviumSignature class, CaviumCipher class, CaviumMac class, or the CaviumKeyAgreement class, you should supply the key as a CaviumKey instead of a standard JCE key object.

You can manually convert a normal key to a CaviumKey using the ImportKey class, and can then manually delete the key after the operation is complete.

**Resolution status:** We are updating the CaviumProvider to properly manage implicit imports. The fix will be announced on the version history page once available.

**Issue:** You cannot specify attributes when unwrapping keys.

**Impact:** All keys are unwrapped as exportable session keys.

**Workaround:** You can script key_mgmt_util to unwrap keys with limited attribute customization, or use the PKCS #11 library to unwrap keys with full template support.

**Resolution status:** We are planning to add full key parameter specification for the JCE SDK’s unwrap command in a future release. The update will be announced on the version history page once available.

**Issue:** The JCE KeyStore is read only.

**Impact:** You cannot store an object type that is not supported by the HSM in the JCE keystore today. Specifically, you cannot store certificates in the keystore. This precludes interoperability with tools like jarsigner, which expect to find the certificate in the keystore.

**Workaround:** You can rework your code to load certificates from local files or from an S3 bucket location instead of from the keystore.

**Resolution status:** We are adding support for certificate storage in the keystore. The feature will be announced on the version history page once available.

**Issue:** Buffers for AES-GCM encryption cannot exceed 16,000 bytes. Also, multi-part AES-GCM encryption is not supported.

**Impact:** You cannot use AES-GCM to encrypt data larger than 16,000 bytes.

**Workaround:** You can use an alternative mechanism, such as AES-CBC, or you can divide your data into pieces and encrypt each piece individually. If you divide the data, you must manage the divided...
ciphertext and its decryption. Because FIPS requires that the initialization vector (IV) for AES-GCM be generated on the HSM, the IV for each AES-GCM-encrypted piece of data will be different.

- **Resolution status:** We are fixing the SDK to fail explicitly if the data buffer is too large. We are evaluating alternatives that support larger buffers without relying on multi-part encryption. Updates will be announced in the AWS CloudHSM forum and on the version history page.

**Issue:** Elliptic-curve Diffie-Hellman (ECDH) key derivation is executed partially within the HSM. Your EC private key remains within the HSM at all times, but the key derivation process is performed in multiple steps. As a result, intermediate results from each step are available on the client. An ECDH key derivation sample is available in the Java code samples (p. 247).

- **Impact:** Software version 3.0 adds ECDH functionality to the JCE. When you use the CKM_ECDH1_DERIVE mechanism to derive the key, it is first available on the client and is then imported into the HSM. A key handle is then returned to your application.
- **Workaround:** If you are implementing SSL/TLS Offload in AWS CloudHSM, this limitation may not be an issue. If your application requires your key to remain within an FIPS boundary at all times, consider using an alternative protocol that does not rely on ECDH key derivation.
- **Resolution status:** We are developing the option to perform ECDH key derivation entirely within the HSM. When available, we'll announce the updated implementation on the version history page.

**Known Issues for the OpenSSL SDK**

**Issue:** Only RSA offload to the HSM is supported by default.

- **Impact:** To maximize performance, the SDK is not configured to offload additional functions such as random number generation or EC-DH operations.
- **Workaround:** Please contact us through a support case if you need to offload additional operations.
- **Resolution status:** We are adding support to the SDK to configure offload options through a configuration file. The update will be announced on the version history page once available.

**Issue:** RSA encryption and decryption with OAEP padding using a key on the HSM is not supported.

- **Impact:** Any call to RSA encryption and decryption with OAEP padding fails with a divide-by-zero error. This occurs because the OpenSSL dynamic engine calls the operation locally using the fake PEM file instead of offloading the operation to the HSM.
- **Workaround:** You can perform this procedure by using either the AWS CloudHSM Software Library for PKCS #11 (p. 214) or the AWS CloudHSM Software Library for Java (p. 239).
- **Resolution status:** We are adding support to the SDK to correctly offload this operation. The update will be announced on the version history page once available.

**Issue:** Only private key generation of RSA and ECC keys is offloaded to the HSM. For any other key type, the OpenSSL AWS CloudHSM engine is not used for call processing. The local OpenSSL engine is used instead. This generates a key locally in software.

- **Impact:** Because the failover is silent, there is no indication that you have not received a key that was securely generated on the HSM. You will see an output trace that contains the string "......+++++" if the key is locally generated by OpenSSL in software. This trace is absent when the operation is offloaded to the HSM. Because the key is not generated or stored on the HSM, it will be unavailable for future use.
- **Workaround:** Only use the OpenSSL engine for key types it supports. For all other key types, use PKCS #11 or JCE in applications, or use key_mgmt_util in the AWS CLI.
Lost Connection to the Cluster

When you configured the AWS CloudHSM client (p. 36), you provided the IP address of the first HSM in your cluster. This IP address is saved in the configuration file for the AWS CloudHSM client. When the client starts, it tries to connect to this IP address. If it can’t—for example, because the HSM failed or you deleted it—you might see errors like the following:

| LIQUIDSECURITY: Daemon socket connection error |
| LIQUIDSECURITY: Invalid Operation |

To resolve these errors, update the configuration file with the IP address of an active, reachable HSM in the cluster.

To update the configuration file for the AWS CloudHSM client

1. Use one of the following ways to find the IP address of an active HSM in your cluster.
   - View the HSMs tab on the cluster details page in the AWS CloudHSM console.
   - Use the AWS Command Line Interface (AWS CLI) to issue the `describe-clusters` command.

