
AWS CloudHSM

User Guide



AWS CloudHSM: User Guide

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

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
Security of backups	6
Durability of backups	7
Client SDK	7
HSM users	7
Pricing	8
Regions	8
Quotas	8
System resources	9
Getting started	10
Create IAM administrators	10
Create an IAM user and administrator group	10
Create a VPC	11
Create a private subnet	12
Create a cluster	12
Review cluster security group	14
Launch an EC2 client	14
Connect EC2 instance to cluster	16
Modify the default security group	16
Connect the Amazon EC2 instance to the AWS CloudHSM cluster	16
Create an HSM	17
Verify HSM identity (optional)	18
Overview	18
Get certificates from the HSM	20
Get the root certificates	22
Verify certificate chains	22
Extract and compare public keys	23
Initialize the cluster	24
Get the cluster CSR	24
Sign the CSR	26
Initialize the cluster	27
Install the client (Linux)	28
Install the AWS CloudHSM client and command line tools	28
Edit the client configuration	30
Install the client (Windows)	30
Activate the cluster	31
Reconfigure SSL (optional)	33
Create a key, a CSR, and then sign the CSR	33
Enable custom SSL for AWS CloudHSM	34
Build an application	36
Managing clusters	38
Connecting to the cluster	38
Place the issuing certificate	38
Bootstrap the Client SDK	39
Adding or removing HSMs	42

Adding an HSM	42
Removing an HSM	44
Deleting a cluster	45
Creating clusters from backups	46
Create clusters from backups (console)	46
Create clusters from backups (AWS CLI)	46
Create clusters from backups (AWS CloudHSM API)	47
Best practices	47
AWS CloudHSM basic operational guidelines	47
Managing backups	49
Working with backups	49
Removing expired keys or inactive users	49
Considering disaster recovery	49
Deleting and restoring backups	50
Delete and restore backups (console)	50
Delete and restore backups (AWS CLI)	50
Delete and restore backups (AWS CloudHSM API)	51
Configuring backup retention	51
Understanding backup retention policy	52
Configure backup retention (console)	52
Configure backup retention (AWS CLI)	53
Configure backup retention (AWS CloudHSM API)	54
Copying backups across Regions	54
Copy backups to different Regions (console)	54
Copy backups to different Regions (AWS CLI)	55
Copy backups to different Regions (AWS CloudHSM API)	55
Tagging resources	56
Adding or updating tags	56
Listing tags	57
Removing tags	57
Managing HSM users and keys	59
Managing HSM users	59
Understanding HSM users	59
HSM user permissions table	60
Using CMU to manage users	61
Managing 2FA	70
Managing quorum authentication	74
Managing keys	87
Key synchronization	88
Using trusted keys	93
AES key wrapping	95
Using the command line	97
Managing cloned clusters	102
Get an IP address for an HSM	103
Related topics	104
Command line tools	105
Understanding command line tools	105
CloudHSM Management Utility	106
Getting started with CMU	106
Reference	109
AWS CloudHSM management utility platform support	151
Key Management Utility	152
Getting started	152
Reference	155
Configure tool	238
Client SDK 3 tool	239
Client SDK 5 tool	245

Client SDKs	258
Choose a Client SDK version	258
Client SDK comparison	259
Check your client SDK version	259
Downloads	260
Supported platforms	314
Migrate from Client SDK 3 to Client SDK 5	317
Upgrade your Client SDK 3	317
PKCS #11 library	324
Installing	324
Authenticating	329
Key types	329
Mechanisms	329
API operations	334
Attributes	336
Samples	346
OpenSSL Dynamic Engine	347
Client SDK 3	348
Client SDK 5	350
JCE provider	352
Client SDK 3	352
Client SDK 5	369
KSP and CNG Providers	383
Verify provider installation	383
Prerequisites	385
Associate a key with a certificate	386
Code sample	387
Integrating third-party applications	392
SSL/TLS offload	392
How it works	392
SSL/TLS offload on Linux	393
SSL/TLS offload on Windows	436
Windows Server CA	448
Prerequisites	448
Create Windows Server CA	449
Sign a CSR	450
Oracle database encryption	451
Set up prerequisites	452
Configure the database	453
Microsoft SignTool	455
Microsoft SignTool with AWS CloudHSM step 1: Set up the prerequisites	456
Microsoft SignTool with AWS CloudHSM step 2: Create a signing certificate	457
Microsoft SignTool with AWS CloudHSM step 3: Sign a file	458
Java Keytool and Jarsigner	459
Use Client SDK 3 to integrate with Java Keytool and Jarsigner	459
Use Client SDK 5 to integrate with Java Keytool and Jarsigner	469
Other third-party vendor integrations	477
Monitoring	478
Client SDK logs	478
Client SDK 3 logging	478
Client SDK 5 logging	480
AWS CloudTrail	482
AWS CloudHSM information in CloudTrail	483
Understanding AWS CloudHSM log file entries	483
CloudWatch Logs	484
How logging works	485
Viewing logs	485

Interpreting logs	488
Log reference	498
CloudWatch metrics	500
Security	501
Data protection	501
Encryption at rest	502
Encryption in transit	502
End-to-end encryption	502
Identity and access management	503
Grant permissions using IAM policies	503
API actions for AWS CloudHSM	504
Condition keys for AWS CloudHSM	505
Predefined AWS managed policies for AWS CloudHSM	505
Customer managed policies for AWS CloudHSM	505
Service-linked roles	507
FIPS validation	508
Resilience	509
Infrastructure security	509
Network isolation	509
Authorization of users	510
VPC endpoints (AWS PrivateLink)	510
Considerations for AWS CloudHSM VPC endpoints	510
Creating an interface VPC endpoint for AWS CloudHSM	510
Creating a VPC endpoint policy for AWS CloudHSM	511
Update management	511
Troubleshooting	512
Known issues	512
Known issues for all HSM instances	512
Known issues for the PKCS #11 library	515
Known issues for the JCE SDK	519
Known issues for the OpenSSL Dynamic Engine	521
Known issues for Amazon EC2 instances running Amazon Linux 2	522
Known issues for integrating third-party applications	523
Lost connection	523
Keep HSM users in sync	525
Verify Performance	526
Resolving cluster creation failures	529
Add the missing permission	529
Create the service-linked role manually	530
Use a nonfederated user	530
Missing AWS CloudHSM audit logs in CloudWatch	531
Retrieving client configuration logs	531
Client SDK 3 support tool	531
Client SDK 5 support tool	531
Non-compliant AES key wraps	533
Determine whether your code generates irrecoverable wrapped keys	533
Actions you must take if your code generates irrecoverable wrapped keys	534
Client SDK 3 key synchronization failures	534
Document history	536
Recent updates	536
Earlier updates	540

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 \(p. 10\)](#).

Contents

- [AWS CloudHSM use cases \(p. 1\)](#)
- [AWS CloudHSM clusters \(p. 2\)](#)
- [AWS CloudHSM cluster backups \(p. 6\)](#)
- [AWS CloudHSM Client SDK \(p. 7\)](#)
- [HSM users \(p. 7\)](#)
- [Pricing \(p. 8\)](#)
- [Regions \(p. 8\)](#)
- [AWS CloudHSM quotas \(p. 8\)](#)

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 \(p. 392\)](#).

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 \(p. 451\)](#).

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 \(p. 8\)](#) 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 \(p. 10\)](#).

For more information about clusters, see the following topics.

Topics

- [Cluster architecture \(p. 3\)](#)

- [Cluster synchronization \(p. 4\)](#)
- [Cluster high availability and load balancing \(p. 5\)](#)

Cluster architecture

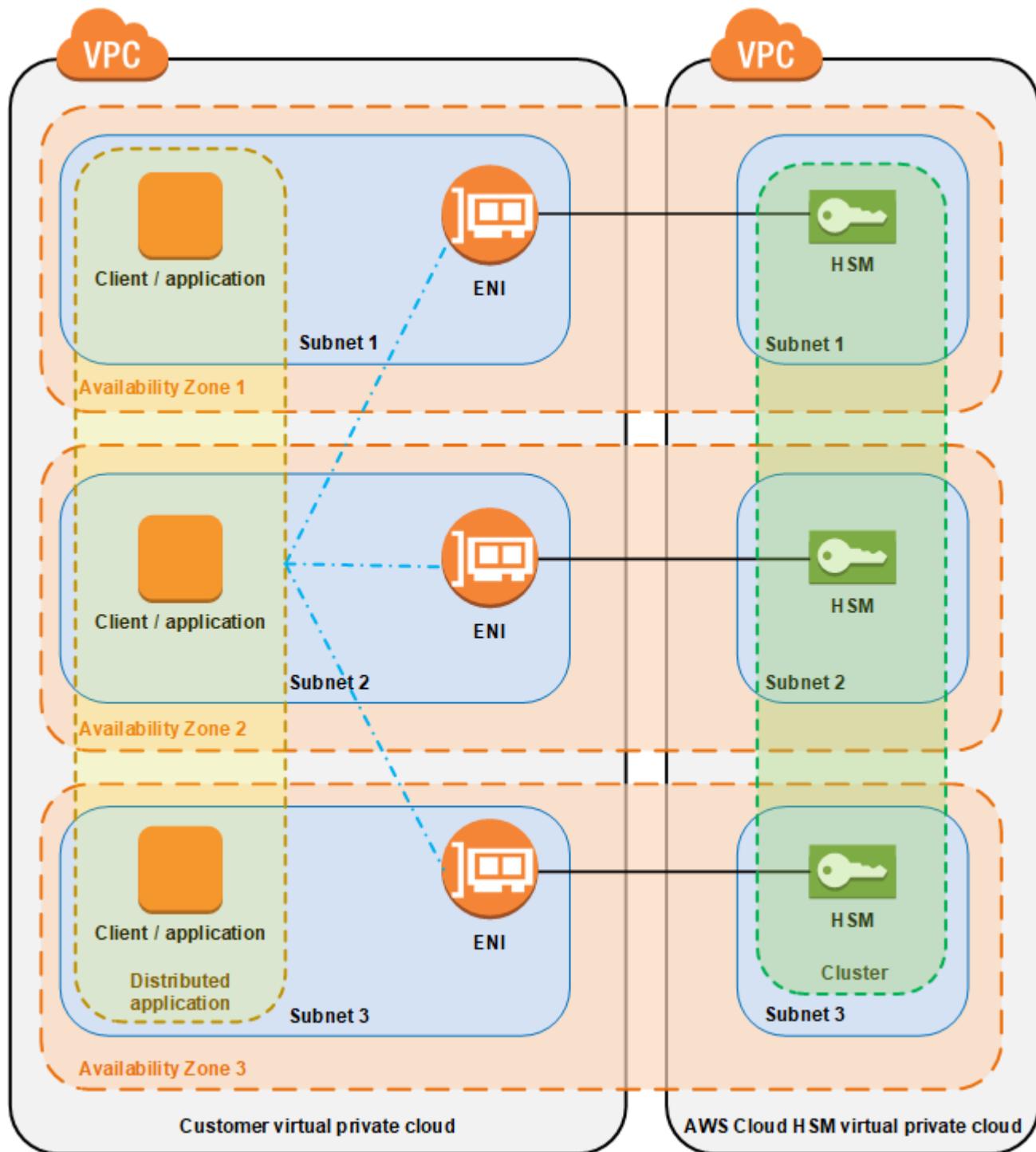
When you create a cluster, you specify an Amazon Virtual Private Cloud (VPC) in your AWS account and one or more subnets in that VPC. We recommend that you create one subnet in each Availability Zone (AZ) in your chosen AWS Region. To learn how, see [Create a private subnet \(p. 12\)](#).

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 \(p. 59\)](#). For more information, see [Keep HSM users in sync \(p. 525\)](#).

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.

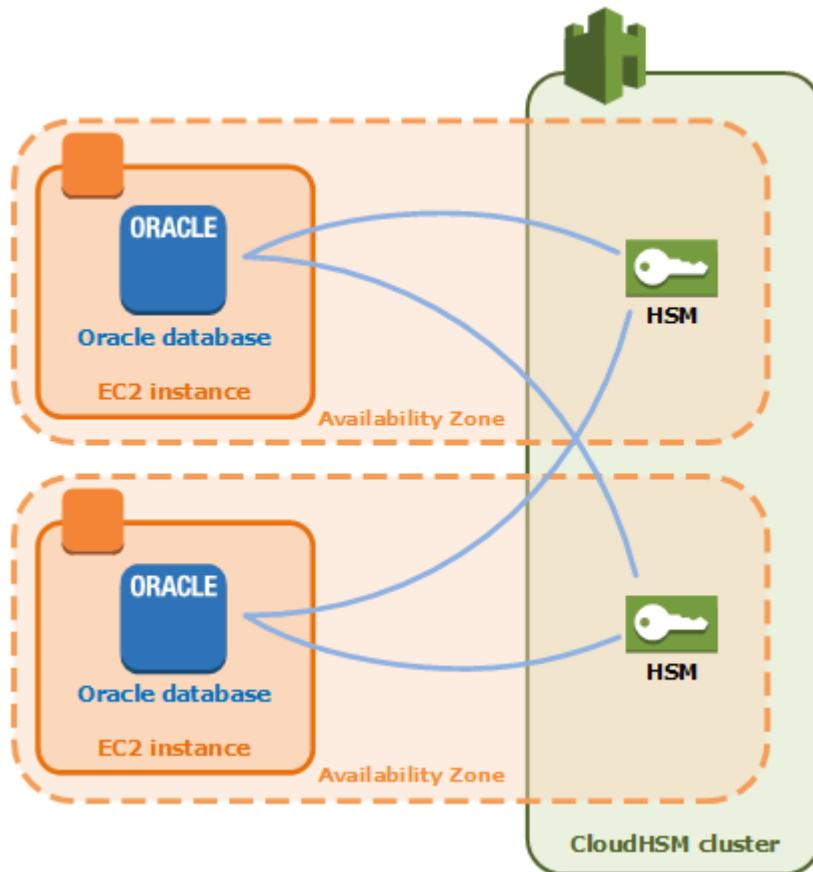
If the HSMs in a cluster fall out of synchronization, AWS CloudHSM automatically resynchronizes them. To enable this, AWS CloudHSM uses the credentials of the [appliance user \(p. 59\)](#). This user exists on all HSMs provided by AWS CloudHSM and has limited permissions. It can get a hash of objects 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

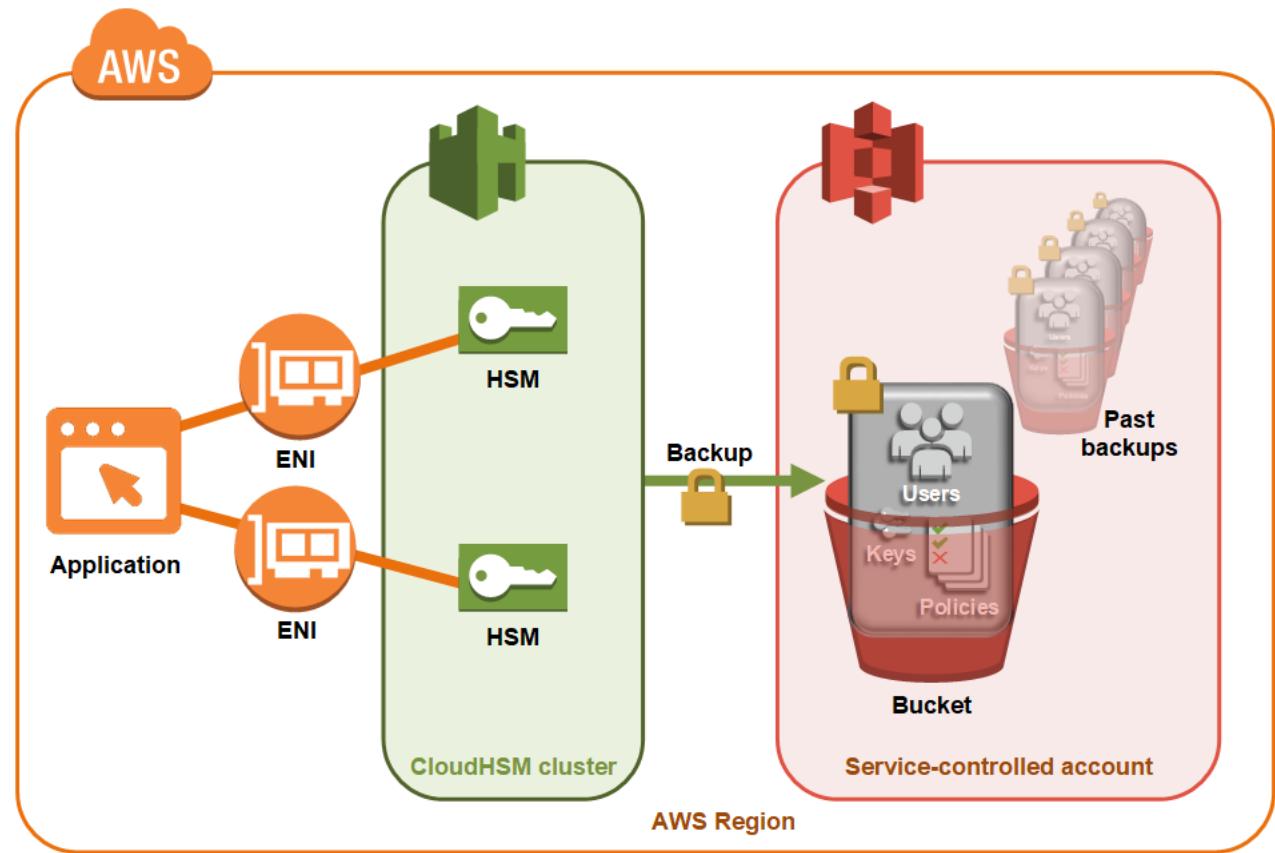
When you create an AWS CloudHSM cluster with more than one HSM, you automatically get load balancing. Load balancing means that the [AWS CloudHSM client \(p. 7\)](#) distributes cryptographic operations across all HSMs in the cluster based on each HSM's capacity for additional processing.

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 cluster backups



AWS CloudHSM makes periodic backups of the users, keys, and policies in the cluster. The service stores backups in a service-controlled Amazon Simple Storage Service (Amazon S3) bucket in the same region as your cluster. The preceding illustration shows the relationship of your backups to the cluster. Backups are secure, durable, and updated on a predictable schedule. For more information about the security and durability of backups, see the following sections. For more information about working with backups, see [Managing backups \(p. 49\)](#).

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.

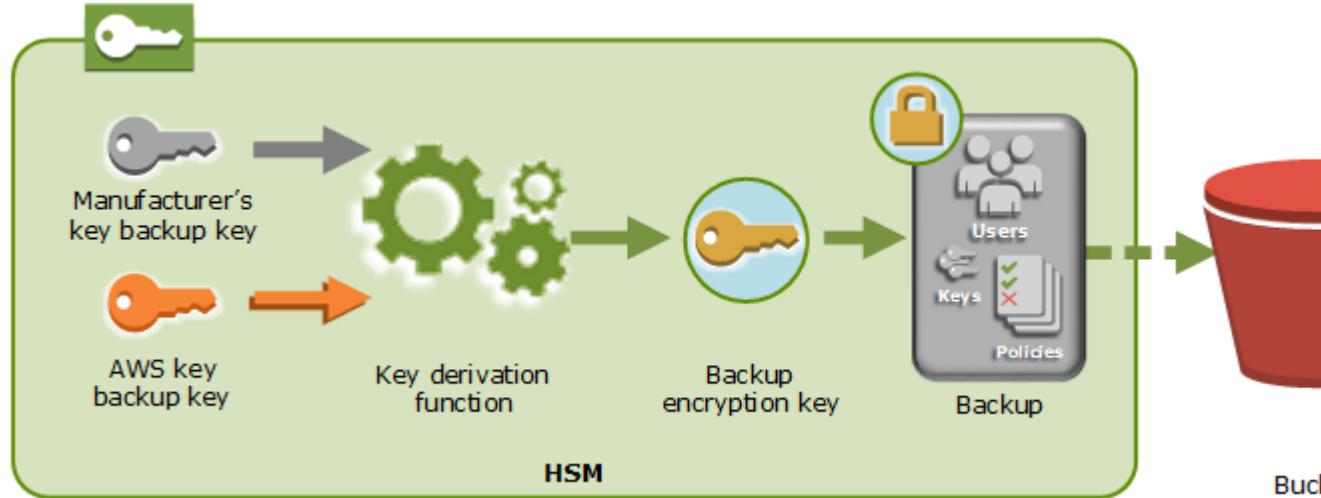
To encrypt its data, the HSM uses a unique, ephemeral encryption key known as the ephemeral backup key (EBK). The EBK is an AES 256-bit encryption key generated inside the HSM when AWS CloudHSM makes a backup. The HSM generates the EBK, then uses it to encrypt the HSM's data with a FIPS-approved AES 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 the service stores a backup in Amazon S3, the service encrypts the backup again using AWS Key Management Service (AWS KMS).

Durability of backups

AWS CloudHSM stores cluster backups in an Amazon S3 bucket that the service controls. Backups have a 99.99999999% durability level, the same as any object stored in Amazon S3.

AWS CloudHSM Client SDK

AWS CloudHSM Client SDK includes software to integrate your applications with the HSM in your cluster and to perform HSM management tasks. This software includes an implementation of Public Key Cryptography Standards #11 (PKCS #11) which describes a standard for working with hardware devices that hold cryptographic information and perform cryptographic functions. The Client SDK also includes software for off-loading cryptographic operations to the HSM from various platform or language-based applications, including applications written in Java, applications that run on Linux and use OpenSSL, or applications that use a limited set of instructions for the Microsoft Windows CNG and KSP providers.

For more information about installing and using the Client SDK or the security of the client connection, see [Client SDKs \(p. 258\)](#) and [End-to-end encryption \(p. 502\)](#).

HSM users

Unlike most AWS services and resources, you do not use AWS Identity and Access Management (IAM) users or IAM policies to access resources within your cluster. Instead, you use *HSM users* directly on the

hardware security module (HSM) with AWS CloudHSM. The HSM authenticates each HSM user by means of credentials that you define and manage. Each HSM user has a *type* that determines which operations you can perform on the HSM as that user. For more information about the different types of HSM users, see [Understanding HSM users \(p. 59\)](#).

Pricing

With AWS CloudHSM, you pay by the hour with no long-term commitments or upfront payments. For more information, see [AWS CloudHSM Pricing](#) on the AWS website.

Regions

For information about the supported Regions for AWS CloudHSM, see [AWS CloudHSM Regions and Endpoints](#) in the *AWS General Reference*, or the [Region Table](#).

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 \(p. 10\)](#) 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.

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

Resource	Default Quota	Adjustable?
Clusters	4	Yes
HSMs	6	Yes
HSMs per cluster	28	No

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

System quotas

Resource	Quota
Keys per cluster	3,300
Number of users per cluster	1,024
Maximum length of a user name	31 characters

Resource	Quota
Required password length	7 to 32 characters
Maximum number of concurrent clients	900

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.

```
#  
# 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 \(p. 10\)](#)
- [Create a virtual private cloud \(VPC\) \(p. 11\)](#)
- [Create a private subnet \(p. 12\)](#)
- [Create a cluster \(p. 12\)](#)
- [Review cluster security group \(p. 14\)](#)
- [Launch an Amazon EC2 client instance \(p. 14\)](#)
- [Connect Amazon EC2 instance to AWS CloudHSM cluster \(p. 16\)](#)
- [Create an HSM \(p. 17\)](#)
- [Verify the identity and authenticity of your cluster's HSM \(optional\) \(p. 18\)](#)
- [Initialize the cluster \(p. 24\)](#)
- [Install and configure the AWS CloudHSM client \(Linux\) \(p. 28\)](#)
- [Install and configure the AWS CloudHSM client \(Windows\) \(p. 30\)](#)
- [Activate the cluster \(p. 31\)](#)
- [Reconfigure SSL with a new certificate and private key \(optional\) \(p. 33\)](#)
- [Build an application \(p. 36\)](#)

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 \(p. 10\)](#) 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 \(p. 505\)](#) 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 \(p. 507\)](#).

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. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user that follows 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 users**.
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 \(p. 503\)](#).

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 **Launch VPC Wizard**.
4. For **Resources to create**, select **VPC, subnets, etc.**
5. For **Name tag auto-generation**, type an identifiable name such as **CloudHSM**.
6. For **DNS options**, check **Enable DNS hostnames**.
7. Leave all other options set to their defaults.
8. Choose **Create VPC**.
9. After the VPC is created, choose **View VPC** to view the details of the VPC you just created.

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](#) in the *AWS General Reference* or the [AWS Region Table](#) 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/>.
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.

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 \(p. 14\)](#) and [configure the cluster's security group \(p. 14\)](#) to allow communication and connections with the HSM.

Important

When you create a cluster, AWS CloudHSM creates a [service-linked role](#) named **AWSServiceRoleForCloudHSM**. 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 \(p. 529\)](#). For more information about service-linked roles, see [Service-linked roles for AWS CloudHSM \(p. 507\)](#).

You can create a cluster from the [AWS CloudHSM console](#), the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To create a cluster (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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**.
6. Specify how long the service should retain backups.
Accept the default retention period of 90 days or type a new value between 7 and 379 days. The service will automatically delete backups in this cluster older than the value you specify here. You can change this later. For more information, see [Configuring backup retention \(p. 51\)](#).
7. Choose **Next**.
8. (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
9. Choose **Review**.
10. 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, the backup retention period, 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 \
    --backup-retention-policy DAYS <number of days to retain backups> \
    --subnet-ids <subnet ID 1> <subnet ID 2> <subnet ID N>

{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
        },
        "VpcId": "vpc-50ae0636",
        "SubnetMapping": {
            "us-west-2b": "subnet-49a1bc00",
            "us-west-2c": "subnet-6f950334",
            "us-west-2a": "subnet-fd54af9b"
        },
        "SecurityGroup": "sg-6cb2c216",
        "HsmType": "hsm1.medium",
        "Certificates": {}
    }
}
```

```
        "State": "CREATE_IN_PROGRESS",
        "Hsms": [],
        "ClusterId": "cluster-igklspoyj5v",
        "CreateTimestamp": 1502423370.069
    }
}
```

To create a cluster (AWS CloudHSM API)

- Send a [CreateCluster](#) request. Specify the HSM instance type, the backup retention policy, 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 \(p. 529\)](#).

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 \(p. 14\)](#) and connect it to your HSMs by [attaching the cluster security group \(p. 16\)](#) 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/>.
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/>.
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/>.

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, Security**, 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:

- 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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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.

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-zone <Availability Zone>
{
    "Hsm": {
        "HsmId": "hsm-ted36yp5b2x",
        "EniIp": "10.0.1.12",
```

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

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 \(p. 18\)](#), or proceed directly to [Initialize the cluster \(p. 24\)](#).

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

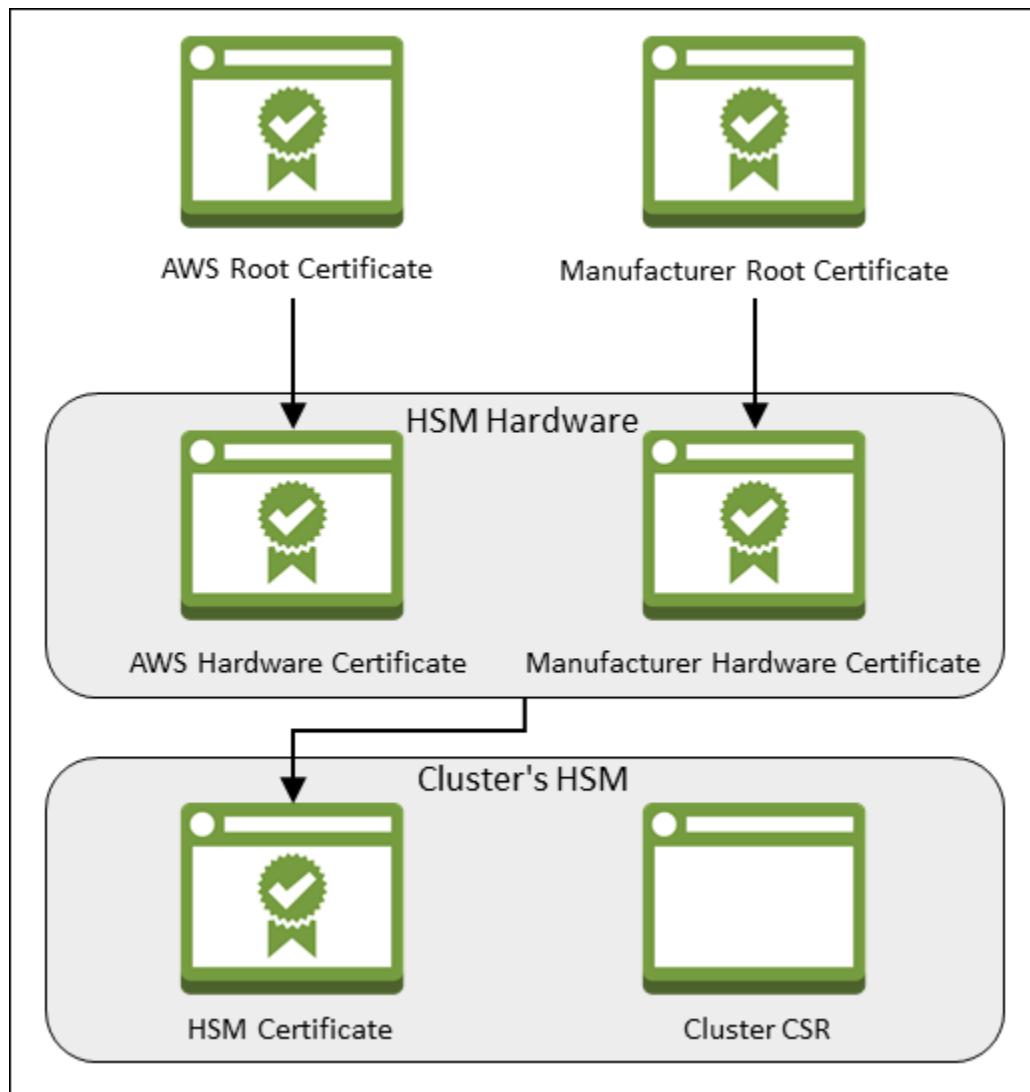
- [Overview \(p. 18\)](#)
- [Get certificates from the HSM \(p. 20\)](#)
- [Get the root certificates \(p. 22\)](#)
- [Verify certificate chains \(p. 22\)](#)
- [Extract and compare public keys \(p. 23\)](#)

Overview

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

- [Get the certificates and CSR \(p. 20\)](#) – 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.
- [Verify the certificate chains \(p. 22\)](#) – 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.
- [Compare public keys \(p. 23\)](#) – 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.

Manufacturer Root Certificate

This is the hardware manufacturer's root certificate.

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 \(p. 26\)](#), you claim the cluster. Then, you can use the signed CSR to [initialize the cluster \(p. 27\)](#).

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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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**. If you're following the getting started, you did this in a [previous step \(p. 17\)](#).
4. 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](#).

 Cluster CSR

Cluster verification certificate

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

 HSM certificate

Choose each link to download and save the CSR and certificates. To simplify the subsequent steps, save all of the files to the same directory and use the default file names.

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.
 - Issue the following command to extract the cluster CSR. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```
$ aws clouhdsmv2 describe-clusters --filters clusterIds=<cluster ID> \
--output text \
--query 'Clusters[0].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.

```
$ 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.

```
$ 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.

```
$ 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

- Download the AWS CloudHSM root certificate: [AWS_CloudHSM_Root-G1.zip](#)
- Download the manufacturer root certificate: [liquid_security_certificate.zip](#)

To download the certificate from the landing page, <https://www.marvell.com/products/security-solutions/liquid-security-hsm-adapters-and-appliances/liquidsecurity-certificate.html>, 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.

- After you download the files, extract (unzip) the contents.

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 \(p. 22\)](#) and [HSM certificate files \(p. 20\)](#) 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.

To verify the HSM certificate with the AWS CloudHSM root certificate

1. Navigate to the directory where you saved the [root certificate \(p. 22\)](#) and [HSM certificate files \(p. 20\)](#) 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.

```
$ 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.

```
$ 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.

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

```
$ 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.

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

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

```
$ 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.

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

After you verify the identity and authenticity of the HSM, proceed to [Initialize the cluster \(p. 24\)](#).

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 \(p. 18\)](#). 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 \(p. 24\)](#)
- [Sign the CSR \(p. 26\)](#)
- [Initialize the cluster \(p. 27\)](#)

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 \(p. 18\)](#), 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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. Choose **Initialize** next to the cluster that you [created previously \(p. 12\)](#).
3. Choose an Availability Zone (AZ) for the HSM that you are creating. Then choose **Create**. If you're following the getting started, you did this in a [previous step \(p. 17\)](#).
4. 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](#).

 Cluster CSR

Cluster verification certificate

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

 HSM certificate

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. 12).

```
$ 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 \(p. 24\)](#)).
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.