   You need this IP address in a subsequent step.

2. Use the following command to stop the client.

   Amazon Linux
   ```
   $ sudo stop cloudhsm-client
   ```

   Amazon Linux 2
   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 6
   ```
   $ sudo stop cloudhsm-client
   ```

   CentOS 7
   ```
   $ sudo service cloudhsm-client stop
   ```

   RHEL 6
   ```
   $ sudo stop cloudhsm-client
   ```

   RHEL 7
   ```
   $ sudo service cloudhsm-client stop
   ```

   Ubuntu 16.04 LTS
   ```
   $ sudo service cloudhsm-client stop
   ```
Windows

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM> net.exe stop AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  Use Ctrl+C in the command window where you started the AWS CloudHSM client.

3. Use the following command to update the client's configuration file, providing the IP address that you found in a previous step.

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

4. Use the following command to start the client.

   Amazon Linux

   ```
   $ sudo start cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 6

   ```
   $ sudo start cloudhsm-client
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client start
   ```

   RHEL 6

   ```
   $ sudo start cloudhsm-client
   ```

   RHEL 7

   ```
   $ sudo service cloudhsm-client start
   ```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```
Keep HSM Users In Sync Across HSMs In The Cluster

To manage your HSM's users (p. 52), you use a AWS CloudHSM command line tool known as cloudhsm_mgmt_util. It communicates only with the HSMs that are in the tool's configuration file. It's not aware of other HSMs in the cluster that are not in the configuration file.

AWS CloudHSM synchronizes the keys on your HSMs across all other HSMs in the cluster, but it doesn't synchronize the HSM's users or policies. When you use cloudhsm_mgmt_util to manage HSM users (p. 52), these user changes might affect only some of the cluster's HSMs—the ones that are in the cloudhsm_mgmt_util configuration file. This can cause problems when AWS CloudHSM syncs keys across HSMs in the cluster, because the users that own the keys might not exist on all HSMs in the cluster.

To avoid these problems, edit the cloudhsm_mgmt_util configuration file before managing users. For more information, see Step 4: Update the cloudhsm_mgmt_util Configuration File (p. 79).

Verify the Performance of the HSM

To verify the performance of the HSMs in your AWS CloudHSM cluster, you can use the pkpspeed (Linux) or pkpspeed_blocking (Windows) tool that is included with the AWS CloudHSM client software. For more information about installing the client on a Linux EC2 instance, see Install and Configure the AWS CloudHSM Client (Linux) (p. 34). For more information about installing the client on a Windows instance, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

After you install and configure the AWS CloudHSM client, run the following command to start it.

Amazon Linux

```bash
# sudo start cloudhsm-client
```

Amazon Linux 2

```bash
# sudo service cloudhsm-client start
```

CentOS 6

```bash
# sudo start cloudhsm-client
```

CentOS 7

```bash
# sudo service cloudhsm-client start
```

RHEL 6

```bash
# sudo start cloudhsm-client
```
### RHEL 7

```bash
# sudo service cloudhsm-client start
```

### Ubuntu 16.04 LTS

```bash
# sudo service cloudhsm-client start
```

### Windows

- For Windows client 1.1.2+:
  ```bash
  C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:
  ```bash
  C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
  ```

If you have already installed the client software, you might need to download and install the latest version to get pkpspeed. You can find the pkpspeed tool at `/opt/cloudhsm/bin/pkpspeed` in Linux or `C:\Program Files\Amazon\CloudHSM` in Windows.

To use pkpspeed, run the `pkpspeed` command or `pkpspeed_blocking.exe`, specifying the user name and password of a crypto user (CU) on the HSM. Then set the options to use while considering the following recommendations.

**Recommendations**

- To test the performance of RSA sign and verify operations, choose the `RSA_CRT` cipher in Linux or option B in Windows. Don't choose RSA (option A in Windows). The ciphers are equivalent, but `RSA_CRT` is optimized for performance.
- Start with a small number of threads. For testing AES performance, one thread is typically enough to show maximum performance. For testing RSA performance (`RSA_CRT`), three or four threads is typically enough.

The following examples show the options that you can choose with pkpspeed (Linux) or pkpspeed_blocking (Windows) to test the HSM’s performance for RSA and AES operations.

**Example – Using pkpspeed to test RSA performance**

You can run this example on Windows, Linux, and compatible operating systems.

**Linux**

Use these instructions for Linux and compatible operating systems.

```bash
/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

```plaintext
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

<table>
<thead>
<tr>
<th>OPERATIONS/second</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>821</td>
<td>1</td>
</tr>
<tr>
<td>833</td>
<td>1</td>
</tr>
<tr>
<td>845</td>
<td>1</td>
</tr>
<tr>
<td>835</td>
<td>1</td>
</tr>
<tr>
<td>837</td>
<td>1</td>
</tr>
<tr>
<td>836</td>
<td>1</td>
</tr>
<tr>
<td>837</td>
<td>1</td>
</tr>
<tr>
<td>849</td>
<td>1</td>
</tr>
<tr>
<td>841</td>
<td>1</td>
</tr>
<tr>
<td>856</td>
<td>1</td>
</tr>
<tr>
<td>841</td>
<td>1</td>
</tr>
<tr>
<td>847</td>
<td>1</td>
</tr>
<tr>
<td>838</td>
<td>1</td>
</tr>
<tr>
<td>843</td>
<td>1</td>
</tr>
<tr>
<td>852</td>
<td>1</td>
</tr>
<tr>
<td>837</td>
<td></td>
</tr>
</tbody>
</table>

Example – Using pkpspeed to test AES performance

Linux

Use these instructions for Linux and compatible operating systems.

```plaintext
/opt/cloudhsm/bin/pkpspeed -s <CU user name> -p <password>
```

SDK Version: 2.03
Resolving Cluster Creation Failures

When you create a cluster, AWS CloudHSM creates the AWSServiceRoleForCloudHSM service-linked role, if the role does not already exist. If AWS CloudHSM cannot create the service-linked role, your attempt to create a cluster might fail.