```
$ 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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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**. If you're following the getting started, you did this in a [previous step \(p. 17\)](#).
4. 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**.
5. On the **Sign certificate signing request (CSR)** page, choose **Next**.
6. 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
{
    "State": "INITIALIZE_IN_PROGRESS",
    "StateMessage": "Cluster is initializing. State will change to INITIALIZED upon
completion."
}
```

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 \(Windows\) \(p. 30\)](#).

Tasks

- [Install the AWS CloudHSM client and command line tools \(p. 28\)](#)
- [Edit the client configuration \(p. 30\)](#)

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.

Amazon Linux

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

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

Amazon Linux 2

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

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

CentOS 7

```
sudo yum install wget
```

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

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

CentOS 8

```
sudo yum install wget
```

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

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

RHEL 7

```
sudo yum install wget
```

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

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

RHEL 8

```
sudo yum install wget
```

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

```
sudo yum install ./clouhdsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/clouhdsmv2-software/CloudHsmClient/Xenial/clouhdsm-client_latest_amd64.deb
```

```
sudo apt install ./clouhdsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/clouhdsmv2-software/CloudHsmClient/Bionic/clouhdsm-client_latest_u18.04_amd64.deb
```

```
sudo apt install ./clouhdsm-client_latest_u18.04_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. 26)—to the following location on the client instance: /opt/clouhdsm/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 \(p. 238\)](#) 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/clouhdsm/bin/configure -a <IP address>  
Updating server config in /opt/clouhdsm/etc/clouhdsm_client.cfg  
Updating server config in /opt/clouhdsm/etc/clouhdsm_mgmt_util.cfg
```

3. Go to [Activate the cluster \(p. 31\)](#).

Install and configure the AWS CloudHSM client (Windows)

To work with an HSM in your AWS CloudHSM cluster on Windows, you need the AWS CloudHSM client software for Windows. You should install it on the Windows Server instance that you created previously.

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

1. Connect to your Windows Server instance.

2. Download the [AWSCloudHSMClient-latest.msi installer](#).
3. Go to your download location and run the installer ([AWSCloudHSMClient-latest.msi](#)) with administrative privilege.
4. Follow the installer instructions, then choose **Close** after the installer has finished.
5. Copy your self-signed issuing certificate—[the one that you used to sign the cluster certificate \(p. 26\)](#)—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 \(p. 31\)](#).

Notes:

- If you are updating the client, existing configuration files from previous installations are *not* overwritten.
- The AWS CloudHSM client installer for Windows automatically registers the Cryptography API: Next Generation (CNG) and key storage provider (KSP). To uninstall the client, run the installer again and follow the uninstall instructions.
- If you are using Linux, you can install the Linux client. For more information, see [Install and configure the AWS CloudHSM client \(Linux\) \(p. 28\)](#).

Activate the cluster

When you activate an AWS CloudHSM cluster, the cluster's state changes from initialized to active. You can then [manage the hardware security module \(HSM\) users \(p. 59\)](#) and [use the HSM \(p. 258\)](#).

To activate the cluster, log in to the HSM with the credentials of the [precrypto officer \(PRECO\) \(p. 59\)](#). 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 \(p. 14\)](#). You can launch a Linux instance or a Windows Server.
2. Use the following command to start the CloudHSM Management Utility (CMU) command line utility.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. (Optional) Use the **listUsers** command to display the existing users.

```
aws-cloudhsm>listUsers
```

```
Users on server 0(server1):
Number of users found:2
```

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PRECO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		

4. 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
loginHSM success on server 0(server1)
```

5. 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).

```
aws-cloudhsm>changePswd PRECO admin <NewPassword>
*****
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****

Do you want to continue(y/n)?y
Changing password for 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.

6. (Optional) Use the **listUsers** command to verify that the user's type changed to [crypto officer \(CO\) \(p. 60\)](#).

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

User Id          User Type      User Name      MofnPubKey
LoginFailureCnt 2FA
1               CO            admin           NO
0               NO
2               AU            app_user        NO
0               NO
```

7. Use the **quit** command to stop the cloudhsm_mgmt_util tool.

```
aws-cloudhsm>quit
```

For more information about working with CMU, see [Understanding HSM Users \(p. 59\)](#) and [Understanding HSM User Management with CMU \(p. 61\)](#).

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. 26) your cluster.

At a high level, this is a two-step process:

1. First, you create a private key, then use that key to create a certificate signing request (CSR). Use the issuing certificate, the certificate you created when you initialized the cluster, to sign the CSR.
2. Next, you use the configure tool to copy the key and certificate to the appropriate directories.

Create a key, a CSR, and then sign the CSR

The steps are the same for Client SDK 3 or Client SDK 5.

To reconfigure SSL with a new certificate and private key

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

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

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
```

Enable custom SSL for AWS CloudHSM

The steps are different for Client SDK 3 or Client SDK 5. For more information about working with the configure command line tool, see [???](#) (p. 238).

Topics

- [Custom SSL for Client SDK 3 \(p. 34\)](#)
- [Custom SSL for Client SDK 5 \(p. 35\)](#)

Custom SSL for Client SDK 3

Use the configure tool for Client SDK 3 to enable custom SSL. For more information about configure tool for Client SDK 3, see [???](#) (p. 239).

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 3 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify ssl-client.crt and ssl-client.key.

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

3. 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.

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

Create a directory.

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

Create a file that contains the certificate with the hash name.

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

Custom SSL for Client SDK 5

Use any of the Client SDK 5 configure tools to enable custom SSL. For more information about configure tool for Client SDK 5, see [???](#) (p. 245).

PKCS #11 library

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify ssl-client.crt and ssl-client.key.

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

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

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

2. With a PowerShell interpreter, use the configure tool to specify ssl-client.crt and ssl-client.key.

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

OpenSSL Dynamic Engine

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify ssl-client.crt and ssl-client.key.

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

JCE provider

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify ssl-client.crt and ssl-client.key.

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

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

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

2. With a PowerShell interpreter, use the configure tool to specify ssl-client.crt and ssl-client.key.

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

Build an application

Build applications and work with keys using AWS CloudHSM.

To get started creating and using keys in your new cluster, you must first create a hardware security module (HSM) user with CloudHSM Management Utility (CMU). For more information, see [Understanding HSM User Management Tasks \(p. 61\)](#), [Getting Started with CMU \(p. 106\)](#), and [How to Manage HSM Users \(p. 64\)](#).

With HSM users in place, you can log in to the HSM and create and use keys with any of the following options:

- Use [key management utility, a command line tool \(p. 152\)](#)
- Build a C application using the [PKCS #11 library \(p. 324\)](#)
- Build a Java application using the [JCE provider \(p. 352\)](#)
- Use the [OpenSSL Dynamic Engine directly from the command line \(p. 347\)](#)
- Use the OpenSSL Dynamic Engine for TLS offload with [NGINX and Apache web servers \(p. 392\)](#)
- Use the CNG and KSP providers to use AWS CloudHSM with [Microsoft Windows Server Certificate Authority \(CA\) \(p. 448\)](#)
- Use the CNG and KSP providers to use AWS CloudHSM with [Microsoft Sign Tool \(p. 455\)](#)

- Use the CNG and KSP providers for TLS offload with [Internet Information Server \(IIS\) web server \(p. 392\)](#)

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 \(p. 10\)](#).

Topics

- [Connect the client SDK to the AWS CloudHSM cluster \(p. 38\)](#)
- [Adding or removing HSMs in an AWS CloudHSM cluster \(p. 42\)](#)
- [Deleting an AWS CloudHSM cluster \(p. 45\)](#)
- [Creating AWS CloudHSM clusters from backups \(p. 46\)](#)
- [Best practices for AWS CloudHSM \(p. 47\)](#)

Connect the client SDK to the AWS CloudHSM cluster

To connect to the cluster with either Client SDK 5 or Client SDK 3, you must first do two things:

- Have an issuing certificate in place on the EC2 instance
- Bootstrap the Client SDK to the cluster

Place the issuing certificate

You create the issuing certificate when you initialize the cluster. Copy the issuing certificate to the default location for the platform on each EC2 instance that connects to the cluster.

Linux

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

Windows

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

With Client SDK 5 you can use the configure tool to specify the location of the issuing certificate.

PKCS #11 library

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

OpenSSL Dynamic Engine

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

JCE provider

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

For more information, see [Configure Tool \(p. 245\)](#).

For more information about initializing the cluster or creating and signing the certificate, see [Initialize the Cluster \(p. 27\)](#).

Bootstrap the Client SDK

The bootstrap process is different depending on the version of the Client SDK you're using, but you must have the IP address of one of the hardware security modules (HSM) in the cluster. You can use the IP

address of any HSM attached to your cluster. After the Client SDK connects, it retrieves the IP addresses of any additional HSMs and performs load balancing and client-side key synchronization operations.

To get an IP address for the cluster

To get an IP address for a HSM (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Clusters**.
4. To open the cluster detail page, in the cluster table, choose the cluster ID.
5. To get the IP address, on the HSMs tab, choose one of the IP addresses listed under **ENI IP address**.

To get an IP address for a HSM (AWS CLI)

- Get the IP address of an HSM by using the **describe-clusters** command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of **EniIp**.

```
$ aws clouhsmv2 describe-clusters

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

For more information about bootstrapping, see [Configure Tool \(p. 238\)](#).

To bootstrap Client SDK 5

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

To bootstrap Client SDK 3

To bootstrap a Linux EC2 instance for Client SDK 3

- Use **configure** to specify the IP address of a HSM in your cluster.

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

To bootstrap a Windows EC2 instance for Client SDK 3

- Use **configure** to specify the IP address of a HSM in your cluster.

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

For more information about **configure**, see [???](#) (p. 238).

Adding or removing HSMs in an AWS CloudHSM cluster

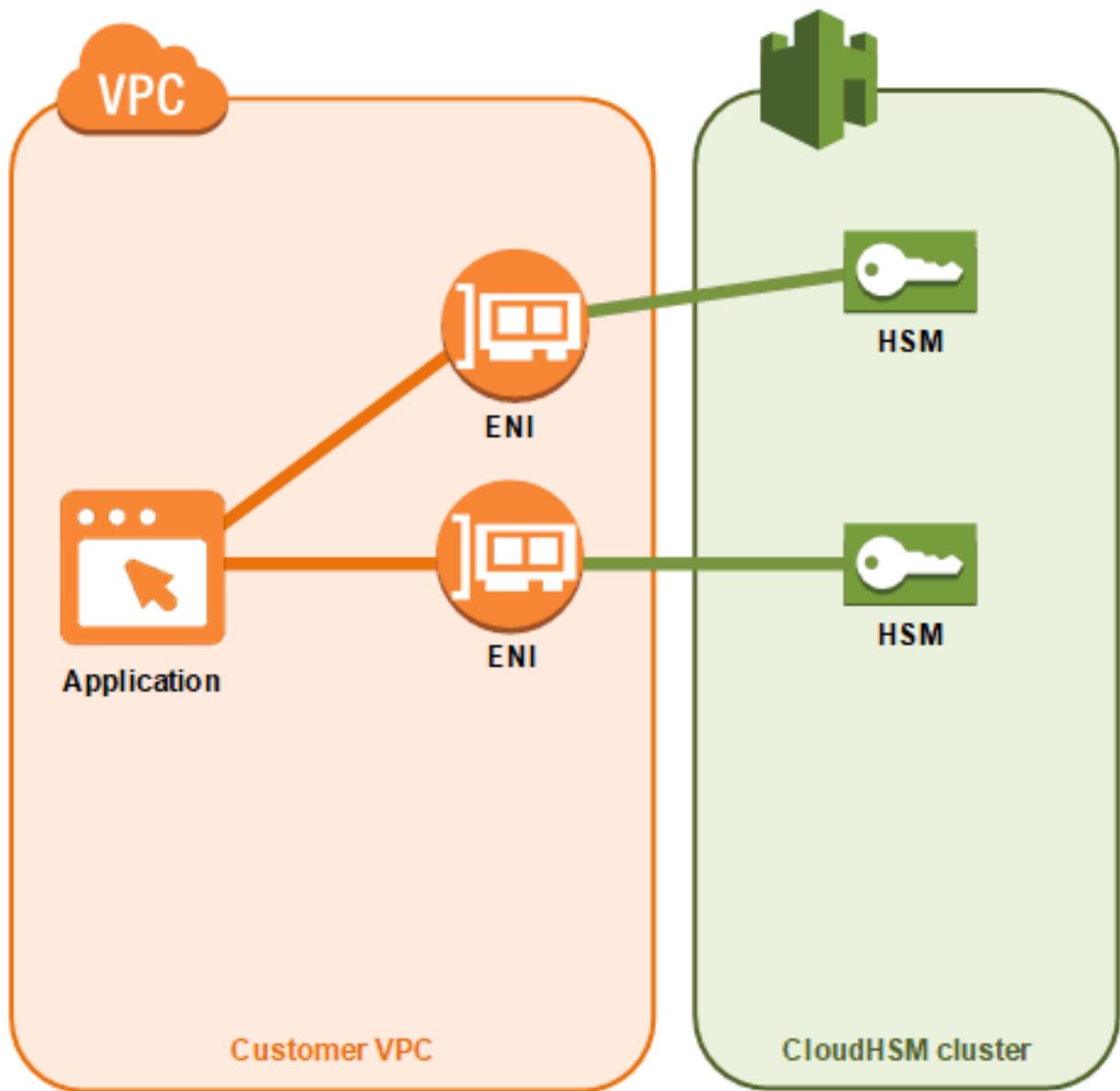
To scale up or down your AWS CloudHSM cluster, add or remove HSMs by using the [AWS CloudHSM console](#) or one of the [AWS SDKs or command line tools](#).

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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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-5a73d5qzrdh",
        "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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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**)

If you don't know the values for these identifiers, issue the [describe-clusters](#) command.

```
$ aws clouhdsmv2 delete-hsm --cluster-id <cluster ID> --eni-ip <HSM IP address>
{
    "HsmId": "hsm-lgavqitns2a"
}
```

To remove an HSM (AWS CloudHSM API)

- Send a [DeleteHsm](#) request, specifying the cluster ID and an identifier for the HSM that you are deleting.

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)

- Open the AWS CloudHSM console at <https://console.aws.amazon.com/clouhdsm/>.
- Choose the cluster that you are deleting. Then choose **Delete cluster**.
- 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.

```
$ aws clouhdsmv2 delete-cluster --cluster-id <cluster ID>
{
    "Cluster": {
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "SourceBackupId": "backup-rtq2dwi2gg6",
        "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.

Creating AWS CloudHSM clusters from backups

To restore an AWS CloudHSM cluster from a backup, create a cluster and specify the backup to restore. After you create the cluster, don't initialize or activate it. Add a HSM to the cluster and your cluster will contain the same users, key material, certificates, configuration, and policies that were in the backup. For more information about managing backups, see [Managing backups \(p. 49\)](#).

Create clusters from backups (console)

To create a cluster from a previous backup (console)

- Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
- Choose **Create cluster**.
- In the **Cluster configuration** section, do the following:
 - For **VPC**, choose a VPC for the cluster that you are creating.
 - For **AZ(s)**, choose a private subnet for each Availability Zone that you are adding to the cluster.
- In the **Cluster source** section, do the following:
 - Choose **Restore cluster from existing backup**.
 - Choose the backup that you are restoring.
- Choose **Next: Review**.
- Review your cluster configuration, then choose **Create cluster**.
- Specify how long the service should retain backups.
Accept the default retention period of 90 days or type a new value between 7 and 379 days. The service will automatically delete backups in this cluster older than the value you specify here. You can change this later. For more information, see [Configuring backup retention \(p. 51\)](#).
- Choose **Next**.
- (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
- Choose **Review**.
- Review your cluster configuration, and then choose **Create cluster**.

Tip

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

Create clusters from backups (AWS CLI)

To determine the backup ID, issue the **describe-backups** command.

To create clusters from backups (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.

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

Create clusters from backups (AWS CloudHSM API)

Refer to the following topic to learn how to create clusters from backups by using the API.

- [CreateCluster](#)

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 backups

AWS CloudHSM makes periodic backups of your cluster at least once every 24 hours. Each backup contains encrypted copies of the following data:

- Users (COs, CUs, and AUs)
- Key material and certificates
- Hardware security module (HSM) configuration and policies

You can't instruct the service to make backups, but you can take certain actions that force the service to create a backup. The service makes a backup when you perform any of the following actions:

- Activate a cluster
- Add an HSM to an active cluster
- Remove an HSM from an active cluster

AWS CloudHSM deletes backups based on the backup retention policy you set when you create clusters. For information about managing backup retention policy, see [Configuring backup retention \(p. 51\)](#).

Topics

- [Working with backups \(p. 49\)](#)
- [Deleting and restoring backups \(p. 50\)](#)
- [Configuring AWS CloudHSM backup retention policy \(p. 51\)](#)
- [Copying backups across AWS Regions \(p. 54\)](#)

Working with backups

When you add an HSM to a cluster that previously contained one or more active HSMs, the service restores a backup onto the new HSM. Use backups to manage a HSM that you use infrequently. When you don't need the HSM, delete it to trigger a backup. Later, when you do need the HSM, create a new one in the same cluster, and this action will restore the backup you previously created with the delete HSM operation.

Removing expired keys or inactive users

You may want to remove unwanted cryptographic materials from your environment such as expired keys or inactive users. This is a two-step process. First, delete these materials from your HSM. Next, delete all existing backups. Following this process ensures you do not restore deleted information when initializing a new cluster from backup. For more information, see [the section called "Deleting and restoring backups" \(p. 50\)](#).

Considering disaster recovery

You can create a cluster from a backup. You might want to do this to set a recovery point for your cluster. Nominate a backup that contains all the users, key material, certificates that you want in your recovery

point, and then use that backup to create a new cluster. For more information about creating a cluster from a backup, see [Creating clusters from backups \(p. 46\)](#).

You can also copy a backup of a cluster into a different region, where you can create a new cluster as a clone of the original. You may want to do this for a number of reasons, including simplification of the disaster recovery process. For more information about copying backups to regions, see [Copying backups across Regions \(p. 54\)](#).

Deleting and restoring backups

After you delete a backup, the service holds the backup for seven days, during which time you can restore the backup. After the seven-day period, you can no longer restore the backup. For more information about managing backups, see [Managing backups \(p. 49\)](#).

Delete and restore backups (console)

To delete a backup (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup to delete.
5. To delete the selected backup, choose **Actions, Delete**.
The Delete backups dialog box appears.
6. Choose **Delete**.

The state of the back changes to `PENDING_DELETE`. You can restore a backup that is pending deletion for up to 7 days after you request the deletion.

To restore a backup (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup in the `PENDING_DELETE` state to restore.
5. To restore the selected backup, choose **Actions, Restore**.

Delete and restore backups (AWS CLI)

Check the status of a backup or find its ID by using the `describe-backups` command from the AWS CLI.

To delete a backup (AWS CLI)

- At a command prompt, run the `delete-backup` command, passing the ID of the backup to be deleted.

```
$ aws cloudhsmv2 delete-backup --backup-id <backup ID>
{
    "Backup": {
```

```
        "CreateTimestamp": 1534461854.64,
        "ClusterId": "cluster-dygnwhmscg5",
        "BackupId": "backup-ro5c4er4aac",
        "BackupState": "PENDING_DELETION",
        "DeleteTimestamp": 1536339805.522
    }
}
```

To restore a backup (AWS CLI)

- To restore a backup, issue the **restore-backup** command, passing the ID of a backup that is in the PENDING_DELETION state.

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

To list backups (AWS CLI)

- 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.

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

Delete and restore backups (AWS CloudHSM API)

Refer to the following topics to learn how to delete and restore backups by using the API.

- [DeleteBackup](#)
- [RestoreBackup](#)

Configuring AWS CloudHSM backup retention policy

The default backup retention policy for clusters is 90 days (with an [exemption for existing clusters \(p. 52\)](#)). That means that we delete all HSM backups older than 90 days. You can set this

period to any number between 7 and 379 days. For more information about managing backups, see [Managing backups \(p. 49\)](#).

Understanding backup retention policy

AWS CloudHSM purges backups based on the backup retention policy you set when you create a cluster. Backup retention policy applies to clusters. If you move a backup to a different region, that backup is no longer associated with a cluster and has no backup retention policy. You must manually delete any backups not associated with a cluster. AWS CloudHSM does not delete the last backup for a deleted cluster or a cluster that contains no hardware security modules (HSM).

[AWS CloudTrail \(p. 482\)](#) reports backups marked for deletion. You can restore backups the service purges just as you would restore [manually deleted backups \(p. 50\)](#). To prevent a race condition, you should change the backup retention policy for the cluster before you restore a backup deleted by the service. If you want to keep the retention policy the same and preserve select backups, you can specify that the service [exclude backups \(p. 52\)](#) from the cluster backup retention policy.

Existing-cluster exemption

AWS CloudHSM launched managed backup retention on 18 November 2020. Clusters created before 18 November 2020 have a backup retention policy of 90 days plus the age of the cluster. For example, if you created a cluster on 18 November 2019, the service would assign your cluster a backup retention policy of one year plus 90 days (455 days).

Note

You can opt out of managed backup retention altogether by contacting support (<https://aws.amazon.com/support>).

Configure backup retention (console)

To configure backup retention policy (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. Click the cluster ID of a cluster in the Active state to manage the backup retention policy for that cluster.
4. To change the backup retention policy, choose **Actions, Change backup retention period**.

The Change backup retention period dialog box appears.

5. In **Backup retention period (in days)**, type a value between 7 and 379 days.
6. Choose **Change backup retention period**.

To exclude or include a backup from backup retention policy (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To view your backups, in the navigation pane choose **Backups**.
3. Click the backup ID of a backup in the Ready state to exclude or include.
4. On the **Backup details** page, take one of the following actions.
 - To exclude a backup with a date in **Expiration time**, choose **Actions, Disable expiration**.
 - To include a backup that does not expire, choose **Actions, Use cluster retention policy**.

Configure backup retention (AWS CLI)

Check the status of a backup or find its ID by using the [describe-backups](#) command from the AWS CLI.

To configure backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-cluster** command. Specify the cluster ID and the backup retention policy.

```
$ aws cloudhsmv2 modify-cluster --cluster-id <cluster ID> \
                                 --backup-retention-policy Type=DAYs,Value=<number of
                                 days to retain backups>
{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYs",
            "Value": 90
        },
        "Certificates": {},
        "ClusterId": "cluster-kdmrarrayc7gi",
        "CreateTimestamp": 1504903546.035,
        "Hsms": [],
        "HsmType": "hsm1.medium",
        "SecurityGroup": "sg-40399d28",
        "State": "ACTIVE",
        "SubnetMapping": {
            "us-east-2a": "subnet-f1d6e798",
            "us-east-2c": "subnet-0e358c43",
            "us-east-2b": "subnet-40ed9d3b"
        },
        "TagList": [
            {
                "Key": "Cost Center",
                "Value": "12345"
            }
        ],
        "VpcId": "vpc-641d3c0d"
    }
}
```

To exclude a backup from backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-backup-attributes** command. Specify the backup ID and set the never-expires flag to preserve the backup.

```
$ aws cloudhsmv2 modify-backup-attributes --backup-id <backup ID> \
                                         --never-expires
{
    "Backup": {
        "BackupId": "backup-ro5c4er4aac",
        "BackupState": "READY",
        "ClusterId": "cluster-dygnwhmscg5",
        "NeverExpires": true
    }
}
```

To include a backup in backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-backup-attributes** command. Specify the backup ID and set the no-never-expires flag to include the backup in backup retention policy, which means the service will eventually delete the backup.

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

Configure backup retention (AWS CloudHSM API)

Refer to the following topics to learn how to manage backup retention by using the API.

- [ModifyCluster](#)
- [ModifyBackupAttributes](#)

Copying backups across AWS Regions

You can copy backups across regions for many reasons, including cross-region resilience, global workloads, and [disaster recovery \(p. 49\)](#). After you copy backups, they appear in the destination region with a `CREATE_IN_PROGRESS` status. Upon successful completion of the copy, the status of the backup changes to `READY`. If the copy fails, the status of the backup changes to `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 about backups or how to create a cluster from a backup, see [Managing backups \(p. 49\)](#) or [Creating clusters from backups \(p. 46\)](#).

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 [Create IAM administrators \(p. 10\)](#).
- 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 clusters \(p. 38\)](#).
- 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.

Copy backups to different Regions (console)

To copy backups to different Regions (console)

- Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.

2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup to copy to a different region.
5. To copy the selected backup, choose **Actions**, **Copy backup to another region**.
The Copy backup to another region dialog box appears.
6. In **Destination region**, choose a region from **Select a region**.
7. (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
8. Choose **Copy backup**.

Copy backups to different Regions (AWS CLI)

To determine the cluster ID or backup ID, run the [describe-clusters](#) or [describe-backups](#) command respectively.

To copy backups to different regions (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.

```
$ 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"
    }
}
```

Copy backups to different Regions (AWS CloudHSM API)

Refer to the following topic to learn how to copy backups to different regions by using the API.

- [CopyBackupToRegion](#)

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 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.

Topics

- [Adding or updating tags \(p. 56\)](#)
- [Listing tags \(p. 57\)](#)
- [Removing tags \(p. 57\)](#)

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**.

Note

If you update the tag key for an existing tag, the console deletes the existing tag and creates a new one.

- 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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
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.

```
$ 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)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.

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.

```
$ 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 managing HSM users and keys in AWS CloudHSM. You can also learn how to use quorum authentication (also known as M of N access control).

Topics

- [Managing HSM users in AWS CloudHSM \(p. 59\)](#)
- [Managing keys in AWS CloudHSM \(p. 87\)](#)
- [Managing cloned clusters \(p. 102\)](#)

Managing HSM users in AWS CloudHSM

In AWS CloudHSM, you must use [CloudHSM Management Utility \(p. 106\)](#) (CMU), a command line tool to create and manage the users on your HSM. You can also setup quorum authentication. For more information about the types of users available and what actions those users can perform, see [Understanding HSM users \(p. 59\)](#).

Topics

- [Understanding HSM users \(p. 59\)](#)
- [HSM user permissions table \(p. 60\)](#)
- [Using CloudHSM Management Utility \(CMU\) to manage users \(p. 61\)](#)
- [Managing two-factor authentication \(2FA\) for crypto officers \(p. 70\)](#)
- [Managing quorum authentication \(M of N access control\) \(p. 74\)](#)

Understanding HSM users

Most operations that you perform on the HSM require the credentials of an *HSM user*. The HSM authenticates each HSM user and each HSM user has a *type* that determines which operations you can perform on the HSM as that user.

Users

- [Precrypto officer \(PRECO\) \(p. 59\)](#)
- [Crypto officer \(CO | PCO\) \(p. 60\)](#)
- [Crypto user \(CU\) \(p. 60\)](#)
- [Appliance user \(AU\) \(p. 60\)](#)

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. The PRECO user can only change its own password and perform read-only operations on the HSM. You

use the PRECO user to activate a cluster. To [activate a cluster \(p. 31\)](#), you log in to the HSM and change the PRECO user's password. When you change the password, the PRECO user becomes the primary crypto officer (PCO).

Crypto officer (CO | PCO)

A crypto officer (CO) can perform user management operations. For example, a CO can create and delete users and change user passwords. PCO is the designation for first CO you create, the primary CO. For more information about CO users, see the [HSM user permissions table \(p. 60\)](#). When you [activate a new cluster \(p. 31\)](#), the user changes from a [Precrypto Officer \(p. 59\)](#) (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 \(p. 60\)](#).

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 \(p. 60\)](#).

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 sorted by the type of HSM user or session that can perform the operation.

	Crypto Officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Get basic cluster info ¹	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Zeroize an HSM ²	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Change own password	✓ Yes	✓ Yes	✓ Yes	Not applicable
Change any user's password	✓ Yes	✗ No	✗ No	✗ No
Add, remove users	✓ Yes	✗ No	✗ No	✗ No
Get sync status ³	✓ Yes	✓ Yes	✓ Yes	✗ No

	Crypto Officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Extract, insert masked objects ⁴	✓ Yes	✓ Yes	✓ Yes	✗ No
Key management functions ⁵	✗ No	✓ Yes	✗ No	✗ No
Encrypt, decrypt	✗ No	✓ Yes	✗ No	✗ No
Sign, verify	✗ No	✓ Yes	✗ No	✗ No
Generate digests and HMACs	✗ No	✓ Yes	✗ No	✗ No

- [1] Basic cluster information includes the number of HSMs in the cluster and each HSM's IP address, model, serial number, device ID, firmware ID, etc.
- [2] When 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 \(p. 12\)](#).
- [3] The 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.
- [4] Masked 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.
- [5] Key management functions include creating, deleting, wrapping, unwrapping, and modifying the attributes of keys.

Using CloudHSM Management Utility (CMU) to manage users

This topic provides step-by-step instruction on managing hardware security module (HSM) users with CloudHSM Management Utility (CMU), a command line tool that comes with the Client SDK. For more information about CMU or HSM users, see [CloudHSM Management Utility \(p. 106\)](#) and [Understanding HSM users \(p. 59\)](#).

Sections

- [Understanding HSM user management with CMU \(p. 61\)](#)
- [Download CloudHSM Management Utility \(p. 62\)](#)
- [How to manage HSM users with CMU \(p. 64\)](#)

Understanding HSM user management with CMU

To manage HSM users, you must log in to the HSM with the user name and password of a [cryptographic officer \(p. 60\)](#) (CO). Only COs can manage users. The HSM contains a default CO named admin. You set the password for admin when you [activated the cluster \(p. 31\)](#).

To use CMU, you must use the configure tool to update the local configuration. CMU creates its own connection to the cluster and this connection is *not* cluster aware. To track cluster information, CMU

maintains a local configuration file. This means that *each time* you use CMU, you should first update the configuration file by running the [configure \(p. 238\)](#) command line tool with the `--cmu` parameter. If you are using Client SDK 3.2.1 or earlier, you must use a different parameter than `--cmu`. For more information, see [the section called “Using CMU with Client SDK 3.2.1 and earlier” \(p. 62\)](#).

The `--cmu` parameter requires you to add the IP address of an HSM in your cluster. If you have multiple HSMs, you can use any IP address. This ensures CMU can propagate any changes you make across the entire cluster. Remember that CMU uses its local file to track cluster information. If the cluster has changed since the last time you used CMU from a particular host, you must add those changes to the local configuration file stored on that host. Never add or remove a HSM while you’re using CMU.

To get an IP address for a HSM (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Clusters**.
4. To open the cluster detail page, in the cluster table, choose the cluster ID.
5. To get the IP address, on the HSMs tab, choose one of the IP addresses listed under **ENI IP address**.

To get an IP address for a HSM (AWS CLI)

- Get the IP address of an HSM by using the [describe-clusters](#) command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp`.

```
$ aws cloudhsmv2 describe-clusters

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

Using CMU with Client SDK 3.2.1 and earlier

With Client SDK 3.3.0, AWS CloudHSM added support for the `--cmu` parameter, which simplifies the process of updating the configuration file for CMU. If you’re using a version of CMU from Client SDK 3.2.1 or earlier, you must continue to use the `-a` and `-m` parameters to update the configuration file. For more information about these parameters, see [Configure Tool \(p. 238\)](#).

Download CloudHSM Management Utility

The latest version of CMU is available for HSM user management tasks whether you are using Client SDK 5 and Client SDK 3.

To download and install CMU

- Download and install CMU.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-mgmt-util-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el6.x86_64.rpm
```

Amazon Linux 2

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

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

CentOS 7.8+

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

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

CentOS 8.3+

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

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

RHEL 7.8+

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

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

RHEL 8.3+

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

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

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-mgmt-util_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

Windows Server 2012

1. Download [CloudHSM Management Utility](#).
2. Run the CMU installer (**AWSCloudHSMMManagementUtil-latest.msi**) with Windows administrative privilege.

Windows Server 2012 R2

1. Download [CloudHSM Management Utility](#).
2. Run the CMU installer (**AWSCloudHSMMManagementUtil-latest.msi**) with Windows administrative privilege.

Windows Server 2016

1. Download [CloudHSM Management Utility](#).
2. Run the CMU installer (**AWSCloudHSMMManagementUtil-latest.msi**) with Windows administrative privilege.

How to manage HSM users with CMU

This section includes basic commands to manage HSM users with CMU.

To create HSM users

Use **createUser** to create new users on the HSM. You must log in as a CO to create a user.

To create a new CO user

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm>loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use `createUser` to create a CO user named `example_officer` with a password of `password1`.

```
aws-cloudhsm>createUser CO example_officer password1
```

CMU prompts you about the create user operation.

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?
```

5. Type `y`.

To create a new CU user

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm>loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **createUser** to create a CU user named **example_user** with a password of **password1**.

```
aws-cloudhsm>createUser CU example_user password1
```

CMU prompts you about the create user operation.

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?
```

5. Type **y**.

For more information about **createUser**, see [createUser \(p. 115\)](#).

To list all HSM users on the cluster

Use **listUsers** command to list all the users on the cluster. You do not have to log in to run **listUsers** and all user types can list users.

To list all users on the cluster

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Use **listUsers** to list all the users on the cluster.

```
aws-cloudhsm>listUsers
```

CMU lists all the users on the cluster.

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

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	example_officer	NO
0	NO		
4	CU	example_user	NO
0	NO		

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

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	example_officer	NO
0	NO		
4	CU	example_user	NO
0	NO		

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

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	example_officer	NO
0	NO		
4	CU	example_user	NO
0	NO		

The PCO is the first CO created on each HSM. The PCO is known as the *primary* CO and this CO has the same permissions as any other CO.

For more information about **listUsers**, see [listUsers \(p. 134\)](#).

To change HSM user passwords

Use **changePswd** to change a password.

User types and passwords are case sensitive, but user names are not case sensitive.

CO, Crypto user (CU), and appliance user (AU) can change their own password. To change the password of another user, you must log in as a CO. You cannot change the password of a user who is currently logged in.

To change your own password

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM.

```
aws-cloudhsm>loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **changePswd** to change your own password.

```
aws-cloudhsm>changePswd CO example_officer <new password>
```

CMU prompts you about the change password operation.

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?
```

5. Type **y**.

CMU prompts you about the change password operation.

```
Changing password for example_officer(CO) on 3 nodes
```

To change the password of another user

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm>loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **changePswd** to change the password of another user.

```
aws-cloudhsm>changePswd CU example_user <new password>
```

CMU prompts you about the change password operation.

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?
```

5. Type **y**.

CMU prompts you about the change password operation.

```
Changing password for example_user(CU) on 3 nodes
```

For more information about **changePswd**, see [changePswd \(p. 111\)](#).

To delete HSM users

Use **deleteUser** to delete a user. You must log in as a CO to delete another user.

Tip

You can't delete crypto users (CU) that own keys.

To delete a user

1. Use the configure tool to update the CMU configuration.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

2. Start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm>loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **deleteUser** to delete a user.

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

CMU deletes the user.

```
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)
```

For more information about **deleteUser**, see [deleteUser \(p. 118\)](#).

Managing two-factor authentication (2FA) for crypto officers

For increased security, you can configure two-factor authentication (2FA) to help protect the cluster. You can only enable 2FA for crypto officers (CO).

Note

You cannot enable 2FA for crypto users (CU) or applications. Two-factor authentication (2FA) is only for CO users.

Topics

- [Understanding 2FA for HSM users \(p. 71\)](#)
- [Working with 2FA for HSM users \(p. 71\)](#)

Understanding 2FA for HSM users

When you log in to a cluster with a 2FA-enabled hardware service module (HSM) account, you provide `cloudhsm_mgmt_util` (CMU) with your password—the first factor, what you know—and CMU provides you with a token and prompts you to have the token signed. To provide the second factor—what you have—you sign the token with a private key from a key pair you've already created and associated with the HSM user. To access the cluster, you provide the signed token to CMU.