This topic explains how to resolve the most common problems so you can create a cluster successfully. You need to create this role only one time. Once the service-linked role is created in your account, you can use any of the supported methods to create additional clusters and to manage them.
The following sections offer suggestions to troubleshoot cluster creation failures that are related to the service-linked role. If you try them but are still unable to create a cluster, contact AWS Support. For more information about the AWSServiceRoleForCloudHSM service-linked role, see Service-Linked Roles for AWS CloudHSM (p. 336).

Topics
- Add the Missing Permission (p. 352)
- Create the Service-Linked Role Manually (p. 352)
- Use a Nonfederated User (p. 352)

Add the Missing Permission

To create a service-linked role, the user must have the iam:CreateServiceLinkedRole permission. If the IAM user who is creating the cluster does not have this permission, the cluster creation process fails when it tries to create the service-linked role in your AWS account.

When a missing permission causes the failure, the error message includes the following text.

This operation requires that the caller have permission to call iam:CreateServiceLinkedRole to create the CloudHSM Service Linked Role.

To resolve this error, give the IAM user who is creating the cluster the AdministratorAccess permission or add the iam:CreateServiceLinkedRole permission to the user's IAM policy. For instructions, see Adding Permissions to a New or Existing User.

Then try to create the cluster (p. 17) again.

Create the Service-Linked Role Manually

You can use the IAM console, CLI, or API to create the AWSServiceRoleForCloudHSM service-linked role. For more information, see Creating a Service-Linked Role in the IAM User Guide.

Use a Nonfederated User

Federated users, whose credentials originate outside of AWS, can perform many of the tasks of a nonfederated user. However, AWS does not allow users to make the API calls to create a service-linked role from a federated endpoint.

To resolve this problem, create a non-federated user (p. 15) with the iam:CreateServiceLinkedRole permission, or give an existing non-federated user the iam:CreateServiceLinkedRole permission. Then have that user create a cluster (p. 17) from the AWS CLI. This creates the service-linked role in your account.

Once the service-linked role is created, if you prefer, you can delete the cluster that the nonfederated user created. Deleting the cluster does not affect the role. Thereafter, any user with the required permissions, included federated users, can create AWS CloudHSM clusters in your account.

To verify that the role was created, open the IAM console at https://console.aws.amazon.com/iam/ and choose Roles. Or use the IAM get-role command in the AWS CLI.

```
$ aws iam get-role --role-name AWSServiceRoleForCloudHSM
{
    "Role": {
        "Description": "Role for CloudHSM service operations",
        "AssumeRolePolicyDocument": {
            "Statement": ["{",
                "Effect": "Allow",
                "Principal": {"AWS": ["*"]},
                "Action": ["sts:AssumeRole"],
                "Resource": "*"
            ]
        }
    }
}
```
Missing AWS CloudHSM Audit Logs in CloudWatch

If you created a cluster before January 20th, 2018, you will need to manually configure a service-linked role (p. 336) in order to enable the delivery of that cluster’s audit logs. For instructions on how to enable a service-linked role on an HSM cluster, see Understanding Service-Linked Roles (p. 336), as well as Creating a Service-Linked Role in the IAM User Guide.

Retrieving Client Configuration Logs

The client log script gathers important environmental information about your operating system, AWS CloudHSM cluster, active or inactive HSMs, and local configuration files. When you open a case to request assistance with your application, this information helps AWS understand your setup and issue, and helps us troubleshoot your issue faster.

The script extracts the following information:

• Operating system and its current version
• Client configuration information from cloudhsm_client.cfg, cloudhsm_mgmt_util.cfg, and application.cfg files
• Client logs from the location specific to the platform
• Cluster and HSM information by using cloudhsm_mgmt_util
• OpenSSL information
• Current client and build version
• Installer version

Running the Client Log Script

The script creates an output file with all the gathered information. You can specify the directory path, where you want to add the output file, as an output parameter in the command. The directory path must have the appropriate write access. Alternatively, you can run the script without specifying the directory path. In such case, the script creates the output file inside the temp directory.

Linux: /opt/cloudhsm/bin/client_info -output
**Windows:** C:\Program Files\Amazon\CloudHSM\client_info -output

Replace the `output` parameter with the directory path where you want to create the output file.
AWS CloudHSM Client and Software Information

To manage and use the HSMs in your cluster, you use the AWS CloudHSM client and related software libraries. If you installed the AWS CloudHSM client for Linux or Windows and any required software libraries, you have all the software needed to use AWS CloudHSM.

This section provides information about supported platforms and a full version history.

Topics
- AWS CloudHSM Client and Software Version History
- Supported Platforms

AWS CloudHSM Client and Software Version History

This page provides release notes for each version of the AWS CloudHSM client and related software libraries. We include download links for the most recent versions. We recommend that you use the most recent version whenever possible.

To check the client version

- On a Linux system, use the following command:

```
rpm -qa | grep ^cloudhsm
```

- On a Windows system, use the following command:

```
wmic product get name,version
```

Release Notes

- Version 3.0
- Version 2.0.4
- Version 2.0.3
- Version 2.0.1
- Version 2.0.0
- Version 1.1.2
- Version 1.1.1
- Version 1.1.0
- Version 1.0.18
- Version 1.0.14
- Version 1.0.11
- Version 1.0.10
Version 3.0

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

Amazon Linux

Download the version 3.0 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Amazon Linux 2

Download the version 3.0 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 6

Download the version 3.0 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 7

Download the version 3.0 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

RHEL 6

Download the version 3.0 software for RedHat Enterprise Linux 6:

- AWS CloudHSM Client
- PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 3.0 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 3.0 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.0 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36). Download the latest version (3.0) software for Windows Server:

• AWS CloudHSM for Windows Server

Version 3.0 provides important improvements to operational stability and performance. This is a recommended update due to various bug fixes.

**AWS CloudHSM Client Software**

• Key_mgmt_util includes the following updates:
  • Removed the default mechanism from wrapKey and unwrapKey. You must explicitly provide a mechanism when using these function.
  • Added support for key wrap and unwrap using AES-GCM. To use this wrapping mechanism, specify -m 10 with wrapKey and unwrapKey. For more information, see wrapKey (p. 201) or unwrapKey (p. 195).
  • Changed the name for AES key wrapping using mechanism 4 from CLOUDHSM_AES_KEY_WRAP to AES_KEY_WRAP_PAD_PKCS5, to reflect that AWS CloudHSM utilizes PKCS5 padding while wrapping keys. For more information, see the list of Known Issues (p. 340).
  • Improved findKey to return keys owned and shared by the CU that is logged in. For more information, see the findKey (p. 141) article.

Cloudhsm_mgmt_util includes the following updates:

• COs can set the OBJ_ATTR_TRUSTED attribute on any key (value 134) in the HSM by using setAttribute to mark a key as trusted.

  **Note**
  OBJ_ATTR_TRUSTED is the only attribute that can be set by a CO. For more information, see the setAttribute (p. 190) command.
- findAllKeys displays keys owned by a CU and shared with that CU. Learn more at findAllKeys (p. 93).

**PKCS #11 Library**

- PKCS #11 no longer requires Redis for high performance. Redis is no longer included in the installation packages. If you used Redis in previous installations, update your start-up and installation scripts to remove Redis commands.
- Added support for encryption and decryption using D3DES ECB and AES_CTR. The full list of supported functions and mechanisms in PKCS#11 is available in the Supported PKCS #11 Mechanisms (p. 220) article.

  Code samples for des_ecb.c and aes_ctr.c are available on GitHub.

- Added support for key derivation using HMAC KDF (SP 800-108) which enables you to use the CKM_SP800_108_COUNTER_KDF mechanism with the C_DeriveKey function. For additional information see the code sample hmac-kdf.c.

- Added support for key wrap and unwrap using AES-GCM, through the CKM_CLOUDHSM_AES_GCM mechanism. For more information, see the aes_gcm_wrapping sample on GitHub.

- Added support for the following attributes: CKA_NEVER_EXTRACTABLE, CKA_DERIVE, CKA_ALWAYS_SENSITIVE, CKA_WRAP_WITH_TRUSTED, CKA_TRUSTED, CKA_WRAP_TEMPLATE, CKA_UNWRAP_TEMPLATE, CKA_DESTROYABLE.

  The full list of supported attributes is in the Supported PKCS #11 Attributes (p. 225) article. To learn about using trusted keys for controlled wrapping and unwrapping, see the article on using trusted keys. To see the available samples that work with the newly supported attributes, go to the AWS CloudHSM examples on GitHub.

- Added the mechanism, CKM_CLOUDSHM_AES_GCM, which is a memory-safe AES-GCM implementation. This proprietary mechanism is a safer alternative to the standard CKM_AES_GCM. CKM_CLOUDSHM_AES_GCM prepends the IV generated by the HSM to the ciphertext instead of writing it back into the CK_GCM_PARAMS structure provided during cipher initialization. You can use CKM_CLOUDSHM_AES_GCM with the C_Encrypt or C_WrapKey functions. When using this mechanism, the pIV variable in the CK_GCM_PARAMS struct must be set to NULL. See the CKM_CLOUDHSM_AES_GCM entry in the table in the Supported PCKS#11-Mechanisms (p. 220) article.

**OpenSSL Dynamic Engine**

- Updated the version for consistency.

**Java Library**

- Added the CloudHSM keystore which supports certificates, allowing interoperability with third party tools like keytool and jarsigner. To learn how to use this keystore, see the CloudHSM Key Store (p. 248) article, or the CloudHSMKeyStoreExampleRunner sample on GitHub. To use the keystore with KeyTool and Jarsigner, refer to the CloudHSM KeyStore Integration guide.

- Added support for encryption and decryption using D3DES ECB and AES_CTR. The full list of supported functions in JDE is available in the Supported Mechanisms (p. 244) article. You can find sample code for the DESedeECMEncryptDecryptRunner sample and AESCTREncryptDecryptRunner sample on GitHub.

- Added support for key wrap and unwrap using AES-GCM. See the AESGCMWrappingRunner sample on GitHub.