Quorum authentication and 2FA

The cluster uses the same key for quorum authentication and for 2FA. This means a user with 2FA enabled is effectively registered for M-of-N-access-control (MofN). To successfully use 2FA and quorum authentication for the same HSM user, consider the following points:

- If you are using quorum authentication for a user today, you should use the same key pair you created for the quorum user to enable 2FA for the user.
- If you add the 2FA requirement for a non-2FA user that is not a quorum authentication user, then you register that user as an MofN user with 2FA authentication.
- If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, you will also remove the registration of the quorum user as an MofN user.
- If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, but you *still want that user to participate in quorum authentication*, then you must register that user again as an MofN user.

For more information about quorum authentication, see [Managing quorum authentication \(p. 74\)](#).

Working with 2FA for HSM users

This section describes how to work with 2FA for HSM users, including creating 2FA HSM users, rotating keys, and logging in to the HSM as 2FA-enabled users. For more information about working with HSM users, see [???](#) (p. 59), [???](#) (p. 61), [???](#) (p. 115), [???](#) (p. 136), and [???](#) (p. 111).

Creating 2FA users

To enable 2FA for an HSM user, use a key that meets the following requirements.

2FA key pair requirements

You can create a new key pair or use an existing key that meets the following requirements.

- Key type: Asymmetric
- Key usage: Sign and Verify
- Key spec: RSA_2048
- Signing algorithm includes:
 - sha256WithRSAEncryption

Note

If you are using quorum authentication or plan to use quorum authentication, see [the section called "Quorum authentication and 2FA" \(p. 71\)](#).

You use CMU and the key pair to create a new CO user with 2FA enabled.

To create CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO.
2. Use **createUser** to create a CO user with 2FA enabled. Use the **-2fa** parameter and specify a location in the file system for the system to write the authdata file. This file will include a digest for each HSM in the cluster.

```
aws-cloudhsm>createUser CO example-user <password> -2fa /path/to/authdata
```

CMU prompts you to use the private key to sign the digests in the authdata file and return the signatures with the public key.

3. Use the private key to sign the digests in the authdata file, add the signatures and the public key to the JSON formatted authdata file, and then provide CMU with the location of the authdata file. For more information, see [the section called "Configuration reference" \(p. 73\)](#).

Logging in 2FA users

To log in as CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO with 2FA enabled.

Note

To log in as a CO user with 2FA, you must first enable 2FA for the CO user. If you specify the **-2fa** parameter for a non-2FA user, the system prompts you as if you were logging in as a 2FA user, but the system ignores the second factor. To enable 2FA for a CO user, see [the section called "Creating 2FA users" \(p. 71\)](#) or [the section called "Managing 2FA for HSM users" \(p. 72\)](#).

2. Use **loginHSM** to log in with 2FA. Specify that the 2FA user is a CO and provide the user name and password as normal, but use the **-2fa** parameter and include a location in the file system for the system to write the authdata file. This file will include a digest for each HSM in the cluster.

```
aws-cloudhsm>loginHSM CO example-user <password> -2fa /path/to/authdata
```

CMU prompts you to use the private key to sign the digests in the authdata file and return the signatures.

3. Use the private key to sign the digests in the authdata file, add the signatures to the JSON formatted authdata file and then provide CMU with the location of the authdata file. For more information, see [the section called "Configuration reference" \(p. 73\)](#).

Managing 2FA for HSM users

Use **change password** to change the password for a 2FA user, or to enable or disable 2FA, or to rotate the 2FA key. Each time you enable 2FA, you must provide a public key for 2FA logins.

Change password performs any of the following scenarios:

- Change the password for a 2FA user
- Change the password for a non-2FA user

- Add 2FA to a non-2FA user
- Remove 2FA from a 2FA user
- Rotate the key for a 2FA user

You can also combine tasks. For example, you can remove 2FA from a user and change the password at the same time, or you might rotate the 2FA key and change the user password.

To change passwords or rotate keys for CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO with 2FA enabled.
2. Use **changePswd** to change the password or rotate the key from CO users with 2FA enabled. Use the **-2fa** parameter and include a location in the file system for the system to write the authdata file. This file includes a digest for each HSM in the cluster.

```
aws-cloudhsm>changePswd CO example-user <new-password> -2fa /path/to/authdata
```

CMU prompts you to use the private key to sign the digests in the authdata file and return the signatures with the public key.

3. Use the private key to sign the digests in the authdata file, add the signatures and the public key to the JSON formatted authdata file and then provide CMU with the location of the authdata file. For more information, see [the section called "Configuration reference" \(p. 73\)](#).

Note

The cluster uses the same key for quorum authentication and 2FA. If you are using quorum authentication or plan to use quorum authentication, see [the section called "Quorum authentication and 2FA" \(p. 71\)](#).

To disable 2FA for CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO with 2FA enabled.
2. Use **changePswd** to remove 2FA from CO users with 2FA enabled.

```
aws-cloudhsm>changePswd CO example-user <new-password>
```

CMU prompts you to confirm the change password operation.

Note

If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, you will also remove the registration of the quorum user as an MoffN user. For more information about quorum users and 2FA, see [the section called "Quorum authentication and 2FA" \(p. 71\)](#).

3. Type **y**.

CMU confirms the change password operation.

Configuration reference

The following is an example of the 2FA properties in the authdata file for both the CMU-generated request and your responses.

```
{  
    "Version": "1.0",  
    "PublicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----",
```

```
"Data": [
  {
    "HsmId": "hsm-lgavqitns2a",
    "Digest": "k5O1p3f6foQRVQH7S8Rrjcau6h3TYqsSdr16A54+qG8=",
    "Signature": "KkdI ... rkrvJ6Q=="
  },
  {
    "HsmId": "hsm-lgavqitns2a",
    "Digest": "IyBcx4I5Vyx1jztwvXinCBQd9lDx8oQe7iRrWjBAi1w=",
    "Signature": "K1hxy ... Q261Q=="
  }
]
```

Data

Top-level node. Contains a subordinate node for each HSM in the cluster. Appears in requests and responses for all 2FA commands.

Digest

This is what you must sign to provide the second factor of authentication. CMU generated in requests for all 2FA commands.

HsmId

The ID of your HSM. Appears in requests and responses for all 2FA commands.

PublicKey

The public key portion of the key pair you generated inserted as PEM-formatted string. You enter this in responses for `createUser` and `changePswd`.

Signature

The Base 64 encoded signed digest. You enter this in responses for all 2FA commands.

Version

The version of the authentication data JSON formatted file. Appears in requests and responses for all 2FA commands.

Managing 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 \(COs\) \(p. 60\)](#) – Creating and deleting HSM users, and changing a different HSM user's password. For more information, see [Using quorum authentication for crypto officers \(p. 80\)](#).

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

Topics

- [Overview of quorum authentication \(p. 75\)](#)
- [Additional details about quorum authentication \(p. 75\)](#)
- [Using quorum authentication for crypto officers: first time setup \(p. 75\)](#)
- [Using quorum authentication for crypto officers \(p. 80\)](#)
- [Change the quorum minimum value for crypto officers \(p. 86\)](#)

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 \(p. 80\)](#).

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.
- The cluster uses the same key for quorum authentication and for two-factor authentication (2FA). For more information about using quorum authentication and 2FA, see [Quorum Authentication and 2FA \(p. 71\)](#).

Using quorum authentication for crypto officers: first time setup

The following topics describe the steps that you must complete to configure your hardware security module (HSM) so that [crypto officers \(COs\) \(p. 60\)](#) 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 \(p. 80\)](#).

Topics

- [Prerequisites \(p. 76\)](#)
- [Create and register a key for signing \(p. 76\)](#)
- [Set the quorum minimum value on the HSM \(p. 79\)](#)

Prerequisites

To understand this example, you should be familiar with the [cloudhsm_mgmt_util \(CMU\) command line tool \(p. 106\)](#). 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 \(p. 59\)](#).

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

  User Id      User Type      User Name      MofnPubKey
LoginFailureCnt
    1          PCO           admin          NO
    0          NO            app_user        NO
    2          AU            officer1       NO
    0          NO            officer2       NO
    3          CO            officer3       NO
    0          NO            officer4       NO
    4          CO            officer5       NO
    0          NO
    5          CO
    0          NO
    6          CO
    0          NO
    7          CO
    0          NO

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

  User Id      User Type      User Name      MofnPubKey
LoginFailureCnt
    1          PCO           admin          NO
    0          NO            app_user        NO
    2          AU            officer1       NO
    0          NO            officer2       NO
    3          CO            officer3       NO
    0          NO            officer4       NO
    4          CO            officer5       NO
    0          NO
    5          CO
    0          NO
    6          CO
    0          NO
    7          CO
    0          NO
```

Create and register a key for signing

To use quorum authentication, each CO must do *all* of the following steps:

Topics

- [Create an RSA key pair \(p. 77\)](#)
- [Create and sign a registration token \(p. 77\)](#)
- [Register the public key with the HSM \(p. 77\)](#)

Create an RSA key pair

There are many different ways to create and protect a key pair. The following examples show how to do it with OpenSSL.

Example – Create a private 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:
```

Next, generate the public key using the private key that you just created.

Example – Create a public key with OpenSSL

The following example demonstrates how to use OpenSSL to create a public key from the private key you just created.

```
$ openssl rsa -in officer1.key -outform PEM -pubout -out officer1.pub
Enter pass phrase for officer1.key:
writing RSA key
```

Create and sign a registration token

You create a token and sign it with the private key you just generated in the previous step.

Example – Create a token

The registration token is just a file with any random data that doesn't exceed the maximum size of 245 bytes. You sign the token with the private key to demonstrate that you have access to the private key. The following command uses echo to redirect a string to a file.

```
$ echo "token to be signed" > quorum_officer.token
```

Sign the token and save it to a signature file. You will need the signed token, the unsigned token, and the public key to register the CO as an MofN user with the HSM.

Example – Sign the token

Use OpenSSL and the private key to sign the registration token and create the signature file.

```
$ openssl dgst -sha256 \
-sign officer1.key \
-out officer1.token.sig officer1.token
```

Register the public key with the HSM

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 **loginHSM** command to log in to the HSM as a CO. For more information, see [???](#) (p. 64).
3. Use the **registerQuorumPubKey** (p. 138) command to register the public key. For more information, see the following example or use the **help registerQuorumPubKey** command.

Example – Register a public key with the HSM

The following example shows how to use the **registerQuorumPubKey** 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:

```
aws-cloudhsm> registerQuorumPubKey CO <officer1> <officer1.token> <officer1.token.sig>
<officer1.pub>

*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****


Do you want to continue(y/n)?y
registerQuorumPubKey success on server 0(10.0.2.14)
```

<officer1.token>

The path to a file that contains an unsigned registration token. Can have any random data of max file size of 245 bytes.

Required: Yes

<officer1.token.sig>

The path to a file that contains the SHA256_PKCS mechanism signed hash of the registration token.

Required: Yes

<officer1.pub>

The path to the file that contains the public key of an asymmetric RSA-2048 key pair. Use the private key to sign the registration token.

Required: Yes

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.

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt 1 0	2FA PCO NO	admin	NO

2	AU	app_user	NO
0	NO		
3	CO	officer1	YES
0	NO		
4	CO	officer2	YES
0	NO		
5	CO	officer3	YES
0	NO		
6	CO	officer4	YES
0	NO		
7	CO	officer5	YES
0	NO		
Users on server 1(10.0.1.4):			
Number of users found:7			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	officer1	YES
0	NO		
4	CO	officer2	YES
0	NO		
5	CO	officer3	YES
0	NO		
6	CO	officer4	YES
0	NO		
7	CO	officer5	YES
0	NO		

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 minimum value \(p. 86\)](#).

To set the quorum minimum value on the HSM

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

```
$ /opt/cloudfsm/bin/cloudfsm_mgmt_util /opt/cloudfsm/etc/cloudfsm_mgmt_util.cfg
```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see [???](#) (p. 64).
3. 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 \(p. 60\)](#) 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-cloudfsm>setMValue 3 2
```

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?y
Setting M Value(2) for 3 on 2 nodes
```

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.

Service Identifier	Service Name	Service Description	HSM Commands
3	USER_MGMT	HSM user management	<ul style="list-style-type: none"> • createUser • deleteUser • changePswd (applies only when changing the password of a different HSM user)
4	MISC_CO	Miscellaneous CO service	<ul style="list-style-type: none"> • setMValue

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 \(p. 80\)](#).

Using quorum authentication for crypto officers

A [crypto officer \(CO\) \(p. 60\)](#) 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?y
Creating User user1(CU) on 2 nodes
createUser failed: RET_MXN_AUTH_FAILED
creating user on server 0(10.0.2.14) failed

Retry/Ignore/Abort?(R/I/A):A
```

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

1. Get a *quorum token* (p. 81).
2. Get approvals (signatures) from other COs (p. 82).
3. Approve the token on the HSM (p. 82).
4. Perform the HSM user management operation (p. 84).

If you have not yet configured the HSM for quorum authentication for COs, do that now. For more information, see [First time setup \(p. 75\)](#).

Get a quorum token

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

To get a quorum token

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

```
$ /opt/cloudmgmt/bin/cloudmgmt_util /opt/cloudmgmt/etc/cloudmgmt_util.cfg
```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see [???](#) (p. 64).
3. 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 \(p. 79\)](#).

```
aws-cloudmgmt>getToken 3 officer1 officer1.token
getToken success on server 0(10.0.2.14)
Token:
Id:1
Service:3
```

```
Node:1
Key Handle:0
User:officer1
getToken success on server 1(10.0.1.4)
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
```

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 `loginHSM` command to log in to the HSM as a CO. For more information, see [???](#) (p. 64).
4. 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.

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.

```
aws-cloudhsm>listTokens

=====
Server 0(10.0.2.14)
=====
----- Token - 0 -----
Token:
Id:1
Service:3
Node:1
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
Current num of approvals : 2
Approver-0: officer1
```

```

Approver-1: officer2
Num of tokens = 1

=====
      Server 1(10.0.1.4)
=====
----- Token - 0 -----
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
Current num of approvals : 2
Approver-0: officer1
Approver-1: officer2
Num of tokens = 1

listTokens success

```

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 \(p. 115\)](#) 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 \(p. 59\)](#).

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.

Note

The MofN token is only valid as long as your current login session is open. If you log out of cloudfsm_mgmt_util or the network connection disconnects, the token is no longer valid. Similarly, an authorized token can only be used within cloudfsm_mgmt_util, it cannot be used to authenticate in a different application.

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

```

aws-cloudfsm>createUser CU user1 password
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****


Do you want to continue(y/n)?y
Creating User user1(CU) on 2 nodes

```

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

```
aws-cloudfsm>listUsers
```

```
Users on server 0(10.0.2.14):
Number of users found:8
```

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	officer1	YES
0	NO		
4	CO	officer2	YES
0	NO		
5	CO	officer3	YES
0	NO		
6	CO	officer4	YES
0	NO		
7	CO	officer5	YES
0	NO		
8	CU	user1	NO
0	NO		

```
Users on server 1(10.0.1.4):
```

```
Number of users found:8
```

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	officer1	YES
0	NO		
4	CO	officer2	YES
0	NO		
5	CO	officer3	YES
0	NO		
6	CO	officer4	YES
0	NO		
7	CO	officer5	YES
0	NO		
8	CU	user1	NO
0	NO		

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)

Retry/rollBack/Ignore?(R/B/I):I
```

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)
=====
Num of tokens = 0

listTokens success
```

Change the quorum minimum value for crypto officers

After you [set the quorum minimum value \(p. 79\)](#) so that [crypto officers \(COs\) \(p. 60\)](#) 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.

Service Identifier	Service Name	Service Description	HSM Commands
3	USER_MGMT	HSM user management	<ul style="list-style-type: none"> • createUser • deleteUser • changePswd (applies only when changing the password of a different HSM user)
4	MISC_CO	Miscellaneous CO service	<ul style="list-style-type: none"> • setMValue

To change the quorum minimum value for crypto officers

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 **loginHSM** command to log in to the HSM as a CO. For more information, see [???](#) (p. 64).
3. Use the **getMValue** command to get the quorum minimum value for service 3. For more information, see the following example.
4. Use the **getMValue** command to get the quorum minimum value for service 4. For more information, see the following example.
5. 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.

6. [Get a quorum token \(p. 81\)](#), taking care to specify service 4 as the service for which you can use the token.
7. [Get approvals \(signatures\) from other COs \(p. 82\)](#).
8. [Approve the token on the HSM \(p. 82\)](#).
9. 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
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
Do you want to continue(y/n)?y
Setting M Value(2) for 4 on 2 nodes
```

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]
```

Managing keys in AWS CloudHSM

In AWS CloudHSM, use any of the following to manage keys on the HSMs in your cluster:

- PKCS #11 library

- JCE provider
- CNG and KSP providers
- key_mgmt_util

Before you can manage keys, you must log in to the HSM with the user name and password of a crypto user (CU). Only a CU can create a key. The CU who creates a key owns and manages that key.

Topics

- [Key synchronization in AWS CloudHSM \(p. 88\)](#)
- [Using trusted keys to control key unwraps \(p. 93\)](#)
- [AES key wrapping in AWS CloudHSM \(p. 95\)](#)
- [Using the command line to manage keys \(p. 97\)](#)

Key synchronization in AWS CloudHSM

This topic describes key synchronization settings in AWS CloudHSM, common issues customers face working with keys on a cluster, and strategies for making keys more durable.

Topics

- [Concepts \(p. 88\)](#)
- [Understanding key synchronization \(p. 89\)](#)
- [Working with client key durability settings \(p. 90\)](#)
- [Synchronizing keys across cloned clusters \(p. 93\)](#)

Concepts

Token keys

Persistent keys that you create during key generate, import or unwrap operations. AWS CloudHSM synchronizes token keys across a cluster.

Session keys

Ephemeral keys that exist only on one hardware security module (HSM) in the cluster. AWS CloudHSM does *not* synchronize session keys across a cluster.

Client-side key synchronization

A client-side process that clones token keys you create during key generate, import or unwrap operations. You can make token keys more durable by running a cluster with a minimum of two HSMs.

Server-side key synchronization

Periodically clones keys to every HSM in the cluster. Requires no management.

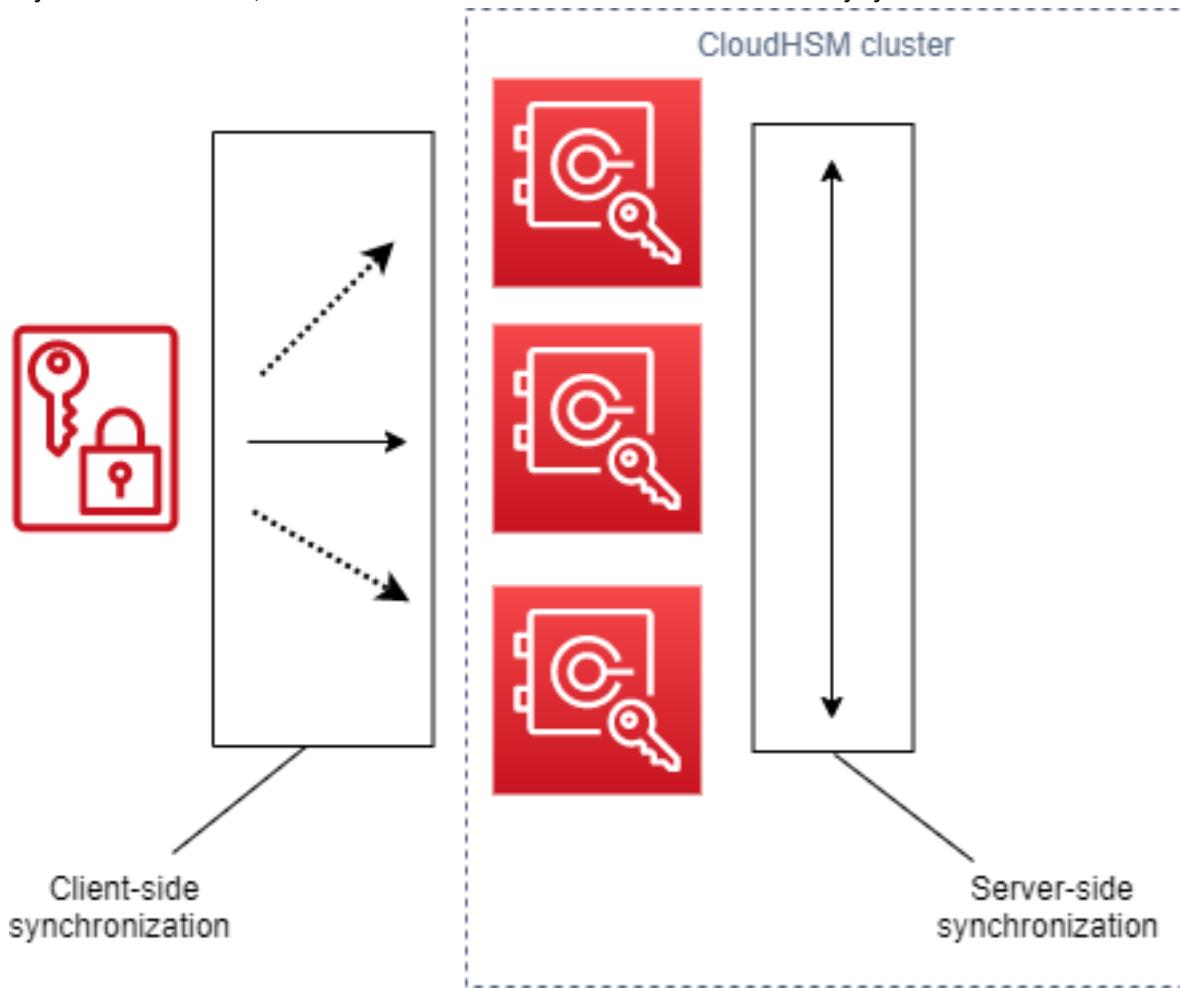
Client key durability settings

Settings you configure on the client that impact key durability. Works differently in Client SDK 5 and Client SDK 3.

- In Client SDK 5, use this setting to run a single HSM cluster.
- In Client SDK 3, use this setting to specify the number of HSMs required for key creation operations to succeed.

Understanding key synchronization

AWS CloudHSM uses key synchronization to clone token keys across all the HSMs in a cluster. You create token keys as persistent keys during key generation, import, or unwrap operations. To distribute these keys across the cluster, CloudHSM offers both client-side and server-side key synchronization.



The goal with key synchronization—both server side and client side—is to distribute new keys across the cluster as quickly as possible after you create them. This is important because the subsequent calls you make to use new keys can get routed to any available HSM in the cluster. If the call you make routes to a HSM without the key, then the call fails. You can mitigate these type failures by specifying that your applications retry subsequent calls made after key creation operations. The time required to synchronize can vary, depending on the workload of your cluster and other intangibles. Use CloudWatch metrics to determine the timing your application should employ in this type situation. For more information, see [CloudWatch Metrics \(p. 500\)](#).

The challenge with key synchronization in a cloud environment is key durability. You create keys on a single HSM and often begin using those keys immediately. If the HSM on which you create keys should fail before the keys have been cloned to another HSM in the cluster, you lose the keys *and* access to anything encrypted by the keys. To mitigate this risk, we offer *client-side synchronization*. Client side synchronization is a client-side process that clones the keys you create during key generate, import, or unwrap operations. Cloning keys as you create them makes keys more durable. Of course, you can't clone keys in a cluster with a single HSM. To make keys more durable, we also recommend you configure your cluster to use a minimum of two HSMs. With client-side synchronization and a cluster with two HSMs, you can meet the challenge of key durability in a cloud environment.

Working with client key durability settings

Key synchronization is mostly an automatic process, but you can manage client-side key durability settings. Client-side key durability settings works differently in Client SDK 5 and Client SDK 3.

- In Client SDK 5, we introduce the concept of *key availability quorums* which requires you to run clusters with a minimum of two HSMs. You can use client-side key durability settings to opt out of the two HSM requirement. For more information about quorums, see [the section called "Client SDK 5 concepts" \(p. 90\)](#).
- In Client SDK 3, you use client-side key durability settings to specify the number of HSMs on which key creation must succeed for the overall operation to be deemed a success.

Client SDK 5 client key durability settings

Client SDK 5 concepts

Key Availability Quorum

AWS CloudHSM specifies the number of HSMs in a cluster on which keys must exist before your application can use the key. Requires clusters with a minimum of two HSMs.

In Client SDK 5, key synchronization is a fully automatic process. With key availability quorum, newly created keys must exist on two HSMs in the cluster before your application can use the key. To use key availability quorum, your cluster must have a minimum of two HSMs.

If your cluster configuration doesn't meet the key durability requirements, any attempt to create or use a token key will fail with the following error message in the logs:

Key <key handle> does not meet the availability requirements - The key must be available on at least 2 HSMs before being used.

You can use client configuration settings to opt out of key availability quorum. You might want to opt out to run a cluster with a single HSM, for example.

Managing client key durability settings

To manage client key durability settings, you must use the configure tool for Client SDK 5.

PKCS #11 library

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe --disable-key-availability-check
```

OpenSSL Dynamic Engine

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

JCE provider

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
C:\Program Files\Amazon\CloudHSM\configure-jce.exe --disable-key-availability-check
```

Client SDK 3 client key durability settings

In Client SDK 3, key synchronization is mostly an automatic process, but you can use the client key durability settings to make keys more durable. You specify the number of HSMs on which key creation must succeed for the overall operation to be deemed a success. Client-side synchronization always makes a best-effort attempt to clone keys to every HSM in the cluster no matter what setting you choose. Your setting enforces key creation on the number of HSMs you specify. If you specify a value and the system cannot replicate the key to that number of HSMs, then the system automatically cleans up any unwanted key material and you can try again.

Important

If you don't set client key durability settings (or if you use the default value of 1), your keys are vulnerable to loss. If your current HSM should fail before the server-side service has cloned that key to another HSM, you lose the key material.

To maximize key durability, consider specifying at least two HSMs for client-side synchronization. Remember that no matter how many HSMs you specify, the workload on your cluster remains the same. Client-side synchronization always makes a best-effort attempt to clone keys to every HSM in the cluster.

Recommendations

- **Minimum:** Two HSMs per cluster
- **Maximum:** One fewer than the total number of HSMs in your cluster

If client-side synchronization fails, the client service cleans up any unwanted keys that may have been created and are now unwanted. This clean up is a best-effort response that may not always work. If cleanup fails, you may have to delete unwanted key material. For more information, see [Key Synchronization Failures \(p. 534\)](#).

Setting up the configuration file for client key durability

To specify client key durability settings, you must edit `cloudhsm_client.cfg`.

To edit the client configuration file

1. Open `cloudhsm_client.cfg`.

Linux:

```
/opt/cloudhsm/etc/cloudhsm_client.cfg
```

Windows:

```
C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

2. In the `client` node of the file, add `create_object_minimum_nodes` and specify a value for the minimum number of HSMs on which AWS CloudHSM must successfully create keys for key creation operations to succeed.

```
"create_object_minimum_nodes" : 2
```

Note

The `key_mgmt_util` (KMU) command-line tool has an additional setting for client key durability. For more information, see [the section called "KMU and client-side synchronization" \(p. 92\)](#)

Configuration reference

These are the client-side synchronization properties, shown in an excerpt of the `cloudhsm_client.cfg`:

```
{
    "client": {
        "create_object_minimum_nodes" : 2,
        ...
    },
    ...
}
```

`create_object_minimum_nodes`

Specifies the minimum number of HSMs required to deem key generation, key import, or key unwrap operations a success. If set, the default is 1. This means that for every key create operation, the client-side service attempts to create keys on every HSM in the cluster, but to return a success, only needs to create a *single key* on one HSM in the cluster.

KMU and client-side synchronization

If you create keys with the `key_mgmt_util` (KMU) command-line tool, you use an optional command line parameter (`-min_srv`) to *limit* the number of HSMs on which to clone keys. If you specify the command-

line parameter *and* a value in the configuration file, AWS CloudHSM honors the LARGER of the two values.

For more information, see the following topics:

- [genDSAKeyPair \(p. 176\)](#)
- [genECCKeyPair \(p. 180\)](#)
- [genRSAKeyPair \(p. 184\)](#)
- [genSymKey \(p. 189\)](#)
- [importPrivateKey \(p. 204\)](#)
- [importPubKey \(p. 207\)](#)
- [imSymKey \(p. 209\)](#)
- [insertMaskedObject \(p. 215\)](#)
- [unWrapKey \(p. 226\)](#)

Synchronizing keys across cloned clusters

Client-side and server-side synchronization are only for synchronizing keys within the *same* cluster. If you copy a backup of a cluster to another region, you can use the `syncKey` command of the `cloudhsm_mgmt_util` (CMU) for synchronizing keys between clusters. You might use cloned clusters for cross-region redundancy or to simplify your disaster recovery process. For more information, see [syncKey \(p. 148\)](#).

Using trusted keys to control key unwraps

AWS CloudHSM supports trusted key wrapping to protect data keys from insider threats. This topic describes how to use the PKCS #11 library attributes to create trusted keys to secure data, but you could also use attributes from the Java Cryptographic Extension (JCE) provider in the same way.

Topics

- [Understanding trusted keys \(p. 93\)](#)
- [How to use trusted keys to control unwraps \(p. 94\)](#)
- [Code sample \(p. 95\)](#)

Understanding trusted keys

You specify actions permitted on a key with attributes that you define when you create or unwrap a key. A *trusted key* is a key that you identify as trusted with an attribute (`CKA_TRUSTED`). You can only make this designation as the cryptographic officer (CO). On the trusted key, you define another attribute (`CKA_UNWRAP_TEMPLATE`) that specifies a *set* of attributes that data keys unwrapped by the trusted key must contain for the unwrap operation to succeed. In this way, you ensure that your unwrapped data keys contain attributes that allow only the use you intend for those keys. Meanwhile, you use another attribute (`CKA_WRAP_WITH_TRUSTED`) to identify all of the data keys you want to wrap with trusted keys. In this way, you restrict data keys so that applications can only use trusted keys to unwrap them. Once you set this attribute on the data keys, the attribute becomes read only and you cannot change it. With these attributes in place, applications can only unwrap your data keys with the keys you trust, and these unwraps must always result in data keys that have attributes you intend to limit the use of those data keys.

Trusted key attributes

In the PKCS #11 library, use the following attributes to specify trusted keys to control key unwraps:

- **CKA_WRAP_WITH_TRUSTED:** Apply this attribute to an exportable data key to specify that you can only wrap this key with keys marked as **CKA_TRUSTED**. Once you set **CKA_WRAP_WITH_TRUSTED** to true, the attribute becomes read-only and you cannot change or remove the attribute.
- **CKA_TRUSTED:** Apply this attribute to the wrapping key (in addition to **CKA_UNWRAP_TEMPLATE**) to specify that a CO has done the necessary diligence and trusts this key. Only a CO can set **CKA_TRUSTED**. The CU owns the key, but only a CO can set **CKA_TRUSTED**.
- **CKA_UNWRAP_TEMPLATE:** Apply this attribute to the wrapping key (in addition to **CKA_TRUSTED**) to specify which attribute names and values the service must automatically apply to data keys that the service unwraps. When an application submits a key for unwrapping, the application can also provide its own unwrap template. If you specify an unwrap template and the application provides its own unwrap template, the HSM uses the union of both templates to apply attribute names and values to the key. 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, then the unwrap request fails.

For more information about PKCS #11 library attributes, see [Supported attributes \(p. 336\)](#).

How to use trusted keys to control unwraps

Step 1: Generate the trusted key

To set up a trusted key, you establish a 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 library application. You can't use advanced attributes with command-line tools.

Step 2: Mark the key as trusted

1. To mark the key as trusted, you must use a cryptographic officer (CO) account with CloudHSM Management Utility (CMU).

```
loginHSM CO <user-name> <password>
```

2. Use **setAttribute** with **OBJ_ATTR_TRUSTED** (value 134) set to TRUE.

```
setAttribute HH 134 1
```

For more information about **setAttribute**, see [setAttribute \(p. 142\)](#)

Step 3: Share the trusted key with the application

To share the trusted key with the application, you must log in with the CU account you created in step 1. Use the [shareKey \(p. 145\)](#) command to share the trusted key you created in step 2 with any CU accounts used by 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 no one can use CU accounts to modify the **CKA_TRUSTED** attribute on the key.

Step 4: Wrap all the data keys

To wrap all the data keys, you use the CU account to import or generate all data keys and wrap them with the trusted wrapping key. You can externally store the wrapped keys, as necessary. Since you can only unwrap data keys with the original wrapping keys, you must apply the appropriate template 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.

Code sample

This example uses an unwrap template to generate a key.

```
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_UNWRAP, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_DECRYPT, &trueValue, sizeof(trueValue) },
    CK_ATTRIBUTE { CKA_UNWRAP_TEMPLATE, unwrapTemplate.data(), unwrapTemplate.size() *
        sizeof(CK_ATTRIBUTE) }
};

func->C_GenerateKeyPair(
    session,
    &rsaMechanism,
    pubAttributes.data(),
    pubAttributes.size(),
    priAttributes.data(),
    priAttributes.size(),
    &pubKey,
    &priKey
);
```

AES key wrapping in AWS CloudHSM

This topic describes the options for AES key wrapping in AWS CloudHSM. AES key wrapping uses an AES key (the wrapping key) to wrap another key of any type (the target key). You use key wrapping to protect stored keys or transmit keys over insecure networks.

Topics

- [Supported algorithms \(p. 95\)](#)
- [Using AES key wrap in AWS CloudHSM \(p. 97\)](#)

Supported algorithms

AWS CloudHSM offers three options for AES key wrapping, each based on how the target key is padded before being wrapped. Padding is done automatically, in accordance with the algorithm you use, when

you call key wrap. The following table lists the supported algorithms and associated details to help you choose an appropriate wrapping mechanism for your application.

AES Key Wrap Algorithm	Specification	Supported Target Key Types	Padding Scheme	AWS CloudHSM Client Availability
AES Key Wrap with Zero Padding	RFC 5649 and SP 800 - 38F	All	Adds zeros after key bits, if necessary, to block align	SDK 3.1 and later
AES Key Wrap with No Padding	RFC 3394 and SP 800 - 38F	Block-aligned keys such as AES and 3DES	None	SDK 3.1 and later
AES Key Wrap with PKCS #5 Padding	None ¹	All	At least 8 bytes are added as per PKCS #5 padding scheme to block align	All

To learn how to use the AES key wrap algorithms from the preceding table in your application, see [Using AES Key Wrap in AWS CloudHSM. \(p. 97\)](#)

Understanding initialization vectors in AES key wrap

Prior to wrapping, CloudHSM appends an initialization vector (IV) to the target key for data integrity. Each key wrap algorithm has specific restrictions on what type of IV is allowed. To set the IV in AWS CloudHSM, you have two options:

- Implicit: set the IV to NULL and CloudHSM uses the default value for that algorithm for wrap and unwrap operations (recommended)
- Explicit: set the IV by passing the default IV value to the key wrap function

Important

You must understand what IV you are using in your application. To unwrap the key, you must provide the same IV that you used to wrap the key. If you use an implicit IV to wrap, then use an implicit IV to unwrap. With an implicit IV, CloudHSM will use the default value to unwrap.

The following table describes permitted values for IVs, which the wrapping algorithm specifies.

AES Key Wrap Algorithm	Implicit IV	Explicit IV
AES Key Wrap with Zero Padding	Required Default value: (IV calculated internally based on specification)	Not allowed
AES Key Wrap with No Padding	Allowed (recommended) Default value: 0xA6A6A6A6A6A6A6A6	Allowed Only this value accepted: 0xA6A6A6A6A6A6A6A6
AES Key Wrap with PKCS #5 Padding	Allowed (recommended)	Allowed

AES Key Wrap Algorithm	Implicit IV	Explicit IV
	Default value: 0xA6A6A6A6A6A6A6A6	Only this value accepted: 0xA6A6A6A6A6A6A6A6

Using AES key wrap in AWS CloudHSM

You wrap and unwrap keys as follows:

- In the [PKCS #11 library \(p. 324\)](#), select the appropriate mechanism for the `C_WrapKey` and `C_UnWrapKey` functions as shown in the following table.
- In the [JCE provider \(p. 352\)](#), select the appropriate algorithm, mode and padding combination, implementing cipher methods `Cipher.WRAP_MODE` and `Cipher.UNWRAP_MODE` as shown in the following table.
- In [key_mgmt_util \(KMU\) \(p. 152\)](#), use commands `wrapKey` and `unWrapKey` with appropriate `m` values as shown in the following table.

AES Key Wrap Algorithm	PKCS #11 Mechanism	Java Method	KMU Argument
AES Key Wrap with Zero Padding	• CKM_AES_KEY_WRAP_PAESWrap/ECB/ ZeroPadding	AESWrap/ECB/ ZeroPadding	$m = 6$
AES Key Wrap with No Padding	• CKM_CLOUDHSM_AES_KEY_WRAP_NOREAD (Vendor Defined Mechanism)	AESWrap/ECB/ NoPadding	$m = 5$
AES Key Wrap with PKCS #5 Padding	• CKM_AES_KEY_WRAP ¹ • CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD (Vendor Defined Mechanism)	AESWrap/ECB/ PKCS5Padding	$m = 4$

¹The CKM_AES_KEY_WRAP mechanism is not compliant with the PKCS #11 2.40 specification. For more information, see the first known issue under [Known Issues for all HSM instances \(p. 512\)](#).

Using the command line to manage keys

You can manage keys on the HSMs in your AWS CloudHSM cluster using the `key_mgmt_util` 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. For more information, see [Getting Started with key_mgmt_util \(p. 152\)](#).

Contents

- [Generate keys \(p. 98\)](#)
- [Import keys \(p. 99\)](#)
- [Export keys \(p. 100\)](#)
- [Delete keys \(p. 101\)](#)
- [Share and unshare keys \(p. 101\)](#)

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 \(p. 189\)](#) 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.

```
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 \(p. 184\)](#) 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 \(p. 180\)](#) 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 \(p. 226\)](#) command.

To import a secret key

1. Use the [genSymKey \(p. 189\)](#) 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

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 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 \(p. 166\)](#) 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 \(p. 232\)](#) command.

Export secret keys

Complete the following steps to export a secret key.

To export a secret key

1. Use the [genSymKey \(p. 189\)](#) 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 \(p. 166\)](#) 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"
```

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 \(p. 232\)](#) command.

- 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

PEM formatted private key is written to rsa2048.key.exp
```

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 \(p. 160\)](#) 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 \(p. 189\)](#) or [genRSAKeyPair \(p. 184\)](#) commands. To share existing keys with a different HSM user, use the [cloudhsm_mgmt_util \(p. 106\)](#) command line tool. This is different from most of the tasks documented in this section, which use the [key_mgmt_util \(p. 152\)](#) 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****  
  
Do you want to continue(y/n)?y
shareKey success on server 0(10.0.2.9)
shareKey success on server 1(10.0.3.11)
shareKey success on server 2(10.0.1.12)
```

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

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

Managing cloned clusters

Use CloudHSM Management Utility (CMU) to synchronize a cluster in a remote region, *if the cluster in that region was originally created from the backup of a cluster in another region*. Let's say you copied a cluster to another region (destination) and then later you want to synchronize changes from the original cluster (source). In scenarios like this, you use CMU to synchronize the clusters. You do this by creating a new CMU configuration file, specifying hardware security modules (HSM) from both clusters in the new file, and then using CMU to connect to the cluster with that file.

To use CMU across cloned clusters

1. Create a copy of your current configuration file and change the name of the copy to something else.

For example, use the following file locations to locate and create a copy of your current configuration file, then change the name of the copy from `cloudhsm_mgmt_config.cfg` to `syncConfig.cfg`.

- Linux: `/opt/cloudhsm/etc/cloudhsm_mgmt_config.cfg`
- Windows: `C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_config.cfg`

2. In the renamed copy, add the Elastic Network Interface (ENI) IP of the destination HSM (the HSM in the foreign region that needs to be synced). We recommend that you add the destination HSM *below* the source HSM.

```
{  
...  
}
```

```
"servers": [
    {
        ...
        "hostname": "<ENI Source IP>",
        ...
    },
    {
        ...
        "hostname": "<ENI Destination IP>",
        ...
    }
]
```

For more information about how to get the IP address, see [the section called “Get an IP address for an HSM” \(p. 103\)](#).

3. Initialize CMU with the new configuration file:

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon
\CloudHSM\data\userSync.cfg
```

4. Check the status messages returned to ensure that the CMU is connected to all desired HSMs and determine which of the returned ENI IPs corresponds to each cluster. Use syncUser and syncKey to manually synchronize users and keys. For more information, see [syncUser \(p. 149\)](#) and [syncKey \(p. 148\)](#).

Get an IP address for an HSM

Use this section to obtain an IP address for an HSM.

To get an IP address for a HSM (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Clusters**.
4. To open the cluster detail page, in the cluster table, choose the cluster ID.
5. To get the IP address, on the HSMs tab, choose one of the IP addresses listed under **ENI IP address**.

To get an IP address for a HSM (AWS CLI)

- Get the IP address of an HSM by using the **describe-clusters** command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp`.

```
$ aws cloudhsmv2 describe-clusters

{
    "Clusters": [
        { ... }
```

```
"HsmS": [  
    {  
        ...  
        "EniIp": "10.0.0.9",  
        ...  
    },  
    {  
        ...  
        "EniIp": "10.0.1.6",  
        ...  
    }  
]
```

Related topics

- [syncUser \(p. 149\)](#)
- [syncKey \(p. 148\)](#)
- [Copying Backups Across Regions \(p. 54\)](#)

AWS CloudHSM command line tools

This topic describes the command line tools available for managing and using AWS CloudHSM.

Topics

- [Understanding command line tools \(p. 105\)](#)
- [CloudHSM Management Utility \(CMU\) \(p. 106\)](#)
- [Key Management Utility \(KMU\) \(p. 152\)](#)
- [Configure tool \(p. 238\)](#)

Understanding command line tools

In addition to the AWS command-line interface (AWS CLI) that you use for managing your AWS resources, AWS CloudHSM offers command-line tools for managing HSM users and creating and managing keys on the HSM. In CloudHSM, you use the familiar AWS CLI to manage your cluster, and the CloudHSM command line tools to manage your HSM.

These are the various command-line tools:

Manage HSM and Clusters

These tools get, create, delete, and tag AWS CloudHSM clusters and HSMs:

- [CloudHSMv2 commands in 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 HSM Users

This tool creates and deletes HSM users, including implementing quorum authentication of user management tasks:

- [cloudhsm_mgmt_util \(p. 106\)](#). This tool is included in the AWS CloudHSM client software.

Manage Keys on the HSM

This tool creates, deletes, imports, and exports symmetric keys and asymmetric key pairs:

- [key_mgmt_util \(p. 152\)](#). This tool is included in the AWS CloudHSM client software.

Helper Tools

These tools help you to use the tools and software libraries.

- [configure \(p. 238\)](#) updates your CloudHSM client configuration files. This enables the AWS CloudHSM to synchronize the HSMs in a cluster.

- [pkpspeed \(p. 526\)](#) measures the performance of your HSM hardware independent of software libraries.

CloudHSM Management Utility (CMU)

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 \(p. 152\)](#).

For a quick start, see [Managing cloned clusters \(p. 106\)](#). For detailed information about the `cloudhsm_mgmt_util` commands and examples of using the commands, see [cloudhsm_mgmt_util command reference \(p. 109\)](#).

Topics

- [Getting started with CloudHSM Management Utility \(CMU\) \(p. 106\)](#)
- [cloudhsm_mgmt_util command reference \(p. 109\)](#)
- [AWS CloudHSM management utility platform support \(p. 151\)](#)

Getting started with CloudHSM Management Utility (CMU)

CloudHSM Management Utility (CMU) enables you to manage hardware security module (HSM) users. Use this topic to get started with basic HSM user management tasks, such as creating users, listing users, and connecting CMU to the cluster.

1. To use CMU, you must first use the configure tool to update the local CMU configuration with the `--cmu` parameter and an IP address from one of the HSMs in your cluster. Do this each time you use CMU to ensure you're managing HSM users on every HSM in the cluster.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

For more information about the CMU connection to the cluster, the configure tool, and obtaining an IP address for your cluster, see [??? \(p. 61\)](#).

2. Use the following command to start CMU.

Linux

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

Windows

```
C:\Program Files\Amazon\CloudHSM>cloudhsm_mgmt_util.exe C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

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.

3. Use the **loginHSM** command to log in to the cluster. Any type user can use this command to log in to the cluster.

The command in the following example logs in *admin*, which is the default [crypto officer \(CO\)](#) (p. 59). You set this user's password when you [activated the cluster](#) (p. 31). You can use the –hpswd parameter to hide your password.

```
aws-cloudhsm>loginHSM CO admin -hpswd
```

The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that you have connected to all the HSMs on the cluster.

```
Enter password:  
  
loginHSM success on server 0(10.0.2.9)  
loginHSM success on server 1(10.0.3.11)  
loginHSM success on server 2(10.0.1.12)
```

4. Use **listUsers** to list all the users on the cluster.

```
aws-cloudhsm>listUsers
```

CMU lists all the users on the cluster.

```
Users on server 0(10.0.2.9):  
Number of users found:2  
  
User Id          User Type        User Name          MofnPubKey  
LoginFailureCnt    2FA            admin             NO  
                1              CO  
                0              NO  
                2              AU  
                0              NO  
                0              NO
```

Users on server 1(10.0.3.11):				
Number of users found:2				
User Id	LoginFailureCnt	User Type	User Name	MofnPubKey
1	0	CO	admin	NO
2	0	AU	app_user	NO

Users on server 2(10.0.1.12):				
Number of users found:2				
User Id	LoginFailureCnt	User Type	User Name	MofnPubKey
1	0	CO	admin	NO
2	0	AU	app_user	NO

5. Use **createUser** to create a CU user named **example_user** with a password of **password1**.

You use CU users in your applications to perform cryptographic and key management operations. You can create CU users because in step 3 you logged in as a CO user. Only CO users can perform user management tasks with CMU, such as creating and deleting users and changing the passwords of other users.

```
aws-cloudhsm>createUser CU example_user password1
```

CMU prompts you about the create user operation.

```
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
```

```
Do you want to continue(y/n)?
```

6. To create the CU user **example_user**, type **y**.
7. Use **listUsers** to list all the users on the cluster.

```
aws-cloudhsm>listUsers
```

CMU lists all the users on the cluster, including the new CU user you just created.

Users on server 0(10.0.2.9):				
Number of users found:3				
User Id	LoginFailureCnt	User Type	User Name	MofnPubKey
1	0	CO	admin	NO
2	0	AU	app_user	NO
3	0	CU	example_user	NO

Users on server 1(10.0.3.11):				
Number of users found:3				

User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	CO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CU	example_user	NO
0	NO		
Users on server 2(10.0.1.12):			
Number of users found:3			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	CO	admin	NO
0	NO		
2	AU	app_user	NO
0	NO		
3	CU	example_user	NO
0	NO		

8. Use the **logoutHSM** command to log out of the HSMs.

```
aws-cloudhsm>logoutHSM

logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
logoutHSM success on server 2(10.0.1.12)
```

9. Use the **quit** command to stop cloudhsm_mgmt_util.

```
aws-cloudhsm>quit

disconnecting from servers, please wait...
```

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 \(p. 152\)](#) command line tool.

For a quick start, see [Managing cloned clusters \(p. 106\)](#).

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.

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 `cloudfsm_mgmt_util` commands.

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

```
aws-cloudfsm> listUsers
```

To end your `cloudfsm_mgmt_util` session, run the following command:

```
aws-cloudfsm> quit
```

For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

The following topics describe commands in `cloudfsm_mgmt_util`.

Note

Some commands in `key_mgmt_util` and `cloudfsm_mgmt_util` have the same names. However, the commands typically have different syntax, different output, and slightly different functionality.

Command	Description	User Type
changePswd (p. 111)	Changes the passwords of users on the HSMs. Any user can change their own password. COs can change anyone's password.	CO
createUser (p. 115)	Creates users of all types on the HSMs.	CO
deleteUser (p. 118)	Deletes users of all types from the HSMs.	CO
findAllKeys (p. 120)	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.	CO, AU
getAttribute (p. 123)	Gets an attribute value for an AWS CloudHSM key and writes it to a file or stdout (standard output).	CU
getCert (p. 126)	Gets the certificate of a particular HSM and saves it in a desired certificate format.	All.
getHSMInfo (p. 127)	Gets information about the hardware on which an HSM is running.	All. Login is not required.
getKeyInfo (p. 128)	Gets owners, shared users, and the quorum authentication status of a key.	All. Login is not required.

Command	Description	User Type
info (p. 132)	Gets information about an HSM, including the IP address, hostname, port, and current user.	All. Login is not required.
listUsers (p. 134)	Gets the users in each of the HSMs, their user type and ID, and other attributes.	All. Login is not required.
loginHSM and logoutHSM (p. 136)	Log in and log out of an HSM.	All.
quit (p. 145)	Quits cloudhsm_mgmt_util.	All. Login is not required.
server (p. 140)	Enters and exits server mode on an HSM.	All.
registerQuorumPubKey (p. 138)	Associates an HSM user with an asymmetric RSA-2048 key pair.	CO
setAttribute (p. 142)	Changes the values of the label, encrypt, decrypt, wrap, and unwrap attributes of an existing key.	CU
shareKey (p. 145)	Shares an existing key with other users.	CU
syncKey (p. 148)	Syncs a key across cloned AWS CloudHSM clusters.	CU, CO
syncUser (p. 149)	Syncs a user across cloned AWS CloudHSM clusters.	CO

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 or crypto user (CU). 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.

To troubleshoot changePswd

Before you run any CMU command, you must start CMU 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 for CMU. 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

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To enable two-factor authentication (2FA) for a CO user, use the `-2fa` parameter and include a file path. For more information, see [the section called "Arguments" \(p. 114\)](#).

```
changePswd <user-type> <user-name> <password | -hpswd> [-2fa </path/to/authdata>]
```

Examples

The following examples show how to use `changePassword` 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 \(p. 132\)](#) 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****


Do you want to continue(y/n)?y
Changing password for bob(CU) on 2 nodes
```

Example : Change the password of another user

You must be a CO or PCO to change the password of another CO, or CU on the HSMs. Before you change the password for another user, use the [info \(p. 132\)](#) 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-ogi3sywxbqx Logged in
as 'alice(CO)'
```

Alice wants to reset the password of another user, John. Before she changes the password, she uses the [listUsers \(p. 134\)](#) 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

User Id      User Type      User Name      MofnPubKey      LoginFailureCnt
2FA
1            PCO           admin          YES             0
NO
2            AU            jane           NO              0
NO
3            CU            bob            NO              0
NO
4            CU            alice          NO              0
NO
5            CO            john          NO              0
NO

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

User Id      User Type      User Name      MofnPubKey      LoginFailureCnt
2FA
1            PCO           admin          YES             0
NO
2            AU            jane           NO              0
NO
3            CU            bob            NO              0
NO
4            CO            alice          NO              0
NO
5            CO            john          NO              0
NO
```

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

```
aws-cloudhsm>changePswd CO john newPassword
*****
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****

Do you want to continue(y/n)?y
Changing password for john(CO) on 2 nodes
```

Arguments

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To enable 2FA for a CO user, use the `-2fa` parameter and include a file path. For more information about working with 2FA, see [Managing 2FA \(p. 70\)](#)

```
changePswd <user-type> <user-name> <password | -hpswd> [-2fa </path/to/authdata>]
```

<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, PCO, and PRECO.

To get the user type, use [listUsers \(p. 134\)](#). For detailed information about the user types on an HSM, see [Understanding HSM users \(p. 59\)](#).

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 | -hpswd >

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. To hide your password, use the `-hpswd` parameter in place of the password and follow the prompts.

Required: Yes

[-2fa </path/to/authdata>]

Specifies enabling 2FA for this CO user. To get the data necessary for setting up 2FA, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about working with 2FA, see [Managing 2FA \(p. 70\)](#).

Required: No

Related topics

- [info \(p. 132\)](#)

- [listUsers \(p. 134\)](#)
- [createUser \(p. 115\)](#)
- [deleteUser \(p. 118\)](#)

createUser

The **createUser** command in `cloudfsm_mgmt_util` creates a user on the HSMs. Only crypto officers (COs) and PCOs can run this command. When the command succeeds, it creates the user in all HSMs in the cluster.

To troubleshoot createUser

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 \(p. 149\)](#) or [createUser \(p. 115\)](#) command only on the HSMs that are missing that user. To prevent configuration errors, run the [configure \(p. 238\)](#) tool with the `-m` option.

Before you run any CMU command, you must start CMU 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 for CMU. 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

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To create a CO user with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information, see [the section called "Arguments" \(p. 117\)](#).

```
createUser <user-type> <user-name> <password | -hpswd> [-2fa </path/to/authdata>]
```

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 \(p. 136\)](#) to log in to the HSM as a crypto officer.

```
aws-cloudfsm> 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'
Do you want to continue(y/n)?y
Creating User alice(CO) on 3 nodes
```

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 \(p. 134\)](#) 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 \(p. 120\)](#).

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

  User Id      User Type      User Name      MofnPubKey
LoginFailureCnt
    1          2FA           admin          YES
    0          PCO           app_user        NO
    0          NO            AU             NO
    0          NO            NO             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
    1          2FA           admin          YES
    0          PCO           app_user        NO
    0          NO            AU             NO
    0          NO            NO             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
    1          2FA           admin          YES
    0          PCO           app_user        NO
    0          NO            AU             NO
    0          NO            NO             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 \(p. 111\)](#) 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'
Do you want to continue(y/n)?y
Creating User bob(CU) on 3 nodes
```

Arguments

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To create a CO user with 2FA enabled, use the `-2fa` parameter and include a file path. For more information about 2FA, see [Managing 2FA \(p. 70\)](#).

```
createUser <user-type> <user-name> <password | -hpswd> [-2fa </path/to/authdata>]
```

<user-type>

Specifies the type of user. This parameter is required.

For detailed information about the user types on an HSM, see [Understanding HSM users \(p. 59\)](#).

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.

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 \(p. 31\)](#).

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 `cloudfsm_mgmt_util` commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

<password | -hpswd >

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 hide your password, use the `-hpswd` parameter in place of the password and follow the prompts.

To change a user password, use [changePswd \(p. 111\)](#). 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

[-2fa </path/to/authdata>]

Specifies the creation of a CO user with 2FA enabled. To get the data necessary for setting up 2FA authentication, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about setting up and working with 2FA, see [Managing 2FA \(p. 70\)](#).

Required: No

Related topics

- `listUsers` (p. 134)
 - `deleteUser` (p. 118)
 - `syncUser` (p. 149)
 - `changePswd` (p. 111)

deleteUser

The **deleteUser** command in `cloudhsm_mgmt_util` deletes a user from the hardware security modules (HSM). Only crypto officers (CO) can run this command. You cannot delete a user who is currently logged into a HSM. For more information about deleting users, see [How to Delete HSM Users \(p. 69\)](#).

Tip

You can't delete crypto users (CU) that own keys.

User type

The following types of users can run this command.

- CO

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 \(p. 134\)](#) 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

      User Id          User Type        User Name           MofnPubKey
LoginFailureCnt          2FA
```

1	PCO	admin	YES
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	alice	NO
0	NO		
Users on server 1(10.0.0.2):			
Number of users found:3			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	YES
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	alice	NO
0	NO		
Users on server 1(10.0.0.3):			
Number of users found:3			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	YES
0	NO		
2	AU	app_user	NO
0	NO		
3	CO	alice	NO
0	NO		

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			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	YES
0	NO		
2	AU	app_user	NO
0	NO		
Users on server 1(10.0.0.2):			
Number of users found:2			
User Id	User Type	User Name	MofnPubKey
LoginFailureCnt	2FA		
1	PCO	admin	YES
0	NO		
2	AU	app_user	NO
0	NO		
Users on server 1(10.0.0.3):			

Number of users found:2				
User Id	User Type	User Name	MofnPubKey	
LoginFailureCnt	2FA			
1	PCO	admin	YES	
0	NO			
2	AU	app_user	NO	
0	NO			

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.

Tip

You can't delete crypto users (CU) that own keys.

Valid values are **CO**, **CU**.

To get the user type, use [listUsers \(p. 134\)](#). For detailed information about the user types on an HSM, see [Understanding HSM users \(p. 59\)](#).

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 cloudfsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

Related topics

- [listUsers \(p. 134\)](#)
- [createUser \(p. 115\)](#)
- [syncUser \(p. 149\)](#)
- [changePswd \(p. 111\)](#)

findAllKeys

The **findAllKeys** command in cloudfsm_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).

findAllKeys 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 \(p. 171\)](#) 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 \(p. 134\)](#) to find all users
- [findKey \(p. 171\)](#) in key_mgmt_util to find the keys that they can use
- [getKeyInfo \(p. 200\)](#) in key_mgmt_util to find the owner and shared users of a particular key they own or share

Before you run any CMU command, you must start CMU 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 for CMU. 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.

```
aws-cloudhsm> findAllKeys 4 0
Keys on server 0(10.0.0.1):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 0(10.0.0.1)

Keys on server 1(10.0.0.2):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.2)
```

```
Keys on server 1(10.0.0.3):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.3)
```

Example : 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
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
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49
```

```
findAllKeys success on server 1(10.0.0.2)
```

Arguments

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>]
```

<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 \(p. 134\)](#).

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 \(p. 111\)](#)
- [deleteUser \(p. 118\)](#)
- [listUsers \(p. 134\)](#)
- [syncUser \(p. 149\)](#)
- [findKey \(p. 171\)](#) in `key_mgmt_util`
- [getKeyInfo \(p. 200\)](#) in `key_mgmt_util`

getAttribute

The `getAttribute` command in `cloudfsm_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 \(p. 123\)](#) 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 \(p. 217\)](#) command. To change the attribute values of existing keys, use [setAttribute \(p. 221\)](#) in `key_mgmt_util` and [setAttribute \(p. 142\)](#) in `cloudfsm_mgmt_util`. For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

Before you run any CMU command, you must start CMU 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 for CMU. 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 \(p. 133\)](#) 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 \(p. 235\)](#).

```
aws-cloudfsm> listAttributes

Following are the possible attribute values for getAttribute:

OBJ_ATTR_CLASS          = 0
OBJ_ATTR_TOKEN           = 1
OBJ_ATTR_PRIVATE          = 2
OBJ_ATTR_LABEL            = 3
OBJ_ATTR_TRUSTED          = 134
OBJ_ATTR_KEY_TYPE          = 256
OBJ_ATTR_ID                = 258
OBJ_ATTR_SENSITIVE          = 259
OBJ_ATTR_ENCRYPT           = 260
OBJ_ATTR_DECRYPT           = 261
OBJ_ATTR_WRAP                 = 262
OBJ_ATTR_UNWRAP              = 263
OBJ_ATTR_SIGN                  = 264
OBJ_ATTR_VERIFY                 = 266
OBJ_ATTR_DERIVE                 = 268
OBJ_ATTR_LOCAL                  = 355
```

OBJ_ATTR_MODULUS	= 288
OBJ_ATTR_MODULUS_BITS	= 289
OBJ_ATTR_PUBLIC_EXPONENT	= 290
OBJ_ATTR_VALUE_LEN	= 353
OBJ_ATTR_EXTRACTABLE	= 354
OBJ_ATTR_NEVER_EXTRACTABLE	= 356
OBJ_ATTR_ALWAYS_SENSITIVE	= 357
OBJ_ATTR_DESTROYABLE	= 370
OBJ_ATTR_KCV	= 371
OBJ_ATTR_WRAP_WITH_TRUSTED	= 528
OBJ_ATTR_WRAP_TEMPLATE	= 1073742353
OBJ_ATTR_UNWRAP_TEMPLATE	= 1073742354
OBJ_ATTR_ALL	= 512

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 \(p. 171\)](#) 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 \(p. 200\)](#) 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 \(p. 217\)](#). For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

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 \(p. 194\)](#) in `key_mgmt_util`
- [listAttributes \(p. 133\)](#)
- [setAttribute \(p. 142\)](#) in `cloudhsm_mgmt_util`
- [setAttribute \(p. 221\)](#) in `key_mgmt_util`
- [Key Attribute Reference \(p. 235\)](#)

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 \(p. 127\)](#) section below. To learn about the role of each of these certificates, see [Verify HSM Identity \(p. 18\)](#).

Before you run any CMU command, you must start CMU 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 for CMU. 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 \(p. 140\)](#).

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

- [server \(p. 140\)](#)

getHSMInfo

The **getHSMInfo** command in `cloudfsm_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 `cloudfsm_mgmt_util` uses to refer to the HSM.

Before you run any CMU command, you must start CMU 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 for CMU. 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       :xxxxxxxxxxxxxxxxxx

    ...
```

Related topics

- [info \(p. 132\)](#)

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 \(p. 218\)](#). To find the keys for a particular user, use [findKey \(p. 171\)](#) –u in `key_mgmt_util`. Crypto officers can use [findAllKeys \(p. 120\)](#) 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 \(p. 145\)](#) in `cloudhsm_mgmt_util`.

Before you run any CMU command, you must start CMU 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 for CMU. 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):
        4
        6
Key Info on server 1(10.0.0.2):
    Token/Flash Key,
    Owned by user 3
    also, shared to following 2 user(s):
        4
        6
```

Example : 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 \(p. 189\)](#). 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
Key Info on server 1(10.0.0.2):
    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 \(p. 171\)](#) 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 \(p. 189\)](#) 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 `clouhdhsm_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-clouhdhsm>getKeyInfo 10

Key Info on server 0(10.0.0.1):

    Token/Flash Key,
    Owned by user 3
    also, shared to following 1 user(s):
        4
        2 Users need to approve to use/manage this key
Key Info on server 1(10.0.0.2):

    Token/Flash Key,
    Owned by user 3
    also, shared to following 1 user(s):
        4
        2 Users need to approve to use/manage this key
```

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 \(p. 200\)](#) in `key_mgmt_util`
- [findKey \(p. 171\)](#) in `key_mgmt_util`
- [findAllKeys \(p. 120\)](#) in `cloudhsm_mgmt_util`
- [listUsers \(p. 134\)](#)
- [shareKey \(p. 145\)](#)

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 CMU command, you must start CMU 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 for CMU. 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
0	LoginState 10.0.0.1 Logged in as 'testuser(CU)'	10.0.0.1	2225	Connected	hsm-udw0tkfg1ab

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 \(p. 127\)](#)
- [loginHSM and logoutHSM \(p. 136\)](#)

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. 123)` and `setAttribute (p. 142)` commands.

For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 235\)](#). To represent all attributes, use 512.

```
Command: listAttributes
```

Description
=====
The following are all of the possible attribute values for <code>getAttribute</code> .
OBJ_ATTR_CLASS = 0
OBJ_ATTR_TOKEN = 1
OBJ_ATTR_PRIVATE = 2
OBJ_ATTR_LABEL = 3
OBJ_ATTR_TRUSTED = 134
OBJ_ATTR_KEY_TYPE = 256
OBJ_ATTR_ID = 258
OBJ_ATTR_SENSITIVE = 259
OBJ_ATTR_ENCRYPT = 260
OBJ_ATTR_DECRYPT = 261
OBJ_ATTR_WRAP = 262
OBJ_ATTR_UNWRAP = 263
OBJ_ATTR_SIGN = 264
OBJ_ATTR_VERIFY = 266
OBJ_ATTR_DERIVE = 268
OBJ_ATTR_LOCAL = 355
OBJ_ATTR_MODULUS = 288
OBJ_ATTR_MODULUS_BITS = 289
OBJ_ATTR_PUBLIC_EXPONENT = 290
OBJ_ATTR_VALUE_LEN = 353
OBJ_ATTR_EXTRACTABLE = 354
OBJ_ATTR_NEVER_EXTRACTABLE = 356
OBJ_ATTR_ALWAYS_SENSITIVE = 357
OBJ_ATTR_DESTROYABLE = 370
OBJ_ATTR_KCV = 371
OBJ_ATTR_WRAP_WITH_TRUSTED = 528
OBJ_ATTR_WRAP_TEMPLATE = 1073742353
OBJ_ATTR_UNWRAP_TEMPLATE = 1073742354
OBJ_ATTR_ALL = 512

Parameters

-h

Displays help for the command.

Required: Yes

Related topics

- [getAttribute \(p. 123\)](#)
- [setAttribute \(p. 142\)](#)
- [Key Attribute Reference \(p. 235\)](#)

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 CMU command, you must start CMU 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 for CMU. 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

  User Id      User Type      User Name      MofnPubKey      LoginFailureCnt
  2FA          PCO           admin          YES            0
  1             NO            app_user       NO             0
  2             NO            AU             crypto_user1   NO             0
  3             NO            CU             crypto_user2   NO             0
  4             NO            CO             officer1      YES            0
  5             NO            CO             officer2      NO             0
  6             NO            CO             officer2      NO             0

Users on server 1(10.0.0.2):
Number of users found:5

  User Id      User Type      User Name      MofnPubKey      LoginFailureCnt
  2FA          PCO           admin          YES            0
  1             NO            app_user       NO             0
  2             NO            AU             crypto_user1   NO             0
  3             NO            CU             crypto_user2   NO             0
  4             NO            CO             officer1      YES            0
  5             NO            CO             officer1      NO             0
```

The output includes the following user attributes:

- **User ID:** Identifies the user in `key_mgmt_util` and `cloudhsm_mgmt_util` (p. 106) commands.
- **User type (p. 59):** 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. 74).

- **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 \(p. 218\)](#) in key_mgmt_util
- [createUser \(p. 115\)](#)
- [deleteUser \(p. 118\)](#)
- [changePswd \(p. 111\)](#)

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. To unlock the account, a cryptographic officer (CO) must reset your password using the [changePswd \(p. 111\)](#) command in cloudhsm_mgmt_util.

To troubleshoot loginHSM and logoutHSM

Before you run these cloudhsm_mgmt_util commands, you must start cloudhsm_mgmt_util.

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 on all HSMs in the cluster.

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.

If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts.

User type

The following users can run these commands.

- Precrypto officer (PRECO)
- Crypto officer (CO)
- Crypto user (CU)

Syntax

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To login with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information, see the section called ["Arguments" \(p. 137\)](#).

```
loginHSM <user-type> <user-name> <password> | -hpswd > [-2fa </path/to/authdata>]
```

```
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 you 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 you have 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 in with a hidden password

This command is the same as the example above, except this time you specify that the system should hide the password.

```
aws-cloudhsm>loginHSM CO admin -hpswd
```

The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that you have connected to the HSMs.

```
Enter password:
loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
aws-cloudhsm>
```

Example : Log out of an HSM

This command logs you 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 you have 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

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To login with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information about working with 2FA, see [Managing 2FA \(p. 70\)](#)

```
loginHSM <user-type> <user-name> <password> [-hpswd] [-2fa </path/to/authdata>]
```

<user type>

Specifies the type of user who is logging in to the HSMs. For more information, see [User Type \(p. 136\)](#) above.

Required: Yes

<user name>

Specifies the user name of the user who is logging in to the HSMs.

Required: Yes

<password | -hpswd >

Specifies the password of the user who is logging in to the HSMs. To hide your password, use the `-hpswd` parameter in place of the password and follow the prompt.

Required: Yes

[`-2fa </path/to/authdata>`]

Specifies that the system should use a second factor to authenticate this 2FA-enabled CO user. To get the necessary data for logging in with 2FA, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about working with 2FA, see [Managing 2FA \(p. 70\)](#).

Required: No

Related topics

- [Getting Started with `clouhsm_mgmt_util` \(p. 106\)](#)
- [Activate the Cluster \(p. 31\)](#)

registerQuorumPubKey

The **registerQuorumPubKey** command in `clouhsm_mgmt_util` associates hardware security module (HSM) users with asymmetric RSA-2048 key pairs. Once you associate HSM users with keys, those users can use the private key to approve quorum requests and the cluster can use the registered public key to verify the signature is from the user. For more information about quorum authentication, see [Managing Quorum Authentication \(M of N Access Control\) \(p. 74\)](#).

Tip

In the AWS CloudHSM documentation, quorum authentication is sometimes referred to as M of N (MofN), which means a minimum of *M* approvers out of a total number *N* approvers.

User type

The following types of users can run this command.

- Crypto officers (CO)

Syntax

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

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-token> <public-key>
```

Examples

This example shows how to use **registerQuorumPubKey** to register crypto officers (CO) as approvers on quorum authentication requests. To run this command, you must have an asymmetric RSA-2048 key pair,

a signed token, and an unsigned token. For more information about these requirements, see the section called “[Arguments](#)” (p. 139).

Example : Register a HSM user for quorum authentication

This example registers a CO named `quorum_officer` as an approver for quorum authentication.