- Added support for ECDH. The derivation is partially executed on the HSM. For additional information, see the list of Known Issues for the JCE SDK (p. 340) before using ECDH with CloudHSM. To learn how to use ECDH with the JCE, see the ECDHOperationsRunner sample on GitHub.
• Trusted key wrapping and key derivation are not yet available through the JCE.

**Windows (CNG, KSP)**

• Updated the version for consistency.
• Added `import_key.exe` to associate pre-existing CloudHSM keys with corresponding certificates.

**Version: 2.0.4**

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

**Amazon Linux**

Download the version 2.0.4 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**Amazon Linux 2**

Download the version 2.0.4 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**CentOS 6**

Download the version 2.0.4 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**CentOS 7**

Download the version 2.0.4 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**RHEL 6**

Download the version 2.0.4 software for RedHat Enterprise Linux 6:
• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 2.0.4 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 2.0.4 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 2.0.4 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the latest version (2.0.4) software for Windows Server:

• AWS CloudHSM Client for Windows Server

Version 2.0.4 provides important improvements to operational stability and performance. This is a recommended update due to various bug fixes.

AWS CloudHSM Client Software

• Improved stability and bug fixes.

PKCS #11 Library

• Updated the version for consistency.

OpenSSL Dynamic Engine

• Updated the version for consistency.

Java Library

• Improved stability and bug fixes.
Windows (CNG, KSP)

- Improved stability and bug fixes.
- You can now securely store and access HSM credentials through Windows Credential Manager (p. 263).

Version: 2.0.3

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

Amazon Linux

Download the version 2.0.3 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Amazon Linux 2

Download the version 2.0.3 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 6

Download the version 2.0.3 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 7

Download the version 2.0.3 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

RHEL 6

Download the version 2.0.3 software for RedHat Enterprise Linux 6:

- AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 2.0.3 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 2.0.3 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 2.0.3 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the latest version (2.0.3) software for Windows Server:

• AWS CloudHSM Client for Windows Server

Version 2.0.3 provides important improvements to operational stability and performance. This is a recommended update due to various bug fixes.

AWS CloudHSM Client Software

• Improved stability and bug fixes.

PKCS #11 Library

• Improved stability and bug fixes.

OpenSSL Dynamic Engine

• Updated the version for consistency.

Java Library

• Improved stability and bug fixes.
Windows (CNG, KSP)

- Improved stability and bug fixes.

Version 2.0.1

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

Amazon Linux

Download the version 2.0.1 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Amazon Linux 2

Download the version 2.0.1 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 6

Download the version 2.0.1 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 7

Download the version 2.0.1 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

RHEL 6

Download the version 2.0.1 software for RedHat Enterprise Linux 6:

- AWS CloudHSM Client
- PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 2.0.1 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 2.0.1 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 2.0.1 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the latest version (2.0.1) software for Windows Server:

• AWS CloudHSM Client for Windows Server

Version 2.0.1 is a strongly recommended upgrade, as it provides various security improvements and bug fixes. Significant changes in this version are as follows:

AWS CloudHSM Client Software

• Security improvements and bug fixes.

PKCS #11 Library

• Security improvements and bug fixes.

OpenSSL Dynamic Engine

• Security improvements and bug fixes.

Java Library

• Security improvements and bug fixes.

Windows (CNG, KSP)
• Security improvements and bug fixes.

Version 2.0.0

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

Amazon Linux

Download the version 2.0.0 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Amazon Linux 2

Download the version 2.0.0 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 6

Download the version 2.0.0 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

CentOS 7

Download the version 2.0.0 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

RHEL 6

Download the version 2.0.0 software for RedHat Enterprise Linux 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
RHEL 7

Download the version 2.0.0 software for RedHat Enterprise Linux 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Ubuntu 16.04 LTS

Download the version 2.0.0 software for Ubuntu 16.04 LTS:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 2.0.0 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the latest version (2.0.0) software for Windows Server:

- AWS CloudHSM Client for Windows Server

Version 2.0.0 provides important improvements to operational stability and performance. It also enables secure key exchange between HSMs. Significant changes in this version are as follows:

**AWS CloudHSM Client Software**

- Performance improvements and bug fixes

**PKCS #11 Library**

- Added RSA OAEP and RSA AES key wrapping mechanisms.
- Added AES-ECB encryption support.
- Added secp256k1 curve support.

For more information about updated key wrapping mechanisms, see AWS CloudHSM Software Library for PKCS #11 (p. 214).

**OpenSSL Dynamic Engine**

- Updated the version for consistency.

**Java Library**
• Improved performance for AES-GCM encrypt and decrypt.
• Added RSA OAEP and RSA AES key wrapping mechanisms. Note that you cannot specify key attributes when unwrapping with the Java library. For more information, see Known Issues for the JCE SDK (p. 344).
• Added AES-ECB encryption support.
• Added secp256k1 curve support.

For more information about updated key wrapping mechanisms, see AWS CloudHSM Software Library for Java (p. 239).

Windows (CNG, KSP)
• Updated the version for consistency.