```
aws-cloudhsm> registerQuorumPubKey CO <quorum_officer> </path/to/quorum_officer.token> </path/to/quorum_officer.token.sig> </path/to/quorum_officer.pub>

*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****


Do you want to continue(y/n)?y
registerQuorumPubKey success on server 0(10.0.0.1)
```

The final command uses the [listUsers](#) (p. 134) command to verify that `quorum_officer` is registered as an MofN user.

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

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

Arguments

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

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-token> <public-key>
```

<`user-type`>

Specifies the type of user. This parameter is required.

For detailed information about the user types on a HSM, see [Understanding HSM users](#) (p. 59).

Valid values:

- **CO:** Crypto officers can manage users, but they cannot manage keys.

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

<registration-token>

Specifies the path to a file that contains an unsigned registration token. Can have any random data of max file size of 245 bytes. For more information about creating an unsigned registration token, see [Create and Sign a Registration Token \(p. 77\)](#).

Required: Yes

<signed-registration-token>

Specifies the path to a file that contains the SHA256_PKCS mechanism signed hash of the registration-token. For more information, see [Create and Sign a Registration Token \(p. 77\)](#).

Required: Yes

<public-key>

Specifies the path to a file that contains the public key of an asymmetric RSA-2048 key pair. Use the private key to sign the registration token. For more information, see [Create an RSA Key Pair \(p. 77\)](#).

Required: Yes

Note

The cluster uses the same key for quorum authentication and for two-factor authentication (2FA). This means you can't rotate a quorum key for a user that has 2FA enabled using `registerQuorumPubKey`. To rotate the key, you must use `changePswd`. For more information about using quorum authentication and 2FA, see [Quorum Authentication and 2FA \(p. 71\)](#).

Related topics

- [Create an RSA Key Pair \(p. 77\)](#)
- [Create and Sign a Registration Token \(p. 77\)](#)
- [Register the Public Key with the HSM \(p. 77\)](#)
- [Managing Quorum Authentication \(M of N Access Control\) \(p. 74\)](#)
- [Quorum Authentication and 2FA \(p. 71\)](#)
- [listUsers \(p. 134\)](#)

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`.

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.

```
server0> exit
```

Arguments

```
server <server-number>
```

<server-number>

Specifies the server number of the target HSM.

Required: Yes

There are no arguments for the `exit` command.

Related topics

- [syncKey \(p. 148\)](#)
- [createUser \(p. 115\)](#)
- [deleteUser \(p. 118\)](#)

setAttribute

The **setAttribute** command in `cloudhsm_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 \(p. 221\)](#) 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 CMU command, you must start CMU 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 for CMU. 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.

```
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 \(p. 189\)](#) 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 \(p. 133\)](#). 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 \(p. 235\)](#).

```
aws-cloudhsm> listAttributes
Following are the possible attribute values for getAttribute:
    OBJ_ATTR_CLASS          = 0
    OBJ_ATTR_TOKEN           = 1
    OBJ_ATTR_PRIVATE          = 2
```

OBJ_ATTR_LABEL	= 3
OBJ_ATTR_TRUSTED	= 134
OBJ_ATTR_KEY_TYPE	= 256
OBJ_ATTR_ID	= 258
OBJ_ATTR_SENSITIVE	= 259
OBJ_ATTR_ENCRYPT	= 260
OBJ_ATTR_DECRYPT	= 261
OBJ_ATTR_WRAP	= 262
OBJ_ATTR_UNWRAP	= 263
OBJ_ATTR_SIGN	= 264
OBJ_ATTR_VERIFY	= 266
OBJ_ATTR_DERIVE	= 268
OBJ_ATTR_LOCAL	= 355
OBJ_ATTR_MODULUS	= 288
OBJ_ATTR_MODULUS_BITS	= 289
OBJ_ATTR_PUBLIC_EXPONENT	= 290
OBJ_ATTR_VALUE_LEN	= 353
OBJ_ATTR_EXTRACTABLE	= 354
OBJ_ATTR_NEVER_EXTRACTABLE	= 356
OBJ_ATTR_ALWAYS_SENSITIVE	= 357
OBJ_ATTR_DESTROYABLE	= 370
OBJ_ATTR_KCV	= 371
OBJ_ATTR_WRAP_WITH_TRUSTED	= 528
OBJ_ATTR_WRAP_TEMPLATE	= 1073742353
OBJ_ATTR_UNWRAP_TEMPLATE	= 1073742354
OBJ_ATTR_ALL	= 512

To get the current value of the decrypt attribute for key 14, the next command uses [getAttribute \(p. 123\)](#) 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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****


Do you want to continue(y/n)? y
setAttribute success on server 0(10.0.0.1)
setAttribute success on server 1(10.0.0.2)
```

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 \(p. 171\)](#) in `key_mgmt_util`. To find the users of a key, use [getKeyInfo \(p. 128\)](#).

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 \(p. 217\)](#). For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

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 \(p. 221\)](#) in `key_mgmt_util`
- [getAttribute \(p. 123\)](#)
- [listAttributes \(p. 133\)](#)

- [Key Attribute Reference \(p. 235\)](#)

quit

The **quit** command in the `cloudfsm_mgmt_util` exits the `cloudfsm_mgmt_util`. Any user of any type can use this command.

Before you run any `cloudfsm_mgmt_util` command, you must start `cloudfsm_mgmt_util`.

User type

The following users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
quit
```

Example

This command exits `cloudfsm_mgmt_util`. Upon successful completion, you are returned to your regular command line. This command has no output parameters.

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

Related topics

- [Getting Started with `cloudfsm_mgmt_util` \(p. 106\)](#)

shareKey

The **shareKey** command in `cloudfsm_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 CMU command, you must start CMU 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 for CMU. 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 \(p. 189\)](#) 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 \(p. 128\)](#) 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
*****CAUTION*****
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****
Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

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. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****  
  
Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

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 \(p. 171\)](#) in key_mgmt_util. To verify that you own a key, use [getKeyInfo \(p. 128\)](#).

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 \(p. 134\)](#).

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 \(p. 128\)](#)

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 \(p. 59\)](#).

To use **syncKey**, you must first [create an AWS CloudHSM configuration file \(p. 102\)](#) 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`. Then log in 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 [the section called “Understanding HSM users” \(p. 59\)](#).

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 \(p. 134\)](#) command to list all identifiers for named users. Then, use the [findAllKeys \(p. 120\)](#) 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 CMU Across Cloned Clusters \(p. 102\)](#) and initialize `cloudhsm_mgmt_util` with the new config file. Then, enter server mode on the source HSM by issuing the [server \(p. 140\)](#) 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**.

```
aws-cloudhsm> syncKey 261251 1
syncKey success
```

Arguments

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

```
syncKey <key handle> <destination hsm>
```

<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 \(p. 120\)](#) 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 \(p. 134\)](#)
- [findAllKeys \(p. 120\)](#)
- [describe-clusters](#) in AWS CLI
- [server \(p. 140\)](#)

syncUser

You can use the **syncUser** command in `cloudfsm_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 CMU command, you must start CMU 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 for CMU. 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 \(p. 134\)](#) 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 CMU Across Cloned Clusters \(p. 102\)](#) 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 \(p. 140\)](#) 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 \(p. 134\)](#).

Required: Yes

<server ID>

Specifies the server number of the HSM to which you are syncing a user.

Required: Yes

Related topics

- [listUsers \(p. 134\)](#)
- [describe-clusters](#) in AWS CLI
- [server \(p. 140\)](#)

AWS CloudHSM management utility platform support

Linux support

- Amazon Linux
- Amazon Linux 2
- CentOS 6.10+
- CentOS 7.3+
- CentOS 8
- Red Hat Enterprise Linux (RHEL) 6.10+
- Red Hat Enterprise Linux (RHEL) 7.3+
- Red Hat Enterprise Linux (RHEL) 8
- Ubuntu 16.04 LTS
- Ubuntu 18.04 LTS

Windows support

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016

Unsupported platforms:

- Windows 2019

Key Management Utility (KMU)

The key management utility (KMU) is a command line tool that helps crypto users (CU) manage keys on the hardware security modules (HSM). KMU 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 \(p. 152\)](#). For detailed information about the commands, see [key_mgmt_util command reference \(p. 155\)](#). For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

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 \(Linux\) \(p. 28\)](#). If you are using Windows, see [Install and configure the AWS CloudHSM client \(Windows\) \(p. 30\)](#).

Topics

- [Getting started with key_mgmt_util \(p. 152\)](#)
- [key_mgmt_util command reference \(p. 155\)](#)

Getting started with key_mgmt_util

AWS CloudHSM includes two command line tools with the [AWS CloudHSM client software \(p. 28\)](#).

The [cloudhsm_mgmt_util \(p. 109\)](#) tool includes commands to manage HSM users. The [key_mgmt_util \(p. 155\)](#) 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 \(p. 152\)](#)
- [Basic usage of key_mgmt_util \(p. 154\)](#)

If you encounter an error message or unexpected outcome for a command, see the [Troubleshooting AWS CloudHSM \(p. 512\)](#) topics for help. For details about the key_mgmt_util commands, see [key_mgmt_util command reference \(p. 155\)](#).

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 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- 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
```

CentOS 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 16.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 18.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Windows

```
c:\Program Files\Amazon\CloudHSM>key_mgmt_util.exe
```

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 \(p. 523\)](#).

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 \(p. 154\)](#)
- [Log out from the HSMs \(p. 155\)](#)
- [Stop key_mgmt_util \(p. 155\)](#)

Log in to the HSMs

Use the **loginHSM** command to log in to the HSMs. The following command logs in as a [crypto user \(CU\) \(p. 59\)](#) named `example_user`. The output indicates a successful login for all three HSMs in the cluster.

```
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 \(p. 152\)](#). For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#). For information about the `cloudhsm_mgmt_util` command line tool, which includes commands to manage the HSM and users in your cluster, see [CloudHSM Management Utility \(CMU\) \(p. 106\)](#).

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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.

Command	Description
aesWrapUnwrap (p. 157)	Encrypts and decrypts the contents of a key in a file.
deleteKey (p. 160)	Deletes a key from the HSMs.
Error2String (p. 161)	Gets the error that corresponds to a key_mgmt_util hexadecimal error code.
exit (p. 162)	Exits the key_mgmt_util.
exportPrivateKey (p. 163)	Exports a copy of a private key from an HSM to a file on disk.
exportPubKey (p. 164)	Exports a copy of a public key from an HSM to a file.
exSymKey (p. 166)	Exports a plaintext copy of a symmetric key from the HSMs to a file.
extractMaskedObject (p. 170)	Extracts a key from an HSM as a masked object file.
findKey (p. 171)	Search for keys by key attribute value.
findSingleKey (p. 175)	Verifies that a key exists on all HSMs in the cluster.
genDSAKeyPair (p. 176)	Generates a Digital Signing Algorithm (DSA) key pair in your HSMs.
genECCKeyPair (p. 180)	Generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs.
genPBEKey (p. 184)	(This command is not supported on the FIPS-validated HSMs.)
genRSAKeyPair (p. 184)	Generates an RSA asymmetric key pair in your HSMs.
genSymKey (p. 189)	Generates a symmetric key in your HSMs.
getAttribute (p. 194)	Gets the attribute values for an AWS CloudHSM key and writes them to a file.
getCaviumPrivKey (p. 197)	Creates a fake PEM-format version of a private key and exports it to a file.
getCert (p. 198)	Retrieves an HSM's partitions certificates and saves them to a file.
getKeyInfo (p. 200)	Gets the HSM user IDs of users who can use the key. If the key is quorum controlled, it gets the number of users in the quorum.
help (p. 202)	Displays help information about the commands available in key_mgmt_util.
importPrivateKey (p. 204)	Imports a private key into an HSM.

Command	Description
importPubKey (p. 207)	Imports a public key into an HSM.
imSymKey (p. 209)	Imports a plaintext copy of a symmetric key from a file into the HSM.
insertMaskedObject (p. 215)	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) .
??? (p. 216)	Determines whether or not a given file contains a real private key or a fake PEM key.
listAttributes (p. 217)	Lists the attributes of an AWS CloudHSM key and the constants that represent them.
listUsers (p. 218)	Gets the users in the HSMs, their user type and ID, and other attributes.
loginHSM and logoutHSM (p. 220)	Log in and out of the HSMs in a cluster.
setAttribute (p. 221)	Converts a session key to a persistent key.
sign (p. 224)	Generate a signature for a file using a chosen private key.
unWrapKey (p. 226)	Imports a wrapped (encrypted) key from a file into the HSMs.
verify (p. 230)	Verifies whether a given key was used to sign a given file.
wrapKey (p. 232)	Exports an encrypted copy of a key from the HSM to a file.

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 166\)](#) 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.

```
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 \(p. 209\)](#) 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.

```
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 on the HSM. This parameter is required. To find key handles, use the [findKey \(p. 171\)](#) command.

To create a wrapping key, use [genSymKey \(p. 189\)](#) to generate an AES key (type 31).

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 \(p. 166\)](#)
- [imSymKey \(p. 209\)](#)
- [unWrapKey \(p. 226\)](#)
- [wrapKey \(p. 232\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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.

```
Command: deleteKey -k 6  
  
Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS  
  
Cluster Error Status  
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS  
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Delete a key (failure)

When the command fails because no key has the specified key handle, **deleteKey** returns an invalid object handle error message.

```
Command: deleteKey -k 252126  
  
Cfm3FindKey returned: 0xa8 : HSM Error: Invalid object handle is passed to this  
operation  
  
Cluster Error Status  
Node id 1 and err state 0x000000a8 : HSM Error: Invalid object handle is passed to  
this operation  
Node id 2 and err state 0x000000a8 : HSM Error: Invalid object handle is passed to  
this operation
```

When the command fails because the current user is not the owner of the key, the command returns an access denied error.

```
Command: deleteKey -k 262152  
  
Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied.
```

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 \(p. 171\)](#).

Required: Yes

Related topics

- [findKey \(p. 171\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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: Cfm3`command-name` returned: .

In this example, [getKeyInfo \(p. 200\)](#) indicates that the current user (user 4) can use the key in cryptographic operations. Nevertheless, when the user tries to use [deleteKey \(p. 160\)](#) to delete the key, the command returns error code 0xc6.

```
Command: deleteKey -k 262162
Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied
Cluster Error Status
Command: getKeyInfo -k 262162
Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS
Owned by user 3
also, shared to following 1 user(s):
4
```

If the 0xc6 error is reported to you, you can use an **Error2String** command like this one to look up the error. In this case, the 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: Error2String -r 0xa8
Error Code c6 maps to HSM Error: Key Access is denied
```

Parameters

-h

Displays help for the command.

Required: Yes

-r

Specifies a hexadecimal error code. The 0x hexadecimal indicator is required.

Required: Yes

exit

The **exit** command in key_mgmt_util exits the key_mgmt_util. Upon successful exit, you will be returned to your standard command line.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#).

Syntax

```
exit
```

Parameters

There are no parameters for this command.

Related topics

- [Start key_mgmt_util \(p. 153\)](#)

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 \(p. 204\)](#) 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 \(p. 232\)](#).

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 \(p. 235\)](#), 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 \(p. 194\)](#) command.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 171\)](#) command.

To determine whether a key can be used as a wrapping key, use [getAttribute \(p. 194\)](#) to get the value of the OBJ_ATTR_WRAP attribute (262). To create a wrapping key, use [genSymKey \(p. 189\)](#) 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 \(p. 204\)](#)
- [wrapKey \(p. 232\)](#)
- [unWrapKey \(p. 226\)](#)
- [genSymKey \(p. 189\)](#)

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 \(p. 207\)](#) 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 \(p. 235\)](#), 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 \(p. 194\)](#) command.

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 207\)](#)

- [Generate Keys \(p. 98\)](#)

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 \(p. 232\)](#). To import a plaintext key, like the ones that exSymKey exports, use [imSymKey \(p. 209\)](#).

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 \(p. 235\)](#), 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 \(p. 164\)](#). To export private keys, use [exportPrivateKey \(p. 163\)](#).

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 189\)](#) 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.

```
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 \(p. 171\)](#) 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 [the section called "Key Attribute Reference" \(p. 235\)](#).

```
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).

```
// Verify that the key to be exported is a symmetric key.
$ grep -A 1 "OBJ_ATTR_CLASS" attributes/attr_8
OBJ_ATTR_CLASS
0x04

// Verify that the key to be exported is extractable.
$ grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_8
OBJ_ATTR_EXTRACTABLE
0x00000001

// Verify that the wrapping key is an AES key
$ grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_262168
OBJ_ATTR_KEY_TYPE
0x1f

// Verify that the wrapping key is a session key
$ grep -A 1 "OBJ_ATTR_TOKEN" attributes/attr_262168
OBJ_ATTR_TOKEN
```

```
0x00

// Verify that the wrapping key can be used for wrapping
$ grep -A 1 "OBJ_ATTR_WRAP" attributes/attr_262168
OBJ_ATTR_WRAP
0x00000001
```

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 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 that 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 \(p. 209\)](#) 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
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

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.

```
Command: exSymKey -k 21 -w 262167 -out aes192_H21.key -wk aes256-forImport.key
Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS
Wrapped Symmetric Key written to file "aes192_H21.key"
```

Parameters

-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 \(p. 171\)](#) command.

To verify that a key can be exported, use the [getAttribute \(p. 194\)](#) command to get the value of the `OBJ_ATTR_EXTRACTABLE` attribute, which is represented by constant 354. Also, you can export only keys that you own. To find the owner of a key, use the [getKeyInfo \(p. 200\)](#) command.

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the [findKey \(p. 171\)](#) 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 \(p. 194\)](#) to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by the constant 262. To create a wrapping key, use [genSymKey \(p. 189\)](#) 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

-m

Specifies the wrapping mechanism. The only valid value is 4, which represents the `NIST_AES_WRAP` mechanism.

Required: No

Default: 4

-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, `exSymKey` uses the key in the HSM that is specified by the `-w` parameter to wrap the key that is being exported and it uses the key in the `-wk` file to unwrap it. The `-w` and `-wk` parameter values must resolve to the same plaintext key.

Required: No

Default: Use the wrapping key on the HSM to unwrap.

Related topics

- [genSymKey \(p. 189\)](#)
- [imSymKey \(p. 209\)](#)
- [wrapKey \(p. 232\)](#)

extractMaskedObject

The **extractMaskedObject** command in `key_mgmt_util` extracts a key from an HSM and saves it to a file as a masked object. Masked objects are *cloned* objects that can only be used after inserting them back into the original cluster by using the [insertMaskedObject \(p. 215\)](#) 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 [copying a backup across regions \(p. 54\)](#) and [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 `OBJ_ATTR_EXTRACTABLE (p. 235)` 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 \(p. 238\)](#).

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 [syncKey \(p. 148\)](#) in the `cloudhsm_mgmt_util` instead.

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM. The **extractMaskedObject** command can be used either by the CU who owns the key or any CO.

Syntax

```
extractMaskedObject -h  
  
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 \(p. 215\)](#)
- [syncKey \(p. 148\)](#)
- [Copying a Backup Across Regions \(p. 54\)](#)
- [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 \(p. 194\)](#).

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 194\)](#). To determine whether the current user owns or shares a particular key, use [getKeyInfo \(p. 200\)](#) or [findAllKeys \(p. 120\)](#) in clouduhsm_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, 262156, 262157, 262158, 262159, 262160, 262161, 262162

Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : 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 \(p. 218\)](#).

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 \(p. 120\)](#) 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 \(p. 194\)](#).

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:** KCV 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.

```
Command: findKey
Total number of keys present 10

number of keys matched from start index 0::9
6, 7, 8, 9, 10, 11, 262156, 262157, 262158, 262159

Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Related topics

- [findSingleKey \(p. 175\)](#)
- [getKeyInfo \(p. 200\)](#)
- [getAttribute \(p. 194\)](#)
- [findAllKeys \(p. 120\) in cloudhsm_mgmt_util](#)
- [Key Attribute Reference \(p. 235\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
findSingleKey -h  
findSingleKey -k <key-handle>
```

Example

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 \(p. 218\)](#) command.

Required: Yes

Related topics

- [findKey \(p. 218\)](#)
- [getKeyInfo \(p. 218\)](#)

- [getAttribute \(p. 171\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 194\)](#). To find the keys for a particular user, use [getKeyInfo \(p. 200\)](#). To find keys based on their attribute values, use [findKey \(p. 171\)](#).

Syntax

```
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: 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: genDSAKeyPair -m 2048 -l DSA-temp -id DSA_temp_pair -sess

Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:     public key handle: 12     private key handle: 14

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

To confirm that the key pair exists only in the session, use the `-sess` parameter of [findKey \(p. 171\)](#) with a value of 1 (true).

```
Command: findKey -sess 1

Total number of keys present 2

number of keys matched from start index 0::1
12, 14

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : 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.

```
Command: genDSAKeyPair -m 2048 -l DSA -id DSA_shared_pair -nex -u 3,5,6

Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:     public key handle: 11     private key handle: 19

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Create a quorum-controlled key pair

This command creates a DSA key pair with the label DSA-mV2. The command uses the `-u` parameter to share the private key with user 4 and 6. It uses the `-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 `-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.

```
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
```

This command uses [getKeyInfo \(p. 200\)](#) 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: getKeyValue -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 \(p. 200\)](#).

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 \(p. 100\)](#). 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 \(p. 221\)](#).

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 \(p. 218\)](#). To share and unshare existing keys, use [shareKey \(p. 145\)](#) 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 \(p. 184\)](#)
- [genSymKey \(p. 189\)](#)
- [genECCKeyPair \(p. 180\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 194\)](#). To find the keys for a particular user, use [getKeyInfo \(p. 200\)](#). To find keys based on their attribute values, use [findKey \(p. 171\)](#).

Syntax

```
genECCKeyPair -h

genECCKeyPair -i <EC curve id>
    -l <label>
    [-id <key ID>]
    [-min_srv <minimum number of servers>]
    [-m_value <0..8>]
    [-nex]
    [-sess]
    [-timeout <number of seconds> ]
    [-u <user-ids>]
    [-attest]
```

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 \(p. 194\)](#) 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
0xbbbb32a
OBJ_ATTR_MODULUS
044a0f9d01d10f7437d9fa20995f0cc742552e5ba16d3d7e9a65a33e20ad3e569e68eb62477a9960a87911e6121d112b698e469
OBJ_ATTR_MODULUS_BITS
0x0000019f
```

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 \(p. 200\)](#).

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 \(p. 100\)](#). 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 \(p. 221\)](#).

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 \(p. 218\)](#). To share and unshare existing keys, use [shareKey \(p. 145\)](#) 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 \(p. 189\)](#)
- [genRSAKeyPair \(p. 184\)](#)
- [genDSAKeyPair \(p. 176\)](#)

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 \(p. 189\)](#). To create asymmetric key pairs, use [genRSAKeyPair \(p. 184\)](#), [genDSAKeyPair \(p. 176\)](#), or [genECCKeyPair \(p. 180\)](#).

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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 194\)](#). To find the keys for a particular user, use [getKeyInfo \(p. 200\)](#). To find keys based on their attribute values, use [findKey \(p. 171\)](#).

Syntax

```
genRSAKeyPair -h

genRSAKeyPair -m <modulus length>
    -e <public exponent>
    -l <label>
    [-id <key ID>]
    [-min_srv <minimum number of servers>]
    [-m_value <0..8>]
    [-nex]
    [-sess]
    [-timeout <number of seconds> ]
    [-u <user-ids>]
    [-attest]
```

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 65537. The output shows that the public key handle is 2100177 and the private key handle is 2100426.

```
Command: genRSAKeyPair -m 2048 -e 65537 -l rsa_test

Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

    Cfm3GenerateKeyPair:    public key handle: 2100177    private key handle: 2100426

    Cluster Status:
    Node id 0 status: 0x00000000 : HSM Return: SUCCESS
    Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

The next command uses [getAttribute \(p. 194\)](#) to get the attributes of the public key that we just created. It writes the output to the `attr_2100177` 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 \(p. 235\)](#).

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`). The results also include the key length (512, `0x200`), the modulus, the modulus length (2048, `0x800`), and the public exponent (65537, `0x10001`).

```
Command: getAttribute -o 2100177 -a 512 -out attr_2100177

Attribute size: 801, count: 26
Written to: attr_2100177 file

    Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS

$ cat attr_2100177
OBJ_ATTR_CLASS
0x02
OBJ_ATTR_KEY_TYPE
0x00
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
```

```

0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x01
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x00
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
rsa_test
OBJ_ATTR_ID

OBJ_ATTR_VALUE_LEN
0x00000200
OBJ_ATTR_KCV
0xc51c18
OBJ_ATTR_MODULUS
0xbb9301cc362c1d9724eb93da8adab0364296bde7124a241087d9436b9be57e4f7780040df03c2c
1c0fe6e3b61aa83c205280119452868f66541bbbfacbbe787b8284fc81deaef2b8ec0ba25a077d
6983c77a1de7b17cbe8e15b203868704c6452c2810344a7f2736012424cf0703cf15a37183a1d2d0
97240829f8f90b063dd3a41171402b162578d581980976653935431da0c1260bfe756d85dca63857
d9f27a541676cb9c7def0ef6a2a89c9b9304bcac16fdf8183c0a555421f9ad5dfeb534cf26b65873
970cdf1a07484f1c128b53e10209cc6f7ac308669112968c81a5de408e7f644fe58b1a9ae1286fec
b3e4203294a96fae06f8f0db7982cb5d7f
OBJ_ATTR_MODULUS_BITS
0x00000800
OBJ_ATTR_PUBLIC_EXPONENT
0x010001
OBJ_ATTR_TRUSTED
0x00
OBJ_ATTR_WRAP_WITH_TRUSTED
0x00
OBJ_ATTR_DESTROYABLE
0x01
OBJ_ATTR_DERIVE
0x00
OBJ_ATTR_ALWAYS_SENSITIVE
0x00
OBJ_ATTR_NEVER_EXTRACTABLE
0x00

```

Example : 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 65537 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2

Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

```

```
Cfm3GenerateKeyPair:    public key handle: 27    private key handle: 28

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

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 \(p. 200\)](#).

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 \(p. 100\)](#). 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 \(p. 221\)](#).

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 \(p. 218\)](#). To share and unshare existing keys, use [shareKey \(p. 145\)](#) 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 \(p. 189\)](#)
- [genDSAKeyPair \(p. 176\)](#)
- [genECCKeyPair \(p. 180\)](#)

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 \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
genSymKey -h

genSymKey -t <key-type>
           -s <key-size>
           -l <label>
           [-id <key-ID>]
           [-min_srv <minimum-number-of-servers>]
           [-m_value <0..8>]
           [-nex]
           [-sess]
           [-timeout <number-of-seconds> ]
           [-u <user-ids>]
           [-attest]
```

Examples

These examples show how to use **genSymKey** to create symmetric keys in your HSMs.

Tip

To use the keys you make with these examples for HMAC operations, you must set `OBJ_ATTR_SIGN` and `OBJ_ATTR_VERIFY` to `TRUE` after you generate the key. To set these values, use **setAttribute** in CloudHSM Management Utility (CMU). For more information, see [setAttribute \(p. 142\)](#).

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: 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: 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 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 \(p. 200\)](#). 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 \(p. 194\)](#). 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.

Tip

To find the attributes of a key that you have created, such as the type, length, label, and ID, use [getAttribute \(p. 194\)](#). To find the keys for a particular user, use [getKeyInfo \(p. 200\)](#). To find keys based on their attribute values, use [findKey \(p. 171\)](#).

For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

```
Command: getAttribute -o 7 -a 512 -attr_7

got all attributes of size 444 attr cnt 17
Attributes dumped into attr_7 file

Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS

$ cat attr_7

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
```

```
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
3DES_shared
OBJ_ATTR_ID
IT-02
OBJ_ATTR_VALUE_LEN
0x00000018
OBJ_ATTR_KCV
0x59a46e
```

Tip

To use the keys you make with these examples for HMAC operations, you must set `OBJ_ATTR_SIGN` and `OBJ_ATTR_VERIFY` to `TRUE` after you generate the key. To set these values, use `setAttribute` in CMU. For more information, see [setAttribute \(p. 142\)](#).

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 \(p. 200\)](#).

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 \(p. 100\)](#).

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 \(p. 221\)](#).

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 \(p. 218\)](#). To share and unshare existing keys, use [shareKey \(p. 145\)](#) in the `cloudhsm_mgmt_util`.

Default: Only the current user can use the key.

Required: No

Related topics

- [exSymKey \(p. 166\)](#)
- [genRSAKeyPair \(p. 184\)](#)
- [genDSAKeyPair \(p. 176\)](#)
- [genECCKeyPair \(p. 180\)](#)
- [setAttribute \(p. 142\)](#)

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 \(p. 123\)](#) 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 \(p. 217\)](#) command. To change the attribute values of existing keys, use [setAttribute \(p. 221\)](#) in `key_mgmt_util` and [setAttribute \(p. 142\)](#) in `cloudhsm_mgmt_util`. For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 217\)](#), 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 \(p. 235\)](#).

```
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 \(p. 235\)](#).

```
$ 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 \(p. 235\)](#).

```
$ cat attribute.txt

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x01
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
```

```
aes256
OBJ_ATTR_ID
test_01
OBJ_ATTR_VALUE_LEN
0x00000020
OBJ_ATTR_KCV
0x1a4b31
```

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 \(p. 171\)](#).

Also, you must own the specified key or it must be shared with you. To find the users of a key, use [getKeyInfo \(p. 200\)](#).

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 \(p. 217\)](#). For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

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 \(p. 123\)](#) in `cloudhsm_mgmt_util`
- [listAttributes \(p. 217\)](#)
- [setAttribute \(p. 221\)](#)
- [findKey \(p. 171\)](#)
- [Key Attribute Reference \(p. 235\)](#)

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 \(p. 393\)](#).

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [login \(p. 154\)](#) 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 \(p. 204\)](#)
- [SSL/TLS Offload on Linux \(p. 393\)](#)

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 \(p. 199\)](#) section that follows. To learn about the role of each of these certificates, see [Verify HSM Identity \(p. 18\)](#).

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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

-f

Specifies the name of the file to which the retrieved certificate will be saved.

Required: Yes

-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 \(p. 18\)](#)

- [getCert \(p. 126\)](#) (in [cloudhsm_mgmt_util \(p. 106\)](#))

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 \(p. 218\)](#). To find the keys for a particular user, use [findKey \(p. 171\)](#) -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 \(p. 145\)](#) in cloudhsm_mgmt_util.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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.

```
Command: getKeyInfo -k 9  
  
Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS  
  
Owned by user 3  
  
also, shared to following 1 user(s):  
  
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 \(p. 171\)](#).

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 **-m_value** parameter when you create the key.

This command uses [genRSAKeyPair \(p. 184\)](#) to create an RSA key pair that is shared with user 4. It uses the **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.

```
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
```

This command uses **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.

```
Command: getKeyInfo -k 28

Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

Owned by user 3

also, shared to following 1 user(s):

        4
2 Users need to approve to use/manage this key
```

Parameters

-h

Displays command line help for the command.

Required: Yes

-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 [findKey \(p. 218\)](#) command.

Required: Yes

Related topics

- [getKeyInfo \(p. 128\)](#) in `cloudhsm_mgmt_util`
- [listUsers \(p. 218\)](#)
- [findKey \(p. 171\)](#)
- [findAllKeys \(p. 120\)](#) 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` \(p. 153\)](#).

Syntax

```
help
```

Example

This example shows the output of the `help` command.

Example

```
Command: help

Help Commands Available:

Syntax: <command> -h
```

Command	Description
=====	=====
exit	Exits this application
help	Displays this information
	Configuration and Admin Commands
getHSMInfo	Gets the HSM Information
getPartitionInfo	Gets the Partition Information
listUsers	Lists all users of a partition
loginStatus	Gets the Login Information
loginHSM	Login to the HSM
logoutHSM	Logout from the HSM
	M of N commands
getToken	Initiate an MxN service and get Token
delToken	delete Token(s)
approveToken	Approves an MxN service
listTokens	List all Tokens in the current partition
	Key Generation Commands
	Asymmetric Keys:
genRSAKeyPair	Generates an RSA Key Pair
genDSAKeyPair	Generates a DSA Key Pair
genECCKeyPair	Generates an ECC Key Pair
	Symmetric Keys:
genPBEKey	Generates a PBE DES3 key
genSymKey	Generates a Symmetric keys
	Key Import/Export Commands
createPublicKey	Creates an RSA public key
importPubKey	Imports RSA/DSA/EC Public key
exportPubKey	Exports RSA/DSA/EC Public key
importPrivateKey	Imports RSA/DSA/EC private key
exportPrivateKey	Exports RSA/DSA/EC private key
imSymKey	Imports a Symmetric key
exSymKey	Exports a Symmetric key
wrapKey	Wraps a key from HSM using the specified handle
unWrapKey	UnWraps a key into HSM using the specified handle
	Key Management Commands
deleteKey	Delete Key
setAttribute	Sets an attribute of an object
getKeyInfo	Get Key Info about shared users/sessions
findKey	Find Key
findSingleKey	Find single Key
getAttribute	Reads an attribute from an object
	Certificate Setup Commands
getCert	Gets Partition Certificates stored on HSM
	Key Transfer Commands
insertMaskedObject	Inserts a masked object
extractMaskedObject	Extracts a masked object
	Management Crypto Commands
sign	Generates a signature
verify	Verifies a signature
aesWrapUnwrap	Does NIST AES Wrap/Unwrap
	Helper Commands
Error2String	Converts Error codes to Strings

getCaviumPrivKey	save key handle in fake PEM format Saves an RSA private key handle in fake PEM format
isValidKeyHandlefile	Checks if private key file has an HSM key handle or a real key
listAttributes	List all attributes for getAttributes
listECCCurveIds	List HSM supported ECC CurveIds

Parameters

There are no parameters for this command.

Related topics

- [loginHSM and logoutHSM \(p. 220\)](#)

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 \(p. 163\)](#) 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 \(p. 232\)](#).

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 \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
importPrivateKey -h

importPrivateKey -l <label>
    -f <key-file>
    -w <wrapping-key-handle>
    [-sess]
    [-id <key-id>]
    [-m_value <0...8>]
    [min_srv <minimum-number-of-servers>]
    [-timeout <number-of-seconds>]
    [-u <user-ids>]
    [-wk <wrapping-key-file>]
    [-attest]
```

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 \(p. 171\)](#) command.