**Version: 1.1.2**

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

Amazon Linux
  Download the version 1.1.2 software for Amazon Linux:
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
  • Java Library

Amazon Linux 2
  Download the version 1.1.2 software for Amazon Linux 2:
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
  • Java Library

CentOS 6
  Download the version 1.1.2 software for CentOS 6:
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
  • Java Library

CentOS 7
  Download the version 1.1.2 software for CentOS 7:
  • AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 6

Download the version 1.1.2 software for RedHat Enterprise Linux 6:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 1.1.2 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 1.1.2 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 1.1.2 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the latest version (1.1.2) software for Windows Server:

• AWS CloudHSM Client for Windows Server

Version 1.1.2 is a strongly recommended upgrade, as it contains a change that runs the AWS CloudHSM client software for Windows as a service, as well as performance improvements and bug fixes. Significant changes in this version are as follows:

AWS CloudHSM Client Software

• The AWS CloudHSM client software for Windows now runs as a Windows service.

PKCS #11 Library
• DER-formatted EC public keys are now correctly imported.

  **Note**
  At this time, AWS CloudHSM continues to support the ability to import EC keys in raw format. Support for this format may be deprecated at a future time, as it is not compliant with PKCS#11 specifications.

• Improved performance and bug fixes.

**OpenSSL Dynamic Engine**

• Updated the version for consistency.

**Java Library**

• Updated the version for consistency.

**Windows (CNG, KSP)**

• The AWS CloudHSM client now runs as a service. For information on updated usage instructions, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36)

**Version: 1.1.1**

To download the software, choose the tab for your preferred operating system, then choose the link to each software package.

**Amazon Linux**

  Download the version 1.1.1 software for Amazon Linux:
  
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
  • Java Library

**Amazon Linux 2**

  Download the version 1.1.1 software for Amazon Linux 2:
  
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
  • Java Library

**CentOS 6**

  Download the version 1.1.1 software for CentOS 6:
  
  • AWS CloudHSM Client
  • PKCS #11 Library
  • OpenSSL Dynamic Engine
• Java Library

CentOS 7

Download the version 1.1.1 software for CentOS 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 6

Download the version 1.1.1 software for RedHat Enterprise Linux 6:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

RHEL 7

Download the version 1.1.1 software for RedHat Enterprise Linux 7:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Ubuntu 16.04 LTS

Download the version 1.1.1 software for Ubuntu 16.04 LTS:

• AWS CloudHSM Client
• PKCS #11 Library
• OpenSSL Dynamic Engine
• Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 1.1.1 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the version (1.1.1) software for Windows Server:

• AWS CloudHSM Client for Windows Server

Significant changes in this version include:

AWS CloudHSM Client Software
- Improved stability and bug fixes.
- In cloud_hsm_mgmt_util, enable_e2e now set by default.
- **SECURITY FIX:** in key_mgmt_util, resolved issue with the incorrect PKCS#1v1.5 signature parsing. This eliminates potential errors when validating signatures with imported RSA keys that use a public exponent of 3. CloudHSM does not allow generating RSA keys with exponents smaller than 65537 to meet FIPS 140-2 requirements.

**PKCS #11 Library**

- Improved stability and bug fixes.
- **SECURITY FIX:** Resolved issue with incorrect PKCS#1v1.5 signature parsing. This eliminates potential errors when validating signatures with imported RSA keys that use a public exponent of 3. CloudHSM does not allow generating RSA keys with exponents smaller than 65537 to meet FIPS 140-2 requirements.
- **BREAKING CHANGE:** To protect against user error, AES-GCM initialization now requires the user supplied IV buffer to be zeroized. NIST requires the IV for AES-GCM to be generated by the HSM and noted by the application after encryption is complete, as described here (p. 220). IV is always 12 bytes long.
- Added support for CKM_RSA_PKCS_KEY_PAIR_GEN mechanism.
- Added software hashing of buffers larger than 16KB for digest, sign and verify operations. Hashes of buffers less than 16KB continue to be offloaded to the HSM as before.
- **BREAKING CHANGE:** Strengthened PKCS#11 compliance, including explicit failure when handling unsupported or inconsistent attributes. If your application was not strictly PKCS#11 compliant before, you may experience errors or failures after updating to this version. Specifically:
  - If an application is already logged in, logging in will now return the error CKR_USER_ALREADY_LOGGED_IN.
  - CKA_KEY_GEN_MECHANISM will cause an error if included in a C_CreateObject call.
  - CKA_ALWAYSSENSITIVE, CKA_LOCAL and CKA_NEVEREXTRACTABLE will cause errors if included in a key generation or import template.
  - CKA_VALUE_LEN is now validated.
  - By default, new keys are scoped as session keys rather than token keys, to comply with PKCS#11.

**OpenSSL Dynamic Engine**

- Improved stability and bug fixes.
- **SECURITY FIX:** Resolved issue with incorrect PKCS#1v1.5 signature parsing. This eliminates potential errors when validating signatures with imported RSA keys that use a public exponent of 3. CloudHSM does not allow generating RSA keys with exponents smaller than 65537 to meet FIPS 140-2 requirements.

**Java Library**

- Improved stability and bug fixes.
- Added software hashing of buffers larger than 16KB for digest, sign and verify operations. Hashes of buffers less than 16KB continue to be offloaded to the HSM as before.
- For non-exportable keys, getFormat and getEncoded now return NULL without throwing an exception.

**Windows (CNG, KSP)**

- **SECURITY FIX:** Resolved issue with incorrect PKCS#1v1.5 signature parsing. This eliminates potential errors when validating signatures with imported RSA keys that use a public exponent of 3. CloudHSM
does not allow generating RSA keys with exponents smaller than 65537 to meet FIPS 140-2 requirements.

**Version: 1.1.0**

To download the software, choose the tab for your preferred operating system, and then choose the link to each software package.