To determine whether a key can be used as a wrapping key, use [getAttribute \(p. 194\)](#) to get the value of the OBJ_ATTR_WRAP attribute (262). To create a wrapping key, use [genSymKey \(p. 189\)](#) 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

-timout

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

-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 \(p. 218\)](#).

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 \(p. 232\)](#)
- [unWrapKey \(p. 226\)](#)
- [genSymKey \(p. 189\)](#)
- [exportPrivateKey \(p. 163\)](#)

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 \(p. 164\)](#) command.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 164\)](#)
- [Generate Keys \(p. 98\)](#)

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 \(p. 166\)](#), 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 \(p. 232\)](#) and [unWrapKey \(p. 226\)](#) commands.

Also, the **imSymKey** command imports only symmetric keys. To import public keys, use [importPubKey \(p. 207\)](#). To import private keys, use [importPrivateKey \(p. 204\)](#) or [wrapKey \(p. 232\)](#).

Imported keys work very much like keys generated in the HSM. However, the value of the [OBJ_ATTR_LOCAL attribute \(p. 235\)](#) 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 \(p. 106\)](#) 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 \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
imSymKey -h

imSymKey -f <key-file>
          -w <wrapping-key-handle>
          -t <key-type>
          -l <label>
          [-id <key-ID>]
          [-sess]
          [-wk <wrapping-key-file> ]
          [-attest]
          [-min_srv <minimum-number-of-servers>]
          [-timeout <number-of-seconds> ]
          [-u <user-ids>]
```

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 \(p. 194\)](#) to get the `OBJ_ATTR_LOCAL` attribute ([attribute 355 \(p. 235\)](#)) of the newly imported key and writes it to the `attr_262180` file.

```
Command: getAttribute -o 262180 -a 355 -out attributes/attr_262180
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
OBJ_ATTR_LOCAL
0x00000000
```

Example : Move a symmetric key between clusters

This example shows how to use [exSymKey \(p. 166\)](#) 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 \(p. 232\)](#) and [unWrapKey \(p. 226\)](#) 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 \(p. 166\)](#) 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 252152, an AES key on the HSMs in cluster 2, as the wrapping key.

Because the wrapping keys that [exSymKey \(p. 166\)](#) 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
Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Unwrapped. Key Handle: 21

Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

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 \(p. 194\)](#) 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
0xc33cbd
```

This similar command uses [getAttribute \(p. 194\)](#) 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
Attributes dumped into attr_21_kcv file

$ 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 \(p. 194\)](#) or [findKey \(p. 171\)](#) 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 the 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

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 \(p. 221\)](#).

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

-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](#)

Required: Yes

-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.

Required: No

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the [findKey \(p. 171\)](#) 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 \(p. 194\)](#) to get the value of the `OBJ_ATTR_WRAP` attribute (262). To create a wrapping key, use [genSymKey \(p. 189\)](#) 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 \(p. 189\)](#)
- [exSymKey \(p. 166\)](#)
- [wrapKey \(p. 232\)](#)
- [unWrapKey \(p. 226\)](#)
- [exportPrivateKey \(p. 163\)](#)
- [exportPubKey \(p. 164\)](#)

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 \(p. 170\)](#) 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 \(p. 54\)](#) and [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 [OBJ_ATTR_EXTRACTABLE \(p. 235\)](#) 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 \(p. 238\)](#).

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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-serv** parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No

Related topics

- [extractMaskedObject \(p. 170\)](#)
- [syncKey \(p. 148\)](#)
- [Copying a Backup Across Regions \(p. 54\)](#)
- [Creating an AWS CloudHSM Cluster from a Previous Backup \(p. 46\)](#)

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 \(p. 393\)](#).

Before you run any `key_mgmt_util` command, you must [start `key_mgmt_util` \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
isValidKeyHandlefile -h
```

```
IsValidKeyHandlefile -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 \(p. 197\)](#)
- [SSL/TLS Offload on Linux \(p. 393\)](#)

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 \(p. 194\)](#) and [setAttribute \(p. 221\)](#) commands. For help interpreting the key attributes, see the [Key Attribute Reference \(p. 235\)](#).

Before you run any `key_mgmt_util` command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 235\)](#).

To represent all attributes in the [getAttribute \(p. 194\)](#) 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 \(p. 133\)](#) in cloudfsm_mgmt_util
- [getAttribute \(p. 194\)](#)
- [setAttribute \(p. 221\)](#)
- [Key Attribute Reference \(p. 235\)](#)

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 \(p. 218\)](#) command in cloudfsm_mgmt_util.

The user commands in key_mgmt_util, **listUsers** and [getKeyInfo \(p. 200\)](#), are read-only commands that crypto users (CUs) have permission to run. The remaining user management commands are part of cloudfsm_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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 171\)](#), [getAttribute \(p. 194\)](#), and [getKeyInfo \(p. 200\)](#).

```
Command: listUsers

        Number Of Users found 4

      Index      User ID      User Type      User Name      MofnPubKey
LoginFailureCnt          2FA
      1            1       PCO      admin           NO
0          NO
      2            2       AU      app_user        NO
0          NO
      3            3       CU      alice         YES
0          NO
      4            4       CU      bob            NO
0          NO
      5            5       CU      trent         YES
0          NO

Cfm3ListUsers returned: 0x00 : HSM Return: SUCCESS
```

The output includes the following user attributes:

- **User ID:** Identifies the user in `key_mgmt_util` and [cloudhsm_mgmt_util \(p. 106\)](#) commands.
- **User type (p. 59):** 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 \(p. 74\)](#).
- **LoginFailureCnt:**
- **2FA:** Indicates that the user has enabled multi-factor authentication.

Parameters

-h

Displays help for the command.

Required: Yes

Related topics

- [listUsers \(p. 218\)](#) in `cloudhsm_mgmt_util`
- [findKey \(p. 171\)](#)
- [getAttribute \(p. 194\)](#)
- [getKeyInfo \(p. 200\)](#)

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 \(p. 153\)](#). In order to manage keys with key_mgmt_util, you must log in to the HSMs as a [crypto user \(CU\) \(p. 60\)](#).

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 \(p. 111\)](#) command in cloudfsm_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 | -hpswd } <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 you into the HSMs as a crypto user (CU) with the username `example_user` and password `aws`. The output shows 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 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 in with a hidden password

This command is the same as the example above, except this time you specify that the system should hide the password.

```
Command: loginHSM -u CU -s example_user -hpswd
```

The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that you have connected to the HSMs.

```
Enter password:
```

```
Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

Command:
```

Example : Log out of the HSMs

This command logs you out of the HSMs. The output shows that you have logged out of all HSMs in the cluster.

```
Command: logoutHSM

Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for this command.

-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.

Required: Yes

{ -p | -hpswd }

Specify the login password with **-p**. The password appears in plaintext when you type it. To hide your password, use the optional **-hpswd** parameter instead of **-p** and follow the prompt.

Required: Yes

Related topics

- [exit \(p. 162\)](#)

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` \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 189\)](#) 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 \(p. 171\)](#) 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 \(p. 235\)](#).

The command uses the `-o` parameter to specify the key handle (262154) and the `-a` parameter to specify the constant that represents the token attribute (1). When you run the command, it prompts you for a value for the token attribute. The only valid value is 1 (true); the value for a persistent key.

```
Command: setAttribute -o 262154 -a 1
This attribute is defined as a boolean value.
Enter the boolean attribute value (0 or 1):1
```

```
Cfm3SetAttribute returned: 0x00 : HSM Return: SUCCESS  
  
Cluster Error Status  
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS  
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

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 \(p. 171\)](#).

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 \(p. 217\)](#).

Required: Yes

Related topics

- [setAttribute \(p. 142\)](#) in `cloudhsm_mgmt_util`

- [getAttribute \(p. 194\)](#)
- [listAttributes \(p. 217\)](#)
- [Key Attribute Reference \(p. 235\)](#)

sign

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 \(p. 189\)](#), [genRSAKeyPair \(p. 184\)](#), or [genECCKeyPair \(p. 180\)](#) commands. You can also import one with the [importPrivateKey \(p. 204\)](#) command. For more information, see [Generate Keys \(p. 98\)](#).

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 \(p. 224\)](#).

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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`.

```
Command: sign -f messageFile -k 266309 -m 1 -out signedFile

Cfm3Sign returned: 0x00 : HSM Return: SUCCESS

signature is written to file signedFile

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

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 follow integers:

Signing Mechanism	Corresponding Integer
SHA1_RSA_PKCS	0
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17
ECDSA_SHA384	18
ECDSA_SHA512	19

Required: Yes

-out

The name of the file to which the signed file will be saved.

Required: Yes

Related topics

- [verify \(p. 230\)](#)
- [importPrivateKey \(p. 204\)](#)
- [genRSAKeyPair \(p. 184\)](#)
- [genECCKeyPair \(p. 180\)](#)
- [genSymKey \(p. 189\)](#)
- [Generate Keys \(p. 98\)](#)

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 \(p. 232\)](#) 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 \(p. 324\)](#) or [JCE \(p. 352\)](#) software libraries to unwrap the key.

Imported keys work like keys generated by AWS CloudHSM. However, the value of their [OBJ_ATTR_LOCAL attribute \(p. 235\)](#) 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 \(p. 153\)](#) and [log in \(p. 154\)](#) 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 \(p. 232\)](#) 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.

```
Command: unWrapKey -f 3DES.key -w 6 -m 4
```

```
Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS
Key Unwrapped. Key Handle: 29
Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : 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.

```
Command: unWrapKey -f 3DES.key -w 6 -noheader -l unwrapped3DES -kc 4 -kt 21 -m 4

Cfm3CreateUnwrapTemplate2 returned: 0x00 : HSM Return: SUCCESS
Cfm2UnWrapWithTemplate3 returned: 0x00 : HSM Return: SUCCESS

Key Unwrapped. Key Handle: 8
Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

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 \(p. 171\)](#) command.

To create a wrapping key, use [genSymKey \(p. 189\)](#) to generate an AES key (type 31) or [genRSAKeyPair \(p. 184\)](#) 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 \(p. 194\)](#) 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 \(p. 221\)](#).

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:

Mechanism	Value
AES_KEY_WRAP_PAD_PKCS5	4
NIST_AES_WRAP_NO_PAD	5
NIST_AES_WRAP_PAD	6
RSA_AES	7
RSA_OAEP (for maximum data size, see the note later in this section)	8
NIST_TDEA_WRAP (key data size must be multiple of 4 bytes)	9
AES_GCM	10
CLOUDHSM_AES_GCM	11

Required: Yes

-t

Hash algorithm	Value
SHA1	2
SHA256	3
SHA384	4
SHA512	5
SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)	6

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 \(p. 232\)](#)
- [exSymKey \(p. 166\)](#)
- [imSymKey \(p. 209\)](#)

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 \(p. 224\)](#) operation.

Signing mechanisms are represented by the integers listed in the [parameters \(p. 231\)](#) section.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) 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 : 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 follow integers:

Signing Mechanism	Corresponding Integer
SHA1_RSA_PKCS	0
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6

Signing Mechanism	Corresponding Integer
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17
ECDSA_SHA384	18
ECDSA_SHA512	19

Required: Yes

Related topics

- [sign \(p. 224\)](#)
- [getCert \(p. 180\)](#)
- [Generate Keys \(p. 98\)](#)

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 \(p. 226\)](#). To export a plaintext key from an HSM, use [exSymKey \(p. 166\)](#) or [exportPrivateKey \(p. 163\)](#) as appropriate. The [aesWrapUnwrap \(p. 157\)](#) command cannot decrypt (unwrap) keys that **wrapKey** encrypts.

Before you run any key_mgmt_util command, you must [start key_mgmt_util \(p. 153\)](#) and [log in \(p. 154\)](#) to the HSM as a crypto user (CU).

Syntax

```
wrapKey -h

wrapKey -k <exported-key-handle>
        -w <wrapping-key-handle>
        -out <output-file>
        [-m <wrapping-mechanism>]
        [-aad <additional authenticated data filename>]
        [-t <hash-type>]
```

```
[ -noheader ]
[ -i <wrapping IV> ]
[ -iv_file <IV file> ]
[ -tag_size <num_tag_bytes>> ]
```

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 \(p. 171\)](#) command.

To verify that a key can be exported, use the [getAttribute \(p. 194\)](#) 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 \(p. 235\)](#).

You can export only those keys that you own. To find the owner of a key, use the [getKeyInfo \(p. 200\)](#) 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 \(p. 171\)](#) command.

To create a wrapping key, use [genSymKey \(p. 189\)](#) to generate an AES key (type 31) or [genRSAKeyPair \(p. 184\)](#) 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 \(p. 194\)](#) 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:

Mechanism	Value
AES_KEY_WRAP_PAD_PKCS5	4
NIST_AES_WRAP_NO_PAD	5
NIST_AES_WRAP_PAD	6
RSA_AES	7
RSA_OAEP (for maximum data size, see the note later in this section)	8
NIST_TDEA_WRAP (key data size must be multiple of 4 bytes)	9
AES_GCM	10
CLOUDHSM_AES_GCM	11

Required: Yes

-t

The value representing the hash algorithm. CloudHSM supports the following algorithms:

Hash algorithm	Value
SHA1	2
SHA256	3
SHA384	4
SHA512	5
SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)	6

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 \(p. 155\)](#). 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 \(p. 166\)](#)
- [imSymKey \(p. 209\)](#)
- [unWrapKey \(p. 226\)](#)

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 \(p. 221\)](#) 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 \(p. 217\)](#). To get the attribute values for a key, use [getAttribute \(p. 194\)](#).

The following table lists the key attributes, their constants, and their valid values.

Attribute	Constant	Values
OBJ_ATTR_CLASS	0	2: Public key in a public–private key pair. 3: Private key in a public–private key pair. 4: Secret (symmetric) key.
OBJ_ATTR_TOKEN	1	0: False. Session key. 1: True. Persistent key.
OBJ_ATTR_PRIVATE	2	0: False. 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.
OBJ_ATTR_LABEL	3	User-defined string. It does not have to be unique in the cluster.
OBJ_ATTR_TRUSTED	134	0: False. 1: True.
OBJ_ATTR_KEY_TYPE	256	0: RSA. 1: DSA. 3: EC. 16: Generic secret. 18: RC4. 21: Triple DES (3DES). 31: AES.
OBJ_ATTR_ID	258	User-defined string. Must be unique in the cluster. The default is an empty string.
OBJ_ATTR_SENSITIVE	259	0: False. Public key in a public–private key pair. 1: True.
OBJ_ATTR_ENCRYPT	260	0: False. 1: True. The key can be used to encrypt data.

Attribute	Constant	Values
OBJ_ATTR_DECRYPT	261	0: False. 1: True. The key can be used to decrypt data.
OBJ_ATTR_WRAP	262	0: False. 1: True. The key can be used to encrypt keys.
OBJ_ATTR_UNWRAP	263	0: False. 1: True. The key can be used to decrypt keys.
OBJ_ATTR_SIGN	264	0: False. 1: True. The key can be used for signing (private keys).
OBJ_ATTR_VERIFY	266	0: False. 1: True. The key can be used for verification (public keys).
OBJ_ATTR_DERIVE	268	0: False. 1: True. The function derives the key.
OBJ_ATTR_MODULUS	288	The modulus that was used to generate an RSA key pair. For other key types, this attribute does not exist.
OBJ_ATTR_MODULUS_BITS	289	The length of the modulus used to generate an RSA key pair. For other key types, this attribute does not exist.
OBJ_ATTR_PUBLIC_EXPONENT	290	The public exponent used to generate an RSA key pair. For other key types, this attribute does not exist.
OBJ_ATTR_VALUE_LEN	353	Key length in bytes.
OBJ_ATTR_EXTRACTABLE	354	0: False. 1: True. The key can be exported from the HSMs.
OBJ_ATTR_LOCAL	355	0: False. The key was imported into the HSMs. 1: True.

Attribute	Constant	Values
OBJ_ATTR_NEVER_EXTRACTABLE	356	0: False. 1: True. The key cannot be exported from the HSMs.
OBJ_ATTR_ALWAYS_SENSITIVE	357	0: False. 1: True.
OBJ_ATTR_DESTROYABLE	370	0: False. 1: True.
OBJ_ATTR_KCV	371	Key check value of the key. For more information, see Additional Details (p. 238) .
OBJ_ATTR_ALL	512	Represents all attributes.
OBJ_ATTR_WRAP_WITH_TRUSTED	528	0: False. 1: True.
OBJ_ATTR_WRAP_TEMPLATE	1073742353	Values should use the attribute template to match the key wrapped using this wrapping key..
OBJ_ATTR_UNWRAP_TEMPLATE	1073742354	Values should use the attribute template applied to any key unwrapped using this wrapping key.

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 \(p. 194\)](#).

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:** KCV for HMAC keys is not supported at this time.

Configure tool

AWS CloudHSM automatically synchronizes data among all hardware security modules (HSM) 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.

Topics

- [Client SDK 3 configure tool \(p. 239\)](#)
- [Client SDK 5 configure tool \(p. 245\)](#)

Client SDK 3 configure tool

Use the Client SDK 3 configure tool to bootstrap the client daemon and configure CloudHSM Management Utility.

Syntax

```
configure -h | --help
  -a <ENI IP address>
  -m [-i <daemon_id>]
  --ssl --pkey <private key file> --cert <certificate file>
  --cmu <ENI IP address>
```

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. To use the –a parameter, you must have the IP address for one of the HSMs in your cluster. Use either the console or the AWS CLI to get the IP address.

To get an IP address for a HSM (console)

1. Open the AWS CloudHSM console at <https://console.aws.amazon.com/cloudhsm/>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Clusters**.
4. To open the cluster detail page, in the cluster table, choose the cluster ID.
5. To get the IP address, on the HSMs tab, choose one of the IP addresses listed under **ENI IP address**.

To get an IP address for a HSM (AWS CLI)

- Get the IP address of an HSM by using the **describe-clusters** command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of **EniIp**.

```
$ aws cloudhsmv2 describe-clusters

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

```
"EniIp": "10.0.1.6",  
...
```

To update the HSM data

1. 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

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

Amazon Linux 2

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

CentOS 7

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

CentOS 8

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

RHEL 7

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

RHEL 8

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

Ubuntu 16.04 LTS

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

Ubuntu 18.04 LTS

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

Windows

- 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.

2. 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 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 16.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 18.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Windows

```
c:\Program Files\Amazon\CloudHSM>configure.exe -a 10.0.0.9
```

3. 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

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- 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
```

When the command completes, the HSM data that the AWS CloudHSM client and key_mgmt_util use is complete and accurate.

Example : Update the HSM Data for CMU from client SDK 3.2.1 and earlier

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. Use this with CMU that ships with Client SDK 3.2.1 and earlier.

- 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 \(p. 239\)](#). This ensures that the data copied into the `cloudhsm_mgmt_util.cfg` file from the `cluster.info` file is complete and accurate.

Linux

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

Windows

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

Example : Update the HSM Data for CMU from client SDK 3.3.0 and later

This example uses the `--cmu` parameter of the **configure** command to update HSM data for CMU. Use this with CMU that ships with Client SDK 3.3.0 and later. For more information about using CMU, see [Using CloudHSM Management Utility \(CMU\) to Manage Users \(p. 61\)](#) and [Using CMU with Client SDK 3.2.1 and Earlier \(p. 62\)](#).

- Use the `--cmu` parameter to pass the IP address of an HSM in your cluster.

Linux

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

Windows

```
c:\Program Files\Amazon\CloudHSM>configure.exe --cmu <IP address>
```

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 \(p. 239\)](#).

This parameter edits the following configuration files:

- `/opt/cloudhsm/etc/cloudhsm_client.cfg`: Used by AWS CloudHSM client and [key_mgmt_util \(p. 152\)](#).
- `/opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg`: Used by [cloudhsm_mgmt_util \(p. 106\)](#).

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 CMU uses.

Note

The **-m** parameter is for use with CMU from Client SDK 3.2.1 and earlier. For CMU from Client SDK 3.3.0 and later, see **--cmu** parameter, which simplifies the process of updating HSM data for CMU.

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.

--cmu <ENI IP address>

Combines the **-a** and **-m** parameters into one parameter. Adds the specified HSM elastic network interface (ENI) IP address to AWS CloudHSM configuration files and then updates the CMU configuration file. Enter an IP address from any HSM in the cluster. For Client SDK 3.2.1 and earlier, see [Using CMU with Client SDK 3.2.1 and Earlier \(p. 62\)](#).

Required: Yes

Related topics

- [Set up key_mgmt_util \(p. 152\)](#)

Client SDK 5 configure tool

Use the Client SDK 5 configure tool to update client-side configuration files.

Each component in Client SDK 5 includes a configure tool with a designator of the component in the file name of the configure tool. For example, the PKCS #11 library for Client SDK 5 includes a configure tool named `configure-pkcs11` on Linux or `configure-pkcs11.exe` on Windows.

Syntax

```
configure-[ pkcs11 | dyn | jce ][ .exe ]
  -a <ENI IP address>
  [--hsm-ca-cert <CustomerCA/certificate/file>]
  [--cluster-id <Cluster ID>]
  [--endpoint <Endpoint>]
  [--region <Region>]
  [--server-client-cert-file <Client/certificate/file>]
  [--server-client-key-file <Private/key/file>]
  [--log-level <Error | warn | info | debug | trace>]
  [--log-rotation <Weekly | daily>]
  [--log-file <Log/file/path>]
  [-h | --help]
  [-V | --version]
  [--disable-key-availability-check]
  [--enable-key-availability-check]
  [--disable-validate-key-at-init]
  [--enable-validate-key-at-init]
```

Examples

These examples show how to use the configure tool for Client SDK 5.

Bootstrap Client SDK 5

Example

This example uses the `-a` parameter to update the HSM data for Client SDK 5. To use the `-a` parameter, you must have the IP address for one of the HSMs in your cluster.

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of a HSM in your cluster.

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

For more information about the –a parameter, see [the section called “Parameters” \(p. 255\)](#).

Specify cluster Region and endpoint for Client SDK 5

Example

This example uses the `cluster-id` parameter to bootstrap Client SDK 5 by making a `DescribeClusters` call.

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of a HSM in your cluster.

```
sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id cluster-1234567
```

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of a HSM in your cluster.

```
C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe --cluster-id cluster-1234567
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of a HSM in your cluster.

```
sudo /opt/cloudhsm/bin/configure-dyn --cluster-id cluster-1234567
```

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of a HSM in your cluster.

```
sudo /opt/cloudhsm/bin/configure-jce --cluster-id cluster-1234567
```

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of a HSM in your cluster.

```
C:\Program Files\Amazon\CloudHSM\configure-jce.exe --cluster-id cluster-1234567
```

You can use the `--region` and `--endpoint` parameters in combination with the `cluster-id` parameter to specify how the system makes the `DescribeClusters` call. For instance, if the region of the cluster is different than the one configured as your AWS CLI default, you should use the `--region` parameter to use that region. Additionally, you have the ability to specify the AWS CloudHSM API endpoint to use for the call, which might be necessary for various network setups, such as using VPC interface endpoints that don't use the default DNS hostname for AWS CloudHSM.

PKCS #11 library

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of a HSM in your cluster with a custom region and endpoint.

```
sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of a HSM in your cluster with a custom region and endpoint.

```
C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of a HSM in your cluster with a custom region and endpoint.

```
sudo /opt/cloudhsm/bin/configure-dyn --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

JCE provider

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of a HSM in your cluster with a custom region and endpoint.

```
sudo /opt/cloudhsm/bin/configure-jce --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of a HSM in your cluster with a custom region and endpoint.

```
C:\Program Files\Amazon\CloudHSM\configure-jce.exe --cluster-id cluster-1234567 --region us-east-1 --endpoint https://cloudhsmv2.us-east-1.amazonaws.com
```

For more information about the --cluster-id, --region, and --endpoint parameters, see [the section called “Parameters” \(p. 255\)](#).

Update client certificate and key for TLS client-server mutual authentication

Example

This examples shows how to use the server-client-cert-file and --server-client-key-file parameters to reconfigure SSL by specifying a custom key and SSL certificate for AWS CloudHSM

PKCS #11 library

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify `ssl-client.crt` and `ssl-client.key`.

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

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

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

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.crt` and `ssl-client.key`.

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

OpenSSL Dynamic Engine

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify `ssl-client.crt` and `ssl-client.key`.

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

JCE provider

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

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

2. Use the configure tool to specify `ssl-client.crt` and `ssl-client.key`.

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

To use a custom certificate and key for TLS client-server mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

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

2. With a PowerShell interpreter, use the configure tool to specify ssl-client.crt and ssl-client.key.

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

For more information about the server-client-cert-file and --server-client-key-file parameters, see [the section called “Parameters” \(p. 255\)](#).

Disable client key durability settings

Example

This example uses the --disable-key-availability-check parameter to disable client key durability settings. To run a cluster with a single HSM, you must disable client key durability settings.

PKCS #11 library

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
C:\Program Files\Amazon\CloudHSM\configure-pkcs11.exe --disable-key-availability-
check
```

OpenSSL Dynamic Engine

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

JCE provider

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
C:\Program Files\Amazon\CloudHSM\configure-jce.exe --disable-key-availability-check
```

For more information about the `--disable-key-availability-check` parameter, see [the section called “Parameters” \(p. 255\)](#).

Manage logging options

Example

This example uses the `log-file`, `log-level`, and `log-rotation` parameters to manage Client SDK 5 logging.

PKCS #11 library

To configure the name of the logging file

- Use the `log-file` option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-file path/to/log
```

For example, use the following command to set the log file name to `cloudhsm-pkcs11.log`.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-file cloudhsm-pkcs11.log
```

If you do not specify a location for the file, the system writes logs to the default location.

- Linux:

```
/opt/cloudhsm/run
```

- Windows:

```
C:\ProgramData\Amazon\CloudHSM
```

To configure the logging level

- Use the log-level option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level warn
```

To configure log rotation

- Use the log-rotation option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-rotation daily
```

OpenSSL Dynamic Engine

To configure the name of the logging file

- Use the log-file option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-file path/to/log
```

For example, use the following command to set the log file name to *cloudhsm-dyn.log*.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-file cloudhsm-dyn.log
```

If you do not specify a location for the file, the system writes logs to stderr

To configure the logging level

- Use the log-level option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-level warn
```

To configure log rotation

- Use the log-rotation option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-rotation daily
```

JCE provider

To configure the name of the logging file

- Use the log-file option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-file path/to/log
```

For example, use the following command to set the log file name to *cloudhsm-dyn.log*.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-file cloudhsm-dyn.log
```

If you do not specify a location for the file, the system writes logs to the default location:

- Linux:

```
/opt/cloudhsm/run
```

- Windows:

```
C:\ProgramData\Amazon\CloudHSM
```

To configure the logging level

- Use the log-level option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-level warn
```

To configure log rotation

- Use the `log-rotation` option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-rotation daily
```

For more information about the `log-file`, `log-level`, and `log-rotation` parameters, see [the section called "Parameters" \(p. 255\)](#).

Place the issuing certificate for Client SDK 5

Example

This example uses the `--hsm-ca-cert` parameter to update the location of the issuing certificate for Client SDK 5.

PKCS #11 library

To place the issuing certificate on Linux for Client SDK 5

- Use the `configure` tool to specify a location for the issuing certificate.

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

To place the issuing certificate on Windows for Client SDK 5

- Use the `configure` tool to specify a location for the issuing certificate.

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

OpenSSL Dynamic Engine

To place the issuing certificate on Linux for Client SDK 5

- Use the `configure` tool to specify a location for the issuing certificate.

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

JCE provider

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

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

For more information about the --hsm-ca-cert parameter, see [the section called "Parameters" \(p. 255\)](#).

Parameters

-a <ENI IP address>

Adds the specified IP address to Client SDK 5 configuration files. Enter any ENI IP address of a HSM from the cluster. For more information about how to use this option, see [Bootstrap Client SDK 5 \(p. 40\)](#).

Required: Yes

--hsm-ca-cert <CustomerCA/certificate/file>

Path to the directory storing the certificate authority (CA) certificate use to connect EC2 client instances to the cluster. You create this file when you initialize the cluster. By default, the system looks for this file in the following location:

Linux

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

Windows

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

For more information about initializing the cluster or placing the certificate, see [??? \(p. 38\)](#) and [??? \(p. 24\)](#).

Required: No

--cluster-id <Cluster ID>

Makes a `DescribeClusters` call to find all of the HSM elastic network interface (ENI) IP addresses in the cluster associated with the cluster ID. The system adds the ENI IP addresses to the AWS CloudHSM configuration files.

Note

If you use the `--cluster-id` parameter from an EC2 instance within a VPC that does not have access to the public internet, then you must create an interface VPC endpoint to connect with AWS CloudHSM. For more information about VPC endpoints, see [??? \(p. 510\)](#).

Required: No

--endpoint <Endpoint>

Specify the AWS CloudHSM API endpoint used for making the `DescribeClusters` call. You must set this option in combination with `--cluster-id`.

Required: No

--region <Region>

Specify the region of your cluster. You must set this option in combination with `--cluster-id`.

If you don't supply the `--region` parameter, the system chooses the region by attempting to read the `AWS_DEFAULT_REGION` or `AWS_REGION` environment variables. If those variables aren't set, then the system checks the region associated with your profile in your AWS config file (typically `~/.aws/config`) unless you specified a different file in the `AWS_CONFIG_FILE` environment variable. If none of the above are set, the system defaults to the `us-east-1` region.

Required: No

--server-client-cert-file <Client/certificate/file>

Path to the client certificate used for TLS client-server mutual authentication.

Only use this option if you don't wish to use the default key and SSL/TLS certificate we include with Client SDK 5. You must set this option in combination with `--server-client-key-file`.

Required: No

--server-client-key-file <Client/key/file>

Path to the client key used for TLS client-server mutual authentication

Only use this option if you don't wish to use the default key and SSL/TLS certificate we include with Client SDK 5. You must set this option in combination with `--server-client-cert-file`.

Required: No

--log-level <error | warn | info | debug | trace>

Specifies the minimum logging level the system should write to the log file. Each level includes the previous levels, with error as the minimum level and trace the maximum level. This means that if you specify errors, the system only writes errors to the log. If you specify trace, the system writes errors, warnings, informational (info) and debug messages to the log. For more information, see [Client SDK 5 Logging \(p. 480\)](#).

Required: No

--log-rotation <weekly | daily>

Specifies the frequency with which the system rotates logs. For more information, see [Client SDK 5 Logging \(p. 480\)](#).

Required: No

--log-file <Path/log/file>

Specifies where the system will write the log file. For more information, see [Client SDK 5 Logging \(p. 480\)](#).

Required: No

-h | --help

Displays help.

Required: No

-v | --version

Displays version.

Required: No

--disable-key-availability-check

Flag to disable key availability quorum. Use this flag to indicate AWS CloudHSM should disable key availability quorum and you can use keys that exist on only one HSM in the cluster. For more information about using this flag to set key availability quorum, see [??? \(p. 90\)](#).

Required: No

--enable-key-availability-check

Flag to enable key availability quorum. Use this flag to indicate AWS CloudHSM should use key availability quorum and not allow you to use keys until those keys exist on two HSMs in the cluster. For more information about using this flag to set key availability quorum, see [??? \(p. 90\)](#).

Enabled by default.

Required: No

--disable-validate-key-at-init

Improves performance by specifying that you can skip an initialization call to verify permissions on a key for subsequent calls. Use with caution.

Background: Some mechanisms in the PKCS #11 library support multi-part operations where an initialization call verifies if you can use the key for subsequent calls. This requires a verification call to the HSM, which adds latency to the overall operation. This option enables you to disable the subsequent call and potentially improve performance.

Required: No

--enable-validate-key-at-init

Specifies that you should use an initialization call to verify permissions on a key for subsequent calls. This is the default option. Use `enable-validate-key-at-init` to resume these initialization calls after you use `disable-validate-key-at-init` to suspend them.

Required: No

Related topics

- [DescribeClusters API operation](#)
- [describe-clusters AWS CLI](#)
- [Get-HSM2Cluster PowerShell cmdlet](#)
- [Bootstrap Client SDK 5 \(p. 40\)](#)
- [AWS CloudHSM VPC endpoints \(p. 510\)](#)
- [Managing Client SDK 5 Key Durability Settings \(p. 90\)](#)
- [Client SDK 5 Logging \(p. 480\)](#)

AWS CloudHSM Client SDKs

You use a Client SDK to offload cryptographic operations to the hardware security modules (HSM) from various platform or language-based applications and to perform certain management tasks on the HSM. AWS CloudHSM offers two major versions, Client SDK 5 and Client SDK 3. For more information, see [Choose a Client SDK version \(p. 258\)](#).

[the section called “PKCS #11 library” \(p. 324\)](#)

The Organization for Advancement of Structured Information Standards (OASIS) owns the ongoing development of the Public Key Cryptography Standards (PKCS). The PKCS #11 standard defines a platform-independent API to perform cryptographic operations on hardware devices like a HSM. Use these libraries to perform cryptographic operations on the HSM. For more information, see [the section called “PKCS #11 library” \(p. 324\)](#).

[the section called “OpenSSL Dynamic Engine” \(p. 347\)](#)

OpenSSL is the default library for cryptographic operations on Linux based operating systems. You can use OpenSSL to perform cryptographic operations to a HSM with an engine. In open source, an engine is similar to a provider. The CloudHSM implementation of an OpenSSL dynamic engine offloads limited operations to the HSM. The most common use case for this is SSL/TLS offload using Apache or Nginx. For more information, see [the section called “OpenSSL Dynamic Engine” \(p. 347\)](#).

[the section called “JCE provider” \(p. 352\)](#)

Java Cryptography Extensions (JCE) provides advanced cryptographic operations like encryption, decryption, and key generation in Java. While the Java Development Kit (JDK) ships with a default implementation of JCE, that implementation is not designed for use with an HSM. However, the JCE offers an architecture that allows vendors to write providers that can easily plug into the JDK. CloudHSM provides one such implementation of a JCE provider that allows Java applications to offload cryptographic operations to the HSM. For more information, see [the section called “JCE provider” \(p. 352\)](#).

[the section called “KSP and CNG Providers” \(p. 383\)](#)

For Microsoft Windows, CloudHSM provides a limited implementation of the CNG and KSP providers that allow selected cryptographic operations to be offloaded to the HSM. The most common use case for this is Active Directory Certificate Services (AD CS). For more information, see [the section called “KSP and CNG Providers” \(p. 383\)](#).