**Amazon Linux**

Download the version 1.1.0 software for Amazon Linux:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**Amazon Linux 2**

Download the version 1.1.0 software for Amazon Linux 2:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**CentOS 6**

Download the version 1.1.0 software for CentOS 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**CentOS 7**

Download the version 1.1.0 software for CentOS 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

**RHEL 6**

Download the version 1.1.0 software for RedHat Enterprise Linux 6:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine (Supports RHEL 6.5 and later)
- Java Library
RHEL 7

Download the version 1.1.0 software for RedHat Enterprise Linux 7:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Ubuntu 16.04 LTS

Download the version 1.1.0 software for Ubuntu 16.04 LTS:

- AWS CloudHSM Client
- PKCS #11 Library
- OpenSSL Dynamic Engine
- Java Library

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 1.1.0 client software for Windows Server includes the required CNG and KSP providers. For details, see Install and Configure the AWS CloudHSM Client (Windows) (p. 36).

Download the version 1.1.0 software for Windows Server:

- AWS CloudHSM Client for Windows Server

Significant changes in this version include the following:

**AWS CloudHSM Client Software**

- Added new Linux platforms.
  - Amazon Linux 2
  - Ubuntu 16.04 LTS
  - RedHat Enterprise Linux 6 (RHEL 6)
  - RedHat Enterprise Linux 7 (RHEL 7)
  - CentOS 6
  - CentOS 7

**CNG/KSP Providers for Windows Server**

The AWS CloudHSM client software for Windows Server includes the required CNG and KSP providers.

- Updated the version for consistency.

**PKCS #11 Library**

- Added support for Linux platforms.

**OpenSSL Dynamic Engine**
• Added support for Linux platforms.

Java Library

• If you downloaded this package prior to May 23, 5PM PDT, you will need to recompile your application for it to work with this version of the JCE, as the `loadNative()` method had temporarily changed from non-static to static. Alternatively, you can download the package again, and install the JCE. We have now restored the `loadNative()` method to static.
• Eliminated the breaking change in version 1.0.18. The `LoginManager.getInstance()` public method accepts `username` and `password` arguments.
• Added support for Linux platforms.

Version: 1.0.18

Significant changes in this version include the following:

AWS CloudHSM Client Software

Added an AWS CloudHSM client for Windows Server. The following Windows Server operating systems are currently supported:

• Microsoft Windows Server 2012 (64-bit)
• Microsoft Windows Server 2012 R2 (64-bit)
• Microsoft Windows Server 2016 (64-bit)

CNG/KSP Providers for Windows Server

• Implemented PKCS7Padding for `C_DecryptUpdate` and `C_EncryptUpdate`.
• CKA_ID no longer required for RSA private key generation.
• Improved multi-threading performance.
• Fixed various bugs.

PKCS #11 Library

• Added support for PKCS7Padding.
• Strengthened checks on key templates.
• Fixed various bugs.

OpenSSL Dynamic Engine

• Added support to `getCaviumPrivKey` for ECC-based keys.
• Improved stability when client daemon connectivity is lost.
• Fixed various bugs.

Java Library

• [Breaking Change] The `LoginManager.getInstance()` public method does not accept `username` and `password` arguments directly.
• Added support for PKCS7Padding.
• Added wrap and unwrap methods.
• Improved stability when client daemon connectivity is lost.
• Fixed various bugs.

Version 1.0.14

Significant changes in this version include the following:

AWS CloudHSM Client Software
• Improved failover behavior.
• Displays version metadata.
• Fixed various bugs.

PKCS #11 Library
• Implemented PKCS7Padding for C_DecryptUpdate and C_EncryptUpdate.
• CKA_ID no longer required for RSA private key generation.
• Improved multi-threading performance.
• Fixed various bugs.

OpenSSL Dynamic Engine
• Added support for CSRs for ECC keys.
• Improved stability and failure handling.

Java Library
No changes. Updated the version number for consistency.

Version 1.0.11

Significant changes in this version include the following:

AWS CloudHSM Client Software
• Improved load balancing.
• Improved performance.
• Improved handling of lost server connections.

PKCS #11 Library
• Added support for the CKM_RSA_PKCS_PSS sign/verify mechanism.

OpenSSL Dynamic Engine
• Updated the version for consistency.

Java Library
• Improved the performance of several algorithms.
• Added Triple DES (3DES) key import feature.
• Various bug fixes.

Version 1.0.10

Significant changes in this version include the following:

AWS CloudHSM Client Software
• Updated the key_mgmt_util command line tool to enable AES wrapped key import.
• Improved performance.
• Fixed various bugs.

PKCS #11 Library
• Updated the version for consistency.

OpenSSL Dynamic Engine
• Updated the version for consistency.

Java Library
• Added support for additional algorithms.
• Improved performance.

Version 1.0.8

Significant changes in this version include the following:

AWS CloudHSM Client Software
• Improved setup experience.
• Added respawning to the client upstart service.
• Fixed various bugs.

PKCS #11 Library
• Fixed bugs to address relative paths in the Redis setup.

OpenSSL Dynamic Engine
• Improved performance.

Java Library
• Updated the version for consistency.

Version 1.0.7

Significant changes in this version include the following:
AWS CloudHSM Client Software

- Added the `pkpspeed` (p. 348) performance testing tool.
- Fixed bugs to improve stability and performance.

PKCS #11 Library

- Added an accelerated version of the library that uses a Redis local cache to improve performance.
- Fixed bugs related to attribute handling.
- Added the ability to generate ECDSA keys.

OpenSSL Dynamic Engine

- Updated the version for consistency.

Java Library

- Added support for additional algorithms.
- Signed the JAR files for compatibility with the Sun JCE provider.

Version 1.0.0

This is the initial release.

Supported Platforms

The AWS CloudHSM client (p. 8) and related software libraries (p. 214) support 64-bit versions of the following operating systems.

Note
Earlier versions of the software do not support all listed operating systems. For detailed information about each version of the AWS CloudHSM client and related software, see AWS CloudHSM Client and Software Version History (p. 355).

Note
If you are running CloudHSM client 1.1.1 or earlier on an Amazon Linux 2 EC2 instance, see Known Issues for Amazon EC2 Instances Running Amazon Linux 2 (p. 342).

<table>
<thead>
<tr>
<th>Library</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS CloudHSM Client</td>
<td>Amazon Linux</td>
</tr>
<tr>
<td></td>
<td>Amazon Linux 2</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 6.7+</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 7.3+</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.7+</td>
</tr>
<tr>
<td></td>
<td>CentOS 7.3+</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 16.04 LTS</td>
</tr>
<tr>
<td>Library</td>
<td>Operating System</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows Server 2012</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows Server 2012 R2</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows Server 2016</td>
</tr>
<tr>
<td><strong>CNG/KSP Providers</strong></td>
<td>Microsoft Windows Server 2012</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows Server 2012 R2</td>
</tr>
<tr>
<td></td>
<td>Microsoft Windows Server 2016</td>
</tr>
<tr>
<td><strong>PKCS #11</strong></td>
<td>Amazon Linux</td>
</tr>
<tr>
<td></td>
<td>Amazon Linux 2</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 6.7+</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 7.3+</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.7+</td>
</tr>
<tr>
<td></td>
<td>CentOS 7.3+</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 16.04 LTS</td>
</tr>
<tr>
<td><strong>OpenSSL Dynamic Engine</strong></td>
<td>Amazon Linux</td>
</tr>
<tr>
<td><em>(Compatible with OpenSSL 1.0.2[f+])</em></td>
<td>Amazon Linux 2</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 6.7+</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 7.3+</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.7+</td>
</tr>
<tr>
<td></td>
<td>CentOS 7.3+</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 16.04 LTS</td>
</tr>
<tr>
<td><strong>JCE Provider</strong></td>
<td>Amazon Linux</td>
</tr>
<tr>
<td><em>(Supported on OpenJDK 1.8)</em></td>
<td>Amazon Linux 2</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 6.7</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux (RHEL) 7.3+</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.7+</td>
</tr>
<tr>
<td></td>
<td>CentOS 7.3+</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 16.04 LTS</td>
</tr>
</tbody>
</table>
# Document History

**Latest documentation update:** September 10, 2018

The following table describes the documentation release history of AWS CloudHSM after May 2018.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 3.0.0 for Windows Server platform. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>October 30, 2019</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 3.0.0 for all platforms, except Windows. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>October 22, 2019</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 2.0.4 for all platforms. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>August 26, 2019</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 2.0.3 for all platforms. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>May 13, 2019</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 2.0.1 for all platforms. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>March 21, 2019</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 2.0.0 for all platforms. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>February 6, 2019</td>
</tr>
<tr>
<td>Added region support (p. 379)</td>
<td>Added AWS CloudHSM support for the EU (Stockholm) and AWS GovCloud (US-East) regions.</td>
<td>December 19, 2018</td>
</tr>
<tr>
<td>Added new content (p. 379)</td>
<td>Released AWS CloudHSM client version 1.1.2 for Windows. For more information, see <a href="https://console.aws.amazon.com/cloudhsm/docs">AWS CloudHSM Client and Version Software History</a>.</td>
<td>November 20, 2018</td>
</tr>
</tbody>
</table>
The following table describes the documentation release history of AWS CloudHSM before June 2018.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>New content</td>
<td>Added AWS CloudHSM client and library support for Amazon Linux 2, Red Hat Enterprise Linux (RHEL) 6, Red Hat Enterprise Linux (RHEL) 7, CentOS 6, CentOS 7, and Ubuntu 16.04 LTS. For more information, see Install and Configure the AWS CloudHSM Client (Linux) (p. 34).</td>
<td>May 10, 2018</td>
</tr>
<tr>
<td>New content</td>
<td>Added a Windows AWS CloudHSM client. For more information, see the following topics:</td>
<td>April 30, 2018</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>New content</td>
<td>Added quorum authentication (M of N access control) for crypto officers (COs). For more information, see Enforcing Quorum Authentication (M of N Access Control) (p. 60).</td>
<td>November 9, 2017</td>
</tr>
<tr>
<td>Update</td>
<td>Added documentation about using the key_mgmt_util command line tool. For more information, see key_mgmt_util Command Reference (p. 125).</td>
<td>November 9, 2017</td>
</tr>
<tr>
<td>New content</td>
<td>Added Oracle Transparent Data Encryption. For more information, see Oracle Database Encryption (p. 302).</td>
<td>October 25, 2017</td>
</tr>
<tr>
<td>New content</td>
<td>Added SSL Offload. For more information, see SSL/TLS Offload (p. 270).</td>
<td>October 12, 2017</td>
</tr>
<tr>
<td>New guide</td>
<td>This release introduces AWS CloudHSM</td>
<td>August 14, 2017</td>
</tr>
</tbody>
</table>