Choose a Client SDK version to work with AWS CloudHSM

AWS CloudHSM includes two major Client SDK versions:

- Client SDK 3, which includes a full set of components for platform and language-based applications compatibility and management tools.
- Client SDK 5, which does not require a client daemon, and offers true cross-platform support for Windows and Linux.

To download, see [the section called “Downloads” \(p. 260\)](#).

Client SDK comparison

In addition to the command-line tools, Client SDK 3 contains components that enable off-loading cryptographic operations to the HSM from various platform or language-based applications. Client SDK 5 does not have parity with Client SDK 3 at this time, but parity is the goal. The following table compares component availability in Client SDK 3 and Client SDK 5.

Component	Client SDK 3	Client SDK 5
PKCS #11 library	Yes	Yes
JCE provider	Yes	Yes
OpenSSL Dynamic Engine	Yes	Yes
CNG and KSP providers	Yes	
CloudHSM Management Utility	Yes ¹	
Key management utility	Yes	
Configure tool	Yes	Yes

[1] To manage HSM users with Client SDK 5, AWS CloudHSM offers a standalone version of CMU. For more information, see [??? \(p. 62\)](#) and [??? \(p. 64\)](#).

Check your client SDK version

Amazon Linux

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

Amazon Linux 2

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 6

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 7

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 8

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 6

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 7

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 8

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

Ubuntu 16.04 LTS

Use the following command:

```
apt list --installed | grep ^cloudhsm
```

Ubuntu 18.04 LTS

Use the following command:

```
apt list --installed | grep ^cloudhsm
```

Windows Server

Use the following command:

```
wmic product get name,version
```

Download AWS CloudHSM Client SDK

In March 2021, AWS CloudHSM released Client SDK version 5.0.0, which introduces an all-new Client SDK with different requirements, capabilities, and platform support. You now have two versions of the Client SDK to choose from, Client SDK 5 and Client SDK 3. For more information, see [the section called “Choose a Client SDK version” \(p. 258\)](#).

Releases

- [Latest releases \(p. 261\)](#)
 - Latest Client SDK 5 release: [Version 5.5.0 \(p. 261\)](#)
 - Latest Client SDK 3 release: [Version 3.4.4 \(p. 263\)](#)
- [Previous Client SDK 5 releases \(p. 266\)](#)
- [Previous Client SDK 3 releases \(p. 284\)](#)
- [Deprecated releases \(p. 314\)](#)
- [End-of-life releases \(p. 314\)](#)

Latest releases

This section includes the latest version of each Client SDK.

Client SDK 5 is fully supported for production environments, but it does not offer every component or the same level of support for cryptographic operations as Client SDK 3. For more information, see [the section called "Client SDK comparison" \(p. 259\)](#).

Versions

- [Latest Client SDK 5 release: Version 5.5.0 \(p. 261\)](#)
- [Latest Client SDK 3 release: Version 3.4.4 \(p. 263\)](#)

Latest Client SDK 5 release: Version 5.5.0

Amazon Linux

Download version 5.5.0 software for Amazon Linux on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum d26dcf4b1850f1ace52a386496063fb1e4a7692f34a1f00d1e4e61a8f59b3363)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 098d3ee36db307e803ae6badff5c57e7581649a25d2fb789bf2e8e5ff851910b)
- [JCE provider](#) (SHA256 checksum 3106dcf572870fbfa0645d82705d509677c2439527b5ad35ec8456ad5d068161)

Amazon Linux 2

Download version 5.5.0 software for Amazon Linux 2 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum c08f71fe022d0041c897f326f5cff17ca2ed5b3d308e9801dd7889ca9e92068e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum bbb90f0bb7bbac9594177b7a9ff92d04bc16853a78b5016f7f5cd40b8a3ed539)
- [JCE provider](#) (SHA256 checksum 65f1d6360985f4a5225b89bccd276b15408a4d09ceb41489c4fa743970315bae)

Download version 5.5.0 software for Amazon Linux 2 on ARM64 architecture:

- [JCE provider](#) (SHA256 checksum 958de0815636e77cbd79e0b60d4b721d4b47a13652f8d5d036bd30d8b202b1e6)

CentOS 7

Download version 5.5.0 software for CentOS 7 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum c08f71fe022d0041c897f326f5cff17ca2ed5b3d308e9801dd7889ca9e92068e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum bbb90f0bb7bbac9594177b7a9ff92d04bc16853a78b5016f7f5cd40b8a3ed539)
- [JCE provider](#) (SHA256 checksum 65f1d6360985f4a5225b89bccd276b15408a4d09ceb41489c4fa743970315bae)

RHEL 7

Download version 5.5.0 software for RHEL 7 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum c08f71fe022d0041c897f326f5cff17ca2ed5b3d308e9801dd7889ca9e92068e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum bbb90f0bb7bbac9594177b7a9ff92d04bc16853a78b5016f7f5cd40b8a3ed539)

- [JCE provider](#) (SHA256 checksum 65f1d6360985f4a5225b89bccd276b15408a4d09ceb41489c4fa743970315bae)

RHEL 8

Download version 5.5.0 software for RHEL 8 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum a27fb3315377aa94f2c757fcc5108673f9f8d50b5fd7423e7e9aa7484e1e451)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum bb52bcf1ed306918bf68782e84012d2ee3de369f7fab5b805e62afe87371335e)
- [JCE provider](#) (SHA256 checksum 3ffd94fe5241e5d74234df4fba20d61f3d6223534bd2b6e03f9179defa0f3e8e)

Ubuntu 18.04 LTS

Download version 5.5.0 software for Ubuntu 18.04 LTS on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum 00aa40e909e73ad96b480073e3b959e5de4ea853b539f9604e9aa592dcdbcfe5)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 1dabd957d1e5777d3da7da7273885b1541868656b033c7ad8d9fde133e03da6a)
- [JCE provider](#) (SHA256 checksum 702a560938718e59120653b5c3359cb6a14f6b835f4c7d3aab0df26071e2ff63)

Windows Server 2016

Download version 5.5.0 software for Windows Server 2016 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum df4197ebfb623acf3b733fc7b4bdf38a76cb09d63059680d56a3af37f740af4)
- [JCE provider](#) (SHA256 checksum b8dc0a7a54715596c0eb40b6e97978850230ec1f76acf926fe6dbc622b2ac0b3)

Windows Server 2019

Download version 5.5.0 software for Windows Server 2019 on x86_64 architecture:

- [PKCS #11 library](#) (SHA256 checksum df4197ebfb623acf3b733fc7b4bdf38a76cb09d63059680d56a3af37f740af4)
- [JCE provider](#) (SHA256 checksum b8dc0a7a54715596c0eb40b6e97978850230ec1f76acf926fe6dbc622b2ac0b3)

Version 5.5.0 adds support for OpenJDK 11, Keytool and Jarsigner integration, and additional mechanisms to the JCE provider. Resolves a [known issue](#) regarding a KeyGenerator class incorrectly interpreting key size parameter as number of bytes instead of bits.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Support for the Keytool and Jarsigner utilities
- Support for OpenJDK 11 on all platforms
- Ciphers
 - AES/CBC/NoPadding Encrypt and Decrypt mode

- AES/ECB/PKCS5Padding Encrypt and Decrypt mode
- AES/CTR/NoPadding Encrypt and Decrypt mode
- AES/GCM/NoPadding Wrap and Unwrap mode
- DESede/ECB/PKCS5Padding Encrypt and Decrypt mode
- DESede/CBC/NoPadding Encrypt and Decrypt mode
- AESWrap/ECB/NoPadding Wrap and Unwrap mode
- AESWrap/ECB/PKCS5Padding Wrap and Unwrap mode
- AESWrap/ECB/ZeroPadding Wrap and Unwrap mode
- RSA/ECB/PKCS1Padding Wrap and Unwrap mode
- RSA/ECB/OAEPPadding Wrap and Unwrap mode
- RSA/ECB/OAEPWithSHA-1ANDMGF1Padding Wrap and Unwrap mode
- RSA/ECB/OAEPWithSHA-224ANDMGF1Padding Wrap and Unwrap mode
- RSA/ECB/OAEPWithSHA-256ANDMGF1Padding Wrap and Unwrap mode
- RSA/ECB/OAEPWithSHA-384ANDMGF1Padding Wrap and Unwrap mode
- RSA/ECB/OAEPWithSHA-512ANDMGF1Padding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPPadding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPWithSHA-1ANDMGF1Padding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPWithSHA-224ANDMGF1Padding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPWithSHA-256ANDMGF1Padding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPWithSHA-384ANDMGF1Padding Wrap and Unwrap mode
- RSAAESWrap/ECB/OAEPWithSHA-512ANDMGF1Padding Wrap and Unwrap mode
- KeyFactory and SecretKeyFactory
 - 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
 - DESede (3DES)
 - GenericSecret
 - HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support
- Sign/Verify
 - RSASSA-PSS
 - SHA1withRSA/PSS
 - SHA224withRSA/PSS
 - SHA256withRSA/PSS
 - SHA384withRSA/PSS
 - SHA512withRSA/PSS
 - SHA1withRSAandMGF1
 - SHA224withRSAandMGF1
 - SHA256withRSAandMGF1
 - SHA384withRSAandMGF1
 - SHA512withRSAandMGF1

Latest Client SDK 3 release: Version 3.4.4

To upgrade Client SDK 3 on Linux platforms, you must use a batch command that upgrades the client daemon and all the libraries at the same time. For more information about upgrade, see [Client SDK 3 Upgrade \(p. 317\)](#).

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 3.4.4 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum 900de424d70f41e661aa636f256a6a79cc43bea6b0fe6eb95c2aaa63e5289505)
- [PKCS #11 library](#) (SHA256 checksum a3f93f084d59fee5d7c859292bc02cb7e7f15fb06e971171ebf9b52bbd229c30)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 8db07b9843d49016b0b6fec46d39881d94e426fc当地1ce2747be14af9313bb0)
- [JCE provider](#) (SHA256 checksum 360617c55bf4caa8e6e78ede079ca68cf9ef11473e7918154c22ba908a219843)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum c9961ffe38921131bd6f3702e10d73588e68b8ab10fbb241723e676f4fa8c4fa)

Amazon Linux 2

Download the version 3.4.4 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- [PKCS #11 library](#) (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- [JCE provider](#) (SHA256 checksum 70e3cdce143c45a76e155ff5969841e0153e011f59eb9f2c6e6be0707030abf)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.4.4.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.4.4 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- [PKCS #11 library](#) (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- [JCE provider](#) (SHA256 checksum 70e3cdce143c45a76e155ff5969841e0153e011f59eb9f2c6e6be0707030abf)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

CentOS 8

Download the version 3.4.4 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 81639c9ec83e501709c4117ba9d98b23dea7838a206ed244c9c6cc0d65130f8c)
- [PKCS #11 library](#) (SHA256 checksum 9a15daa87b8616cf03a6bf6b375f53451ef448dbc54bf2c27fbc2be7823fc633)
- [JCE provider](#) (SHA256 checksum 2b1c4208992903cf7bcc669c1392c59a64fbfc82e010c626ffa58d0cb8e9126b)

- [AWS CloudHSM Management Utility](#) (SHA256 checksum
3adbecc802e0854c23aa4b8d80540d1748903c8dba93b6c8042fb7885051c360)

Note

Due to the recent End of Life of CentOS 8, we will no longer be able to support this platform with next release.

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.4.4.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.4.4 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 7d61d835ae38c6ce121d102b516527f342a76ac31733768097d5cab8bc482610)
- [PKCS #11 library](#) (SHA256 checksum 2099f324ff625e1a46d96c1d5084263ca1d650424d7465ead43fe767d6687f36)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
6d8e81ad1208652904fe4b6abc4f174e866303f2302a6551c3fbef617337e663)
- [JCE provider](#) (SHA256 checksum 70e3cdce143c45a76e155ffb5969841e0153e011f59eb9f2c6e6be0707030abf)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
5a702fe5e50dc6055daa723df71a0874317c9ff5844eea30104587a61097ecf4)

RHEL 8

Download the version 3.4.4 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 81639c9ec83e501709c4117ba9d98b23dea7838a206ed244c9c6cc0d65130f8c)
- [PKCS #11 library](#) (SHA256 checksum 9a15daa87b8616cf03a6bf6b375f53451ef448dbc54bf2c27fbc2be7823fc633)
- [JCE provider](#) (SHA256 checksum 2b1c4208992903cf7bcc669c1392c59a64fbfc82e010c626ffa58d0cb8e9126b)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
3adbecc802e0854c23aa4b8d80540d1748903c8dba93b6c8042fb7885051c360)

Ubuntu 16.04 LTS

Download the version 3.4.4 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 317c92c2e0b5d60afab1beb947f053d13ddaacb994cccc2c2b898e997ece29b9)
- [PKCS #11 library](#) (SHA256 checksum 91451c420c51488a022569fd32f052a3b988a2883ea4c2ac952acb61a2fea37c)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
4098771ad0e38df9bf14d50520ca49b9395f819f0387e2bc3b0e61abb5888e66)
- [JCE provider](#) (SHA256 checksum e136ff183271c2f9590a9fccb8261a7eb809506686b070e3854df1b8686c6641)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
cbf24a4032f393a913a9898b1b27036392104e8e05d911cab84049b2bcca2541)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.4.4 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum cf57d5e0e95efbf032aac8887aebd59ac8cc80e97c69e7c39fdad40873374fe8)
- [PKCS #11 library](#) (SHA256 checksum 428f8bdad7925db5401112f707942ee8f3ca554f4ab53fa92237996e69144d2f)
- [JCE provider](#) (SHA256 checksum 1ff17b8f7688e84f7f0bfc96383564dca598a1cab2f2c52c888d0361682f2b9e)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum afe253046146ed6177c520b681efc680dac1048c4a95b3d8ad0f305e79bbe93e)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.4.4 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.4.4) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum 2fe2d9039034b57d82324bd3ed3e867801acbfb3ae48caa63a9801ce6202e758)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 195445261c6a97d81ef27c0ed5e856031f5be2632bc819d3a3b50be1e8507bdf)

Version 3.4.4 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.17.1.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Previous Client SDK 5 releases

This section lists previous Client SDK 5 releases. For the current Client SDK 5 release, see [the section called “Latest releases” \(p. 261\)](#).

Version 5.4.2

Amazon Linux

Download the version 5.4.2 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum dd904ebbb8c9cd8a58c8ed1ba6b329ee92dfe45d89e4ac807030c0421aadbed6)

- [OpenSSL Dynamic Engine](#) (SHA256 checksum
bbf81f8aa24092b7ab3c6c1b2011c46f1304ee6f003ef630a600673ed906a0a5)
- [JCE provider](#) (SHA256 checksum ea3f178bd35bef7400421c2b66755cc6130e9297960f36ca5aefb64537afeb1b)

Amazon Linux 2

Download the version 5.4.2 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum 8d892210db097a942ae99213c8f20fe78798840d208b0c5329baa9eda6783b9e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
68cb82352a9880d16035181bd14ee4085ff16b48b9d8d501df352dc0fb214009)
- [JCE provider](#) (SHA256 checksum 891a99f9c85182dd11c62f1745363b1fbe2ae442abbaea01c9681cd2d32ffe1c)

CentOS 7

Download the version 5.4.2 software for CentOS 7:

- [PKCS #11 library](#) (SHA256 checksum 8d892210db097a942ae99213c8f20fe78798840d208b0c5329baa9eda6783b9e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
68cb82352a9880d16035181bd14ee4085ff16b48b9d8d501df352dc0fb214009)
- [JCE provider](#) (SHA256 checksum 891a99f9c85182dd11c62f1745363b1fbe2ae442abbaea01c9681cd2d32ffe1c)

CentOS 8

Download the version 5.4.2 software for CentOS 8:

- [PKCS #11 library](#) (SHA256 checksum bf3ffa7bc1c4fd7bc9509e4f7c8bc3a013dbdae362cacf6ed278730fa43a0da0)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
2da903322f91f7f2ba32edf48676f4b9f23d83ceb4395ea7a51c6a0f132ddc9b)
- [JCE provider](#) (SHA256 checksum d5d1d813fdbf34e98873209606d8a55d56181598b1a539a0a0a62d4efd0addf2)

Note

Due to the recent End of Life of CentOS 8, we will no longer be able to support this platform with next release.

RHEL 7

Download the version 5.4.2 software for RHEL 7:

- [PKCS #11 library](#) (SHA256 checksum 8d892210db097a942ae99213c8f20fe78798840d208b0c5329baa9eda6783b9e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
68cb82352a9880d16035181bd14ee4085ff16b48b9d8d501df352dc0fb214009)
- [JCE provider](#) (SHA256 checksum 891a99f9c85182dd11c62f1745363b1fbe2ae442abbaea01c9681cd2d32ffe1c)

RHEL 8

Download the version 5.4.2 software for RHEL 8:

- [PKCS #11 library](#) (SHA256 checksum bf3ffa7bc1c4fd7bc9509e4f7c8bc3a013dbdae362cacf6ed278730fa43a0da0)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
2da903322f91f7f2ba32edf48676f4b9f23d83ceb4395ea7a51c6a0f132ddc9b)
- [JCE provider](#) (SHA256 checksum d5d1d813fdbf34e98873209606d8a55d56181598b1a539a0a0a62d4efd0addf2)

Ubuntu 18.04 LTS

Download the version 5.4.2 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum 6e78e18f64d2a6a3f9eeba1e2b846b9c91340816e2f15cd9b77fa959188716ab)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 2f52ddbd73a242ed0458747fd5b25be94328d813f862d9e4dc1c273362120744)
- [JCE provider](#) (SHA256 checksum 9d335b372cf7f9b43e9ef5838897dc1d05e8cd02b31427e0a5f763489905a187)

Windows Server 2016

Download the version 5.4.2 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum e0b4a2753f1f940614a697ec91a7777b411ffc3ae3f0d52b61e94f62f6b5149c)
- [JCE provider](#) (SHA256 checksum 0d502232118e5508e6f68a405a50f1cde0d6e7905553ecbb2c7fec6a7735c7f4)

Windows Server 2019

Download the version 5.4.2 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum e0b4a2753f1f940614a697ec91a7777b411ffc3ae3f0d52b61e94f62f6b5149c)
- [JCE provider](#) (SHA256 checksum 0d502232118e5508e6f68a405a50f1cde0d6e7905553ecbb2c7fec6a7735c7f4)

Version 5.4.2 includes improved stability and bug fixes for all SDKs. This is also the last release for the CentOS 8 platform. For more information, see the [CentOS website](#).

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Version 5.4.1

Amazon Linux

Download the version 5.4.1 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum e9a87d12874933f3bc943232c9de9efc6020f8d907272e10b461d0300375a784)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 8a8b17895e0e473b2ee3f7bd15dd9deb443f3f0af1b33c89e4c2dfa67d9f452b)
- [JCE provider](#) (SHA256 checksum 9dc53f4bcfab4a062534f82e65e2708aa60c9b965b1ec7eb61e9cae57863c7aa)

Amazon Linux 2

Download the version 5.4.1 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum 17e28ef5a0921aa4a0832aebb5c4451836f2cf238f8034d6270cd50d93b61920)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 75225f3b06a852614b5c1f657ce6b1aed6fcf87dec631072aba1764c6def1506)
- [JCE provider](#) (SHA256 checksum c71b1a314844adb623f14dcfbdbafa0a7cfa68a94021e51d48c32371d92d8e6a)

CentOS 7

Download the version 5.4.1 software for CentOS 7:

- [PKCS #11 library](#) (SHA256 checksum 17e28ef5a0921aa4a0832aebb5c4451836f2cf238f8034d6270cd50d93b61920)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 75225f3b06a852614b5c1f657ce6b1aed6fcf87dec631072aba1764c6def1506)
- [JCE provider](#) (SHA256 checksum c71b1a314844adb623f14dcfbdbafa0a7cfa68a94021e51d48c32371d92d8e6a)

CentOS 8

Download the version 5.4.1 software for CentOS 8:

- [PKCS #11 library](#) (SHA256 checksum 5a01dcf5f1be7f081a4e47b686d266fd7bc94c7a26f502eaafc6f2bc99bec2b2)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 62776a4b8a7a540c6578d747633d32a2f52c54080ba45cc450206a08a5a4a428)
- [JCE provider](#) (SHA256 checksum 1311d64d916b27c13c0890a25f6aa7f6f62ff37a8203e8bde6524bc94e661860)

Note

Due to the recent End of Life of CentOS 8, we will no longer be able to support this platform with next release.

RHEL 7

Download the version 5.4.1 software for RHEL 7:

- [PKCS #11 library](#) (SHA256 checksum 17e28ef5a0921aa4a0832aebb5c4451836f2cf238f8034d6270cd50d93b61920)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 75225f3b06a852614b5c1f657ce6b1aed6fcf87dec631072aba1764c6def1506)
- [JCE provider](#) (SHA256 checksum c71b1a314844adb623f14dcfbdbafa0a7cfa68a94021e51d48c32371d92d8e6a)

RHEL 8

Download the version 5.4.1 software for RHEL 8:

- [PKCS #11 library](#) (SHA256 checksum 5a01dcf5f1be7f081a4e47b686d266fd7bc94c7a26f502eaafc6f2bc99bec2b2)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 62776a4b8a7a540c6578d747633d32a2f52c54080ba45cc450206a08a5a4a428)
- [JCE provider](#) (SHA256 checksum 1311d64d916b27c13c0890a25f6aa7f6f62ff37a8203e8bde6524bc94e661860)

Ubuntu 18.04 LTS

Download the version 5.4.1 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum 91db61b43066545b68543d6ecf819785ab6d22379978d4fbef2488f70eab8679)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum ef9b8ba5ac3fb8a1d215e5bd36ff69d1bb7514d7a430927af4ba64caf8a8d05)
- [JCE provider](#) (SHA256 checksum 3a008934485a3c6be35479e2c14c48bc7ab9c30f5b76af272fa9c79f148c98eb)

Windows Server 2016

Download the version 5.4.1 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum f861f3170bb15c2f512d1d581179e6548cc295f3c1d45afb005b898b5cd26474)
- [JCE provider](#) (SHA256 checksum 1205867ee6a5b19f3bda7df13719edc98e6cb67b0561f4b7d775b91aa30626cf)

Windows Server 2019

Download the version 5.4.1 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum f861f3170bb15c2f512d1d581179e6548cc295f3c1d45afb005b898b5cd26474)
- [JCE provider](#) (SHA256 checksum 1205867ee6a5b19f3bda7df13719edc98e6cb67b0561f4b7d775b91aa30626cf)

Version 5.4.1 resolves a [known issue \(p. 519\)](#) with the PKCS #11 library. This is also the last release for the CentOS 8 platform. For more information, see the [CentOS website](#).

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Version 5.4.0

Amazon Linux

Download the version 5.4.0 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum 8e643e52bca4690bbe7f5c26d871fa3af067a4a1dcda7d41f57002e5a4e31795)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
be6e50291efafe50d9ffcba630367e47f71b2e4c3ae9ee3d3ab5b298eea2f72f)
- [JCE provider](#) (SHA256 checksum 0634c3fc61ce4a0c04ce161be55a4145748ccadeae04510066a984ce784b0531)

Amazon Linux 2

Download the version 5.4.0 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum 7cd01f3632c06a118e7c33c2b19e70d669ffb0fca906f6944a878d1f55fdb23a)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
a7e63134bdb3f567ab8931783e9a15602193818ed64f3a11bc1e98dd1c000324)
- [JCE provider](#) (SHA256 checksum ca6dfc9b9d96b351445320570b8e73ef761ddfc7bc6954d61b51302c6532c020)

CentOS 7

Download the version 5.4.0 software for CentOS 7:

- [PKCS #11 library](#) (SHA256 checksum 7cd01f3632c06a118e7c33c2b19e70d669ffb0fc906f6944a878d1f55fdb23a)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum a7e63134bdb3f567ab8931783e9a15602193818ed64f3a11bc1e98dd1c000324)
- [JCE provider](#) (SHA256 checksum ca6dfc9b9d96b351445320570b8e73ef761ddfc7bc6954d61b51302c6532c020)

CentOS 8

Download the version 5.4.0 software for CentOS 8:

- [PKCS #11 library](#) (SHA256 checksum 661e8a42dcdbc6e98f06a0fe8e19d88c80f5207558afe7a1dbfd8e6fb02d6871)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 536ccffd4a0e4d2a762f6eed334536e5ed8522009ad96fc980de59b463855119)
- [JCE provider](#) (SHA256 checksum 2d8c64a301e585f00bb52676e240ceaa415a0dad63dc392428d8dfc36c552165)

RHEL 7

Download the version 5.4.0 software for RHEL 7:

- [PKCS #11 library](#) (SHA256 checksum 7cd01f3632c06a118e7c33c2b19e70d669ffb0fc906f6944a878d1f55fdb23a)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum a7e63134bdb3f567ab8931783e9a15602193818ed64f3a11bc1e98dd1c000324)
- [JCE provider](#) (SHA256 checksum ca6dfc9b9d96b351445320570b8e73ef761ddfc7bc6954d61b51302c6532c020)

RHEL 8

Download the version 5.4.0 software for RHEL 8:

- [PKCS #11 library](#) (SHA256 checksum 661e8a42dcdbc6e98f06a0fe8e19d88c80f5207558afe7a1dbfd8e6fb02d6871)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 536ccffd4a0e4d2a762f6eed334536e5ed8522009ad96fc980de59b463855119)
- [JCE provider](#) (SHA256 checksum 2d8c64a301e585f00bb52676e240ceaa415a0dad63dc392428d8dfc36c552165)

Ubuntu 18.04 LTS

Download the version 5.4.0 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum a477dd926049663cbda3d7a534eda360023f54a149aa3d8308e34b575b87e70d)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum d04823a20ecb4c26135b9587f33a219fe9148e3850fe85e9fd43c818ea8ee75e)
- [JCE provider](#) (SHA256 checksum 7101b3fc98760a3110021b9c2e168685f7bdb6acd624c023d350f7af1252ca86)

Windows Server 2016

Download the version 5.4.0 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum 08607f7f7690946df7c1eb7f8b5012c5383c12afe1cbfdd5db2a326efcb13709)
- [JCE provider](#) (SHA256 checksum fa4052f6aa2865a410d3582f8a1d76e90ff068e8950067cb6fb962a38a0432b6)

Windows Server 2019

Download the version 5.4.0 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum 08607f7f7690946df7c1eb7f8b5012c5383c12afe1cbfdd5db2a326efcb13709)
- [JCE provider](#) (SHA256 checksum fa4052f6aa2865a410d3582f8a1d76e90ff068e8950067cb6fb962a38a0432b6)

Version 5.4.0 adds initial support for the JCE provider for all platforms. The JCE provider is compatible with OpenJDK 8.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- **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.
- DESede (3DES)
- HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support.

- **Ciphers (encrypt and decrypt only)**

- AES/GCM/NoPadding
- AES/ECB/NoPadding
- AES/CBC/PKCS5Padding
- DESede/ECB/NoPadding
- DESede/CBC/PKCS5Padding
- AES/CTR/NoPadding
- RSA/ECB/PKCS1Padding
- RSA/ECB/OAEP Padding
- RSA/ECB/OAEPWithSHA-1ANDMGF1Padding
- RSA/ECB/OAEPWithSHA-224ANDMGF1Padding
- RSA/ECB/OAEPWithSHA-256ANDMGF1Padding
- RSA/ECB/OAEPWithSHA-384ANDMGF1Padding
- RSA/ECB/OAEPWithSHA-512ANDMGF1Padding

- **Digests**

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

- **Sign/Verify**

- NONEwithRSA
- SHA1withRSA
- SHA224withRSA

- SHA256withRSA
- SHA384withRSA
- SHA512withRSA
- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA
- Integration with the Java KeyStore

Version 5.3.0

Amazon Linux

Download the version 5.3.0 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum 66f34388dd6188b33d3b41c9f520be4504ad4f5a5eee758ba53106e9f3a264be)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 33e5245592b43157f0658472e10c541f0e4fb7342bd4e6c221a5472663face81)

Amazon Linux 2

Download the version 5.3.0 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum a66ccb515a2155977093dd15577625adc3d57be2b7345714711a541d14eb31ee)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 14fa0428df7cb290b7e3f9ee3f3eb2d1345672f362ed25da01cdbedad8cf1511)

CentOS 7

Download the version 5.3.0 software for CentOS 7:

- [PKCS #11 library](#) (SHA256 checksum a66ccb515a2155977093dd15577625adc3d57be2b7345714711a541d14eb31ee)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 14fa0428df7cb290b7e3f9ee3f3eb2d1345672f362ed25da01cdbedad8cf1511)

CentOS 8

Download the version 5.3.0 software for CentOS 8:

- [PKCS #11 library](#) (SHA256 checksum f44a0451cdcdf87240a23aab4bf18fc0d32a2edd0806b1d169b831523f762b4c)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 71173b74a9c20364f5aff64553d0eb6984dc91ec84c8f6a6b7ff2ceca0db27c3)

RHEL 7

Download the version 5.3.0 software for RHEL 7:

- [PKCS #11 library](#) (SHA256 checksum a66ccb515a2155977093dd15577625adc3d57be2b7345714711a541d14eb31ee)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 14fa0428df7cb290b7e3f9ee3f3eb2d1345672f362ed25da01cdbedad8cf1511)

RHEL 8

Download the version 5.3.0 software for RHEL 8:

- [PKCS #11 library](#) (SHA256 checksum f44a0451cdcdf87240a23aab4bf18fc0d32a2edd0806b1d169b831523f762b4c)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 71173b74a9c20364f5aff64553d0eb6984dc91ec84c8f6a6b7ff2ceca0db27c3)

Ubuntu 18.04 LTS

Download the version 5.3.0 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum dc1b378f0cd554f588ed783cf0dd7b209352347f8a888fa63e24554f27bac91a)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 9a8373497882ccc325ee4a533b1e7bcfdfe5ad43eb6b11f2d104a3b41303a90a)

Windows Server 2016

Download the version 5.3.0 software for Windows Server 2016: [PKCS #11 library](#) (SHA256 checksum 1d262717746aa1ed33db380b78ac0bab8bbb43a444425f4518c09b4288f7f4de)

Windows Server 2019

Download the version 5.3.0 software for Windows Server 2019: [PKCS #11 library](#) (SHA256 checksum 1d262717746aa1ed33db380b78ac0bab8bbb43a444425f4518c09b4288f7f4de)

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Add support for ECDSA sign/verify with curves P-256, P-384, and secp256k1.
- Add support for the platforms: Amazon Linux, Amazon Linux 2, Centos 7.8+, RHEL 7.8+.
- Add support for OpenSSL version 1.0.2.
- Improved stability and bug fixes.

Version 5.2.1

Amazon Linux

Download the version 5.2.1 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum 710654ef82794e2cdab49ae621ac83b64c23b38b2d935e2f8d04a311994730f5)

Amazon Linux 2

Download the version 5.2.1 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum b3ac4b0d4a27d58a3ae3df45702c2c1197daf0a5703fb8d403813019451a9f36)

CentOS 7.8+

Download the version 5.2.1 software for CentOS 7.8+:

- [PKCS #11 library](#) (SHA256 checksum b3ac4b0d4a27d58a3ae3df45702c2c1197daf0a5703fb8d403813019451a9f36)

CentOS 8.3+

Download the version 5.2.1 software for CentOS 8.3+:

- [PKCS #11 library](#) (SHA256 checksum 57909c25f0c93c2af8b78a833aec7acdd09d7dc448c268b89893d8a03c7b9a45)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum e9d3daecd68f66acaadc6036bd5aa108a0c59cb91d3f31ee9b758c5590812086)

RHEL 7.8+

Download the version 5.2.1 software for RedHat Enterprise Linux 7.8+:

- [PKCS #11 library](#) (SHA256 checksum b3ac4b0d4a27d58a3ae3df45702c2c1197daf0a5703fb8d403813019451a9f36)

RHEL 8.3+

Download the version 5.2.1 software for RedHat Enterprise Linux 8.3+:

- [PKCS #11 library](#) (SHA256 checksum 57909c25f0c93c2af8b78a833aec7acdd09d7dc448c268b89893d8a03c7b9a45)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum e9d3daecd68f66acaadc6036bd5aa108a0c59cb91d3f31ee9b758c5590812086)

Ubuntu 18.04 LTS

Download the version 5.2.1 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum 29ab3ea6fc9ff84ce94f5e4e79c254190e4a6d0af5d4c5583416a892998019f0)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 55e48b58358e6a2e701ba42dd0fd29bcae57d08b0af77a3f44c62b397edd8b9a)

Windows Server 2016

Download the latest version 5.2.1 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum b82c2afa1e8353fb47fa392ba20be480e0d2ccb0e361acb6d8ca89f8f8803545)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Windows Server 2019

Download the latest version 5.2.1 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum b82c2afa1e8353fb47fa392ba20be480e0d2ccb0e361acb6d8ca89f8f8803545)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.2.0

Amazon Linux

Download the version 5.2.0 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum 6ba98c1cd05e80d96e690c146c4d870f34a8971f542a3b7c3b30c96ac2bdf940)

Amazon Linux 2

Download the version 5.2.0 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum 3df6395a9d15ad3d1b1c19bae78dbac96a704be304d96f9ae9101a78f7573797)

CentOS 7.8+

Download the version 5.2.0 software for CentOS 7.8+:

- [PKCS #11 library](#) (SHA256 checksum 3df6395a9d15ad3d1b1c19bae78dbac96a704be304d96f9ae9101a78f7573797)

CentOS 8.3+

Download the version 5.2.0 software for CentOS 8.3+:

- [PKCS #11 library](#) (SHA256 checksum d1d3f8d2ec98ae7bdcc4ffc8e3f6affd4b7d11113ab8480ba2c7dd6ed17c280e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum f5d8fd0d694c481f6c51a1e9ff2c45873f31df1000db671901a26f5041f905f8)

RHEL 7.8+

Download the version 5.2.0 software for RedHat Enterprise Linux 7.8+:

- [PKCS #11 library](#) (SHA256 checksum 3df6395a9d15ad3d1b1c19bae78dbac96a704be304d96f9ae9101a78f7573797)

RHEL 8.3+

Download the version 5.2.0 software for RedHat Enterprise Linux 8.3+:

- [PKCS #11 library](#) (SHA256 checksum d1d3f8d2ec98ae7bdcc4ffc8e3f6affd4b7d11113ab8480ba2c7dd6ed17c280e)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum f5d8fd0d694c481f6c51a1e9ff2c45873f31df1000db671901a26f5041f905f8)

Ubuntu 18.04 LTS

Download the version 5.2.0 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum 25448e26a2f600ee53143779001bb001111aa37d34b861f4a88a7e507eb6ec44)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 797061b4c4a2550172ec0d49c694ca76ddefc0fb157a2ce39a5f39a650767c36)

Windows Server 2016

Download the latest version 5.2.0 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum 64c9afa1856a7166707d563ab4eedc2dc27132df7f5e76c0467ca996828fff0b)

For information about Windows Server platform support for Client SDK 5, see [??? \(p. 314\)](#).
Windows Server 2019

Download the latest version 5.2.0 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum 64c9afa1856a7166707d563ab4eedc2dc27132df7f5e76c0467ca996828fff0b)

For information about Windows Server platform support for Client SDK 5, see [??? \(p. 314\)](#).

Version 5.2.0 adds support additional key types and mechanisms to the PKCS #11 library.

PKCS #11 library

Key Types

- ECDSA– P-224, P-256, P-384, P-521 and secp256k1 curves
- Triple DES (3DES)

Mechanisms

- CKM_EC_KEY_PAIR_GEN
- CKM_DES3_KEY_GEN
- CKM_DES3_CBC
- CKM_DES3_CBC_PAD
- CKM_DES3_ECB
- CKM_ECDSA
- CKM_ECDSA_SHA1
- CKM_ECDSA_SHA224
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384
- CKM_ECDSA_SHA512
- CKM_RSA_PKCS for Encrypt/Decrypt

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.1.0

Amazon Linux

Download the version 5.1.0 software for Amazon Linux:

- [PKCS #11 library](#) (SHA256 checksum cd9016efeb1d7339be1fda4cff0d32f9144a119077da9f409f7e6f27c1d54c8b)

Amazon Linux 2

Download the version 5.1.0 software for Amazon Linux 2:

- [PKCS #11 library](#) (SHA256 checksum 9674d705032b39087a8ddaa793647fa0e31968c3ede3ca67f3ea65be4f0d77a1)

CentOS 7.8+

Download the version 5.1.0 software for CentOS 7.8+:

- [PKCS #11 library](#) (SHA256 checksum 9674d705032b39087a8ddaa793647fa0e31968c3ede3ca67f3ea65be4f0d77a1)

CentOS 8.3+

Download the version 5.1.0 software for CentOS 8.3+:

- [PKCS #11 library](#) (SHA256 checksum 0c0de23d884500b47cf0df89943f902c5a52cb48a6088693c51e31a240bc0bc3)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum fd2ff8f5fca5ed3d92ff602c6673e8b92daa70d904c7428dbe90ea6a7b5492cdb)

RHEL 7.8+

Download the version 5.1.0 software for RedHat Enterprise Linux 7.8+:

- [PKCS #11 library](#) (SHA256 checksum 9674d705032b39087a8ddaa793647fa0e31968c3ede3ca67f3ea65be4f0d77a1)

RHEL 8.3+

Download the version 5.1.0 software for RedHat Enterprise Linux 8.3+:

- [PKCS #11 library](#) (SHA256 checksum 0c0de23d884500b47cf0df89943f902c5a52cb48a6088693c51e31a240bc0bc3)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum fd2ff8f5fca5ed3d92ff602c6673e8b92daa70d904c7428dbe90ea6a7b5492cdb)

Ubuntu 18.04 LTS

Download the version 5.1.0 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (SHA256 checksum f03e683f37fe82209451b95704d42716c1e6155611c6c02e7838e5e41c429019)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 956b51bb5a20a302c938c8ad29542a487b2c85fe7a7c9e3386f7d280d6913058)

Windows Server 2016

Download the latest version 5.1.0 software for Windows Server 2016:

- [PKCS #11 library](#) (SHA256 checksum 520c9cd19fc48dcf61b2e3f2d1951cefa9ba5e41874a9db7c926a04e03147c8d)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Windows Server 2019

Download the latest version 5.1.0 software for Windows Server 2019:

- [PKCS #11 library](#) (SHA256 checksum 520c9cd19fc48dcf61b2e3f2d1951cefa9ba5e41874a9db7c926a04e03147c8d)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Version 5.1.0 adds support for additional mechanisms to the PKCS #11 library.

PKCS #11 library

Mechanisms

- CKM_RSA_PKCS for Wrap/Unwrap
- CKM_RSA_PKCS_PSS
- CKM_SHA1_RSA_PKCS_PSS
- CKM_SHA224_RSA_PKCS_PSS
- CKM_SHA256_RSA_PKCS_PSS
- CKM_SHA384_RSA_PKCS_PSS
- CKM_SHA512_RSA_PKCS_PSS
- CKM_AES_ECB
- CKM_AES_CTR
- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_SP800_108_COUNTER_KDF
- CKM_GENERIC_SECRET_KEY_GEN
- CKM_SHA_1_HMAC
- CKM_SHA224_HMAC
- CKM_SHA256_HMAC
- CKM_SHA384_HMAC
- CKM_SHA512_HMAC
- CKM_RSA_PKCS_OAEP Wrap/Unwrap only
- CKM_RSA_AES_KEY_WRAP
- CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD

API Operations

- C_CreateObject
- C_DeriveKey
- C_WrapKey
- C_UnWrapKey

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.0.1

Amazon Linux

Download the version 5.0.1 software for Amazon Linux:

- [PKCS #11 library](#) (introductory support)

Amazon Linux 2

Download the version 5.0.1 software for Amazon Linux 2:

- [PKCS #11 library](#) (introductory support)

CentOS 7.8+

Download the version 5.0.1 software for CentOS 7.8+:

- [PKCS #11 library](#) (introductory support)

CentOS 8.3+

Download the version 5.0.1 software for CentOS 8.3+:

- [PKCS #11 library](#) (introductory support)
- [OpenSSL Dynamic Engine](#) (introductory support)

RHEL 7.8+

Download the version 5.0.1 software for RedHat Enterprise Linux 7.8+:

- [PKCS #11 library](#) (introductory support)

RHEL 8.3+

Download the version 5.0.1 software for RedHat Enterprise Linux 8.3+:

- [PKCS #11 library](#) (introductory support)
- [OpenSSL Dynamic Engine](#) (introductory support)

Ubuntu 18.04 LTS

Download the version 5.0.1 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#) (introductory support)
- [OpenSSL Dynamic Engine](#) (introductory support)

Windows Server 2016

Download the latest version 5.0.1 software for Windows Server 2016:

- [PKCS #11 library](#) (introductory support)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Windows Server 2019

Download the latest version 5.0.1 software for Windows Server 2019:

- [PKCS #11 library](#) (introductory support)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Version 5.0.1 adds initial support for OpenSSL Dynamic Engine.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Initial release of OpenSSL Dynamic Engine.
- This release offers introductory support for key types and OpenSSL APIs:
 - RSA key generation for 2048, 3072, and 4096-bit keys
 - OpenSSL APIs:
 - [RSA Sign](#) using RSA PKCS with SHA1/224/256/384/512 & RSA PSS
 - [RSA Key Generation](#)

For more information, see [OpenSSL Dynamic Engine \(p. 347\)](#).

- Platforms supported: CentOS 8.3+, Red Hat Enterprise Linux (RHEL) 8.3+, and Ubuntu 18.04 LTS
 - Requires: OpenSSL 1.1.1

For more information, see [Supported Platforms \(p. 314\)](#).

- Support for SSL/TLS Offload on CentOS 8.3+, Red Hat Enterprise Linux (RHEL) 8.3, and Ubuntu 18.04 LTS, including NGINX 1.19 (for select cipher suites).

For more information, see [Using SSL/TLS Offload on Linux \(p. 393\)](#).

Version 5.0.0

Amazon Linux

Download the version 5.0.0 software for Amazon Linux:

- [PKCS #11 library](#)

Amazon Linux 2

Download the version 5.0.0 software for Amazon Linux 2:

- [PKCS #11 library](#)

CentOS 7.8+

Download the version 5.0.0 software for CentOS 7.8+:

- [PKCS #11 library](#)

CentOS 8.3+

Download the version 5.0.0 software for CentOS 8.2:

- [PKCS #11 library](#)

RHEL 7.8+

Download the version 5.0.0 software for RedHat Enterprise Linux 7.8+:

- [PKCS #11 library](#)

RHEL 8.3+

Download the version 5.0.0 software for RedHat Enterprise Linux 8.2:

- [PKCS #11 library](#)

Ubuntu 18.04 LTS

Download the version 5.0.0 software for Ubuntu 18.04 LTS:

- [PKCS #11 library](#)

Windows Server 2016

Download the latest version 5.0.0 software for Windows Server 2016:

- [PKCS #11 library](#)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Windows Server 2019

Download the latest version 5.0.0 software for Windows Server 2019:

- [PKCS #11 library](#)

For information about Windows Server platform support for Client SDK 5, see [???](#) (p. 314).

Version 5.0.0 is the first release.

PKCS #11 library

- This is the initial release.

[Introductory PKCS #11 library support in client SDK version 5.0.0](#)

This section details support for key types, mechanisms, API operations and attributes Client SDK version 5.0.0.

Key Types:

- **AES**– 128, 192, and 256-bit AES keys
- **RSA**– 2048-bit to 4096-bit RSA keys, in increments of 256 bits

Mechanisms:

- CKM_AES_GCM
- CKM_AES_KEY_GEN
- CKM_CLOUDHSM_AES_GCM
- CKM_RSA_PKCS
- CKM_RSA_X9_31_KEY_PAIR_GEN

- CKM_SHA1
- CKM_SHA1_RSA_PKCS
- CKM_SHA224
- CKM_SHA224_RSA_PKCS
- CKM_SHA256
- CKM_SHA256_RSA_PKCS
- CKM_SHA384
- CKM_SHA384_RSA_PKCS
- CKM_SHA512
- CKM_SHA512_RSA_PKCS

API Operations:

- C_CloseAllSessions
- C_CloseSession
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate
- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate
- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo

- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignUpdate
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyUpdate

Attributes:

- GenerateKeyPair
 - All RSA Key attributes
- GenerateKey
 - All AES Key attributes
- GetAttributeValue
 - All RSA Key attributes
 - All AES Key attributes

Samples:

- [Generate keys \(AES, RSA, EC\)](#)
- [List key attributes](#)
- [Encrypt and decrypt data with AES GCM](#)
- [Sign and verify data with RSA](#)

Previous Client SDK 3 releases

This section lists previous Client SDK 3 releases. For the current Client SDK 3 release, see [the section called “Latest releases” \(p. 261\)](#).

Version 3.4.3

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 3.4.3 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum a92337e0f1d8a7d5db89b16f7530535746af1affd609df0db7381cfeab2e04fd)
- [PKCS #11 library](#) (SHA256 checksum 499be5d60d07ea90cc70522875f9ba477e4a356f37051086c51e31d0f6196501)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 5605a34334f120afe2ecf6bcd5eb0cc30a46abe2e31eddbf48091bbd8819ebb)
- [JCE provider](#) (SHA256 checksum a3d78eb4f8195e37a1f263bc3e16f1e767a78bd88e6b9d276b1e64e6ded003f9)

- [AWS CloudHSM Management Utility](#) (SHA256 checksum
82c0dbf9eb322a2462a66105cc93dd6c3f11d59e8e9845b3d93419286130a10c)

Amazon Linux 2

Download the version 3.4.3 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum 16e3fd76c19216f270219524bdb4ab82a0ccdbec04175b4a7e34615cadd741ac)
- [PKCS #11 library](#) (SHA256 checksum 0f9ac194bc3fcfb615e4735d87e6174f4630d776f2211cbe9d5e2cc90631095f)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
ca0007b6b0d05f0bb6beaebc9bf39c2feefc1cd51437d46c9db2781046d7353)
- [JCE provider](#) (SHA256 checksum 0109c40792414beae32e78f6100024d8db79cf1216fdfba36b5f2bd458fb6610)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
045af33133c0a809c45876fdc45f81234613f1cb718e10b138df0f6b08427365)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.4.3.

Use [the section called "Version 3.2.1" \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.4.3 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 16e3fd76c19216f270219524bdb4ab82a0ccdbec04175b4a7e34615cadd741ac)
- [PKCS #11 library](#) (SHA256 checksum 0f9ac194bc3fcfb615e4735d87e6174f4630d776f2211cbe9d5e2cc90631095f)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
ca0007b6b0d05f0bb6beaebc9bf39c2feefc1cd51437d46c9db2781046d7353)
- [JCE provider](#) (SHA256 checksum 0109c40792414beae32e78f6100024d8db79cf1216fdfba36b5f2bd458fb6610)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
045af33133c0a809c45876fdc45f81234613f1cb718e10b138df0f6b08427365)

CentOS 8

Download the version 3.4.3 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 5436a0b1c2df34ccf846c79a5479a986e125d8b04234a2b1fa38db19bb9ba0b8)
- [PKCS #11 library](#) (SHA256 checksum 9d5abe25513294ede35677f0c896547fc7671e19dbf6514066161d933b84fed1)
- [JCE provider](#) (SHA256 checksum f04e48e47cfdbccca47bb45f988c95832244b2ab18b3c040249c91ad2a99ab64)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
da50c29bbeb65f84d9b4769bf3539a9b5189c51c7e0ae8981b85290aa4f5d152)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.4.3.

Use [the section called "Version 3.2.1" \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.4.3 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 16e3fd76c19216f270219524bdb4ab82a0ccdbec04175b4a7e34615cadd741ac)
- [PKCS #11 library](#) (SHA256 checksum 0f9ac194bc3fcfb615e4735d87e6174f4630d776f2211cbe9d5e2cc90631095f)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum ca0007b6b0d05f0bb6beaebc9bf39c2feefc1cd51437d46c9db2781046d7353)
- [JCE provider](#) (SHA256 checksum 0109c40792414beae32e78f6100024d8db79cf1216fdfba36b5f2bd458fb6610)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 045af33133c0a809c45876fdc45f81234613f1cb718e10b138df0f6b08427365)

RHEL 8

Download the version 3.4.3 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 5436a0b1c2df34ccf846c79a5479a986e125d8b04234a2b1fa38db19bb9ba0b8)
- [PKCS #11 library](#) (SHA256 checksum 9d5abe25513294ede35677f0c896547fc7671e19dbf6514066161d933b84fed1)
- [JCE provider](#) (SHA256 checksum f04e48e47cfdbcce47bb45f988c95832244b2ab18b3c040249c91ad2a99ab64)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum da50c29bbeb65f84d9b4769bf2539a9b5189c51c7e0ae8981b85290aa4f5d152)

Ubuntu 16.04 LTS

Download the version 3.4.3 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum ab6cf9173c4023f0766f2b67e66d507ccd597302576fc9507c23734560f67f9)
- [PKCS #11 library](#) (SHA256 checksum 71a721747a6c593d3b1061879dbbf891640dab2909ca9efdef4dad3b66013863)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 20b8a764e341a2d2b6e4a7197e5ff32ec9e215fc5314d309f14fdfd1e8f6e986)
- [JCE provider](#) (SHA256 checksum 6deb8ad0c7950f558302a3d9828e7f7a1ed7faba7148f738e1c8dd324b631163)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 0a74107394bca140dfab4abd20dbc486c5969df7de5ee51b610fb787398531d2)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.4.3 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 03d8b339494c5b6d7f84fb96225f64f2a10e60070aab9f9bc9c7e04bb7b5916)
- [PKCS #11 library](#) (SHA256 checksum 8fffc5b2988bb2845476117ef881b4a62785fbcd2a383cd656765921d552eac)
- [JCE provider](#) (SHA256 checksum b6a103606fcf54e85ac98da257eef872139148727cb5734c43e55d6952c547e4)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 5d611d31132af9e667027a1143b9d3e744ef701be0f46609777c31b8658db3b1)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.4.3 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.4.3) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum 56b32ed4d4c26b3a4e5476cb83defe6b5db0881954d00ec859fb1d7122a7d07d)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum fe2cdcae7b83a6add0d56bac1ac467767e80520d323b78b6012c8f05e8a903cc1)

Version 3.4.3 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.17.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.2

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 3.4.2 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum 5fc89ace8bb30f02ecaed54edb125157d4c5da48016428d08212bccb4a56fc6)
- [PKCS #11 library](#) (SHA256 checksum d85b2e769bcc96efa86b1bfc7bac22ddeb1b237ad6bab6b7402b61c9db638578)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 8c36fd8a8701b738294ce9d6b612a366f1b139d4bdb8f52b3bf7c5bfc8d391c4)
- [JCE provider](#) (SHA256 checksum 589d744275c825050248909c67ee756fc3d3f35ff8932d80a4a981f4b275b981)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum ba55f0dee3bf728f960bc4182f8178de63a2f17363c830a7480d07ff27872f44)

Amazon Linux 2

Download the version 3.4.2 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum 8110f2d5f3ec0dbdc1f7c49511610ad678b0a27823edcba2e3c4c79f6859aa8f)
- [PKCS #11 library](#) (SHA256 checksum 2c2225f29a30347810b0bdef617036b6c4ba698955bd5cb1c588e8230a7dbd24)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 3b07f1588e1093c6b67f2c8810239679b9a42ac9472a065bf97fd4d75a4a9f28)

- [JCE provider](#) (SHA256 checksum ecaa9ea834ef51c58ab3601248d28264a23173b19f10180640c3cf1addb1a808)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2ee1625ab2634e96a2bf5fea3df87d085991a4bf404e9beea993c58de2902db2)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.4.2.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.4.2 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 8110f2d5f3ec0dbdc1f7c49511610ad678b0a27823edcba2e3c4c79f6859aa8f)
- [PKCS #11 library](#) (SHA256 checksum 2c2225f29a30347810b0bdef617036b6c4ba698955bd5cb1c588e8230a7dbd24)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 3b07f1588e1093c6b67f2c8810239679b9a42ac9472a065bf97fd4d75a4a9f28)
- [JCE provider](#) (SHA256 checksum ecaa9ea834ef51c58ab3601248d28264a23173b19f10180640c3cf1addb1a808)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2ee1625ab2634e96a2bf5fea3df87d085991a4bf404e9beea993c58de2902db2)

CentOS 8

Download the version 3.4.2 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 1a1f94924acb5d0778f40d5af827e270e67074744ecc2ff84eb44f3f2bb3cb71)
- [PKCS #11 library](#) (SHA256 checksum 922ff0f24ee129f57bddbd7a757521d1117e4e910f17047c674b395121760fe)
- [JCE provider](#) (SHA256 checksum e68dd76218eb476b235d7fd1aa911b5c8567511bd9787ed45744d6d146c519ac)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum f9a8c504c6c8b3df10d098dc299a921158ca820c0d015a167b1498036c33b0bd)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.4.2.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.4.2 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 8110f2d5f3ec0dbdc1f7c49511610ad678b0a27823edcba2e3c4c79f6859aa8f)
- [PKCS #11 library](#) (SHA256 checksum 2c2225f29a30347810b0bdef617036b6c4ba698955bd5cb1c588e8230a7dbd24)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 3b07f1588e1093c6b67f2c8810239679b9a42ac9472a065bf97fd4d75a4a9f28)
- [JCE provider](#) (SHA256 checksum ecaa9ea834ef51c58ab3601248d28264a23173b19f10180640c3cf1addb1a808)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2ee1625ab2634e96a2bf5fea3df87d085991a4bf404e9beea993c58de2902db2)

RHEL 8

Download the version 3.4.2 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 1a1f94924acb5d0778f40d5af827e270e67074744ecc2ff84eb44f3f2bb3cb71)
- [PKCS #11 library](#) (SHA256 checksum 922ff0f24ee129f57bddbd7a757521d1117e4e910f17047c674b395121760fe)
- [JCE provider](#) (SHA256 checksum e68dd76218eb476b235d7fd1aa911b5c8567511bd9787ed45744d6d146c519ac)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum f9a8c504c6c8b3df10d098dc299a921158ca820c0d015a167b1498036c33b0bd)

Ubuntu 16.04 LTS

Download the version 3.4.2 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum f8e20b397cc5fa78c400cc59b9b4fdfa231b6df310a554e77bb6002c9c43259e)
- [PKCS #11 library](#) (SHA256 checksum f9f7e86190d1e051a837028f103315ed50461cbd7c9d249fac9e5fc8601b6381)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 827d686f7bd961c70a35b262a7003edec5d4b05f346eb9c51c05f585c15543ee)
- [JCE provider](#) (SHA256 checksum 74ce4fba33f6b1f4a79a6d62ace011b215904483397b676e2740a094c058d76d)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 35bbcf5be6b1ab68da11db61c28e11b4bee0d7cdb03be6430d7ee77a8d23ae28)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.4.2 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 555c659973e24e7dc788c80e97fae982844b14f34381a80b212991141a24fe6)
- [PKCS #11 library](#) (SHA256 checksum 7dc31b5db1b0f75ea2ed1f9f14dc62198a0d2feb08278183b958b125fcf969a)
- [JCE provider](#) (SHA256 checksum 67893de3e86202bb73bd9541b5a7c6afa53c6fd030aab321af4a384a0f2e690)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum cc79e50155bd1bb2f395a2538bd0e768292f83b3fc0fa9237a46fb745f4112c1)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.4.2 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.4.2) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum 551d0b584aa3200f97c6c5c8ae64d43ba2922fd2ed1feb60ea8b235909fad873)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum b06695e6da332771b097f637d27f3c82b8c7e86e1976a5817e61e7d2ba8d8013)

Version 3.4.2 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.16.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.1

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 3.4.1 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum ca038e1219d6999ed62b680fde18fe0b992af082eaf9360f351a21ee2d42f786)
- [PKCS #11 library](#) (SHA256 checksum 6f532aba4e15a802befea15c69874bc0e01c8076b98c7b43e4722f220912d4f8)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum b26afb7b84d95ac1d5a0d1a0de53c48dda6ddd52306b0d3f72d4927647efc06)
- [JCE provider](#) (SHA256 checksum 76fa2c963cb23cea11644e03151a33f6fad249e872ec7609af6ab1f54371a4d5)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum a1cdd92cb4ce229aa33905f33ac9e65f147acc2e21b672453c0e87a2ba6d1896)

Amazon Linux 2

Download the version 3.4.1 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum dff54434429e24100c66f78aba4ca44a6c44d0017b9efe533399065bd13efbf8)
- [PKCS #11 library](#) (SHA256 checksum ca6a9b3f42d9f765992906380638099a04dd8b5f2d3d24e2b616b2cf2f836fdd)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum b14c1718b3d68205d7447bf467bdaa0c21b8bc0e6cd25c7065ffff006f270807)
- [JCE provider](#) (SHA256 checksum 55327fc3feac4f1ccaa0eafc49d77c9f47d07e18b4332fed9f559c11c97e769)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2e830f22bee0ceea1273c512f14d437e321f71616f619aa0b5d7490374648573)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.4.1.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.4.1 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum dff54434429e24100c66f78aba4ca44a6c44d0017b9efe533399065bd13efbf8)

- [PKCS #11 library](#) (SHA256 checksum ca6a9b3f42d9f765992906380638099a04dd8b5f2d3d24e2b616b2cf2f836fd)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum b14c1718b3d68205d7447bf467bdaa0c21b8bc0e6cd25c7065ffff006f270807)
- [JCE provider](#) (SHA256 checksum 55327fc3feac4f1ccaa0eafc49d77c9f47d07e18b4332fed9f559c11c97e769)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2e830f22bee0ceea1273c512f14d437e321f71616f619aa0b5d7490374648573)

CentOS 8

Download the version 3.4.1 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum cd7770e2ada3f885250b9c18f23cb8fdef576be6b79ee5293e1f670f6ab9aac4)
- [PKCS #11 library](#) (SHA256 checksum 3ecbfce9a614c61f2767d984c27296547c68894a3b838780b77cdf2b6cadb1ef)
- [JCE provider](#) (SHA256 checksum 7762c85a113f98909f89dab1eb7529f2327c84345ea05a592c491773b7b3c104)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2495720d653b332f3dae405daa6ef462e57e8424d69d75191841ed6960d2cdbe)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.4.1.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.4.1 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum dff54434429e24100c66f78aba4ca44a6c44d0017b9efe533399065bd13efbf8)
- [PKCS #11 library](#) (SHA256 checksum ca6a9b3f42d9f765992906380638099a04dd8b5f2d3d24e2b616b2cf2f836fd)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum b14c1718b3d68205d7447bf467bdaa0c21b8bc0e6cd25c7065ffff006f270807)
- [JCE provider](#) (SHA256 checksum 55327fc3feac4f1ccaa0eafc49d77c9f47d07e18b4332fed9f559c11c97e769)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2e830f22bee0ceea1273c512f14d437e321f71616f619aa0b5d7490374648573)

RHEL 8

Download the version 3.4.1 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum cd7770e2ada3f885250b9c18f23cb8fdef576be6b79ee5293e1f670f6ab9aac4)
- [PKCS #11 library](#) (SHA256 checksum 3ecbfce9a614c61f2767d984c27296547c68894a3b838780b77cdf2b6cadb1ef)
- [JCE provider](#) (SHA256 checksum 7762c85a113f98909f89dab1eb7529f2327c84345ea05a592c491773b7b3c104)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 2495720d653b332f3dae405daa6ef462e57e8424d69d75191841ed6960d2cdbe)

Ubuntu 16.04 LTS

Download the version 3.4.1 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum ff2a986ea8ec6d37b6484536bbab1e9406c6025a80f2264aefa563c50b865f8f)
- [PKCS #11 library](#) (SHA256 checksum 17665e2f31f692ffb74d809b05e2672a6d493a48d1010397c8900ada65af4c1a)

- [OpenSSL Dynamic Engine](#) (SHA256 checksum 826af655e5cf7224d84fdf0a3545966fcefb18d1fe43a64dbe85cd7589594747)
- [JCE provider](#) (SHA256 checksum 027d4f19416bb34c124443840ddf8ddb6e7b417107a5c4e2705158d93532d378)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 367272bd1840e0c8c6bce663d3f4469f3940abec9e037b455cb3113086aa598f)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.4.1 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 1759bb86f4ca8b9494b59a7cac27f540f4667a3f86745f5d507db12a217b16c1)
- [PKCS #11 library](#) (SHA256 checksum 470a2cd6d2bdb1cdc014ad3ac3e4e97867ca50888fbe64784238f9d2eff345d2)
- [JCE provider](#) (SHA256 checksum 83edeae4b22d26849b9806f3ad7d729b25cf555e06f8d0bb90e1032f1e908207)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 8aeb482106d1af6de81c89369c98516c97955876a5a645a415fe8e7c065bda66)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.4.1 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.4.1) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum 9164495d9716adb19b42a02be53b7ae2c796b3a76e592f9aa29443de34dcbbeb)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum b6192d23912feefc0cde0ff43d27479792d06a416e824a664fdd112202cae0c)

Version 3.4.1 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.15.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.0

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 3.4.0 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum e5ca8d26805ea1d11da3d8801e706423d766071a7ede0bedfcae45e561f45d17)
- [PKCS #11 library](#) (SHA256 checksum 3851c30ca0d426dc054000ad6929016fc783043404edf98a4033fd9b64d29f6c)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum ae19c864fe4b0a96e9d8e3c03a41b84b2f0d4a91ec65485d34624eb964fc90c)
- [JCE provider](#) (SHA256 checksum 3251d7c11528c5eda386c9c250ad9178acf303832136c6b267a0b9f825436c4b)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 9f87b4b7bdb219df05905c132a732ffcb2544a3c9a68af0859c6434d5c7747e7)

Amazon Linux 2

Download the version 3.4.0 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum 4b520de05217f7e6077bd94af6788da60e19d1b3f28e5a17669232d519c83857)
- [PKCS #11 library](#) (SHA256 checksum 64262f715786172c7c0da0ab74136097c2b7a1641e3b284ae827f1486fbcc56b)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 05320772c49a622bda0284dd503c8ae5a13c1669f0e4b475873e53b5a7c074e6)
- [JCE provider](#) (SHA256 checksum 845b2788a654b81a5bbaf19cefd7865ceab8a7c779d927eb282eea86b0819007)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum ce6741813d29a41cfb23722fb0d140a2fedf90a44a0ddcac39a607457eabe91a)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.4.0.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.4.0 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 4b520de05217f7e6077bd94af6788da60e19d1b3f28e5a17669232d519c83857)
- [PKCS #11 library](#) (SHA256 checksum 64262f715786172c7c0da0ab74136097c2b7a1641e3b284ae827f1486fbcc56b)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 05320772c49a622bda0284dd503c8ae5a13c1669f0e4b475873e53b5a7c074e6)
- [JCE provider](#) (SHA256 checksum 845b2788a654b81a5bbaf19cefd7865ceab8a7c779d927eb282eea86b0819007)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum ce6741813d29a41cfb23722fb0d140a2fedf90a44a0ddcac39a607457eabe91a)

CentOS 8

Download the version 3.4.0 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 853b05e6ea6e239f42e1bbf70be26adcc8205d2d172dfc3d57342c14d0060a1e)
- [PKCS #11 library](#) (SHA256 checksum 51415be53ee10ddc8e85d5bcb0f052a4e29c086434c7fac152b57c7ac37bc3f5)
- [JCE provider](#) (SHA256 checksum e7a39e46084cab6e193c13c57ea021f0570246cce90b21863d6b40f60a7a8cd7)

- [AWS CloudHSM Management Utility](#) (SHA256 checksum
5de8d9d9a88deae2ffacd4923e429aad885d600adeb1d0bdb771da177fae647)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.4.0.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.4.0 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 4b520de05217f7e6077bd94af6788da60e19d1b3f28e5a17669232d519c83857)
- [PKCS #11 library](#) (SHA256 checksum 64262f715786172c7c0da0ab74136097c2b7a1641e3b284ae827f1486fbcb56b)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
05320772c49a622bda0284dd503c8ae5a13c1669f0e4b475873e53b5a7c074e6)
- [JCE provider](#) (SHA256 checksum 845b2788a654b81a5bbaf19cefd7865ceab8a7c779d927eb282eea86b0819007)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
ce6741813d29a41cfb23722fb0d140a2fedf90a44a0ddcac39a607457eabe91a)

RHEL 8

Download the version 3.4.0 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 853b05e6ea6e239f42e1bbf70be26adcc8205d2d172dfc3d57342c14d0060a1e)
- [PKCS #11 library](#) (SHA256 checksum 51415be53ee10ddc8e85d5bc0f052a4e29c086434c7fac152b57c7ac37bc3f5)
- [JCE provider](#) (SHA256 checksum e7a39e46084cab6e193c13c57ea021f0570246cce90b21863d6b40f60a7a8cd7)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
5de8d9d9a88deae2ffacd4923e429aad885d600adeb1d0bdb771da177fae647)

Ubuntu 16.04 LTS

Download the version 3.4.0 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 1f80e1a7e2fc35481cc4a6e7fba3869e863becbc09aba32ef2a81a2494c2e49)
- [PKCS #11 library](#) (SHA256 checksum 8fb002f8d5810ee43b8ef020831372a3a9d0f5a7fa35dca23f7d93a2a74a63bf)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
2c1717f99cb4a56d47d8517ba97847c644112d0a4f37da435898f77cc794a508)
- [JCE provider](#) (SHA256 checksum 9f0708e2ec644f5b87dbd6fb0683f690e78371f3ef866d1384ab80e3b4a1c1a6)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
fec730e64467371fbe5b3b8485215712637fe6139fc45832095fb945fb4317d1)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.4.0 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum a78832a1666b41a85869fc0362c19ecc33113243970a4cc88eeae993c4cd47c1)
- [PKCS #11 library](#) (SHA256 checksum f9610a82d55b17202c2ad064650199abbc2f8412fe28ef24819a899633feeaa80)

- [JCE provider](#) (SHA256 checksum 5b450c1519594b9630f06620c6e23079dcbafe2999852ea95768c699461585eb)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum 8bdc208a258976c5cb5fa97bcd19a3a5cb156ecfb507a019c99bf5886b5f03f)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.4.0 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.4.0) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum 95dea3a5df195e7b4ce55f1870af9402c49655f42feba8e9d6bf131cdc990c6a)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum a329a47a63a8ecd110ee59b686b217b9c154903d7388f15c75bb79506f014714)

Version 3.4.0 adds updates to all components.

AWS CloudHSM Client Software

- Improved stability and bug fixes.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.3.2

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 3.3.2 software for Amazon Linux:

- [AWS CloudHSM Client](#) (SHA256 checksum 1f73c4e86fff4c8a3b465f6d21bd81c7c767267476f24c29e45ebab7f470f9a8)
- [PKCS #11 library](#) (SHA256 checksum d9a4333bddbc7807a34806ff46a63802bc9f4a021358230f1c8292357cd8f43a)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum 16153d539667b16905bd0ec0de3d023ce8af6bddf4eb03eba8f945152322e2e6)
- [JCE provider](#) (SHA256 checksum 568324f6484049156026e903b886195c1f2e46efa338bb5d5a0a5501a77148aa)

- [AWS CloudHSM Management Utility](#) (SHA256 checksum
df9c833de5c828b0de11b3ef93ec41988a7c17bc3950d772c7dc9674876d745f)

Amazon Linux 2

Download the version 3.3.2 software for Amazon Linux 2:

- [AWS CloudHSM Client](#) (SHA256 checksum 2cd5e0b022fe9091e027f019be1ea81923392ca6d065bfcca6532aa5b9492a99)
- [PKCS #11 library](#) (SHA256 checksum ae062f4675f7547639a8696d39cec1afc09db0b9fa6ded6e335b2a81025ab993)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
a0876a42f802c0fb67e5301045c435a4a9494ed84110b87c8fc3524e9afc29a)
- [JCE provider](#) (SHA256 checksum 640c7e3e43ca27178c003ca153a90814f7c78ada3e86a4aa4ce663bb784c4b8d)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
cf76cf044b01d9168a408d78aedae79295626bc4b6eb040d82663c5d8d814f6e)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.3.2.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.3.2 software for CentOS 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 2cd5e0b022fe9091e027f019be1ea81923392ca6d065bfcca6532aa5b9492a99)
- [PKCS #11 library](#) (SHA256 checksum ae062f4675f7547639a8696d39cec1afc09db0b9fa6ded6e335b2a81025ab993)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
a0876a42f802c0fb67e5301045c435a4a9494ed84110b87c8fc3524e9afc29a)
- [JCE provider](#) (SHA256 checksum 640c7e3e43ca27178c003ca153a90814f7c78ada3e86a4aa4ce663bb784c4b8d)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
cf76cf044b01d9168a408d78aedae79295626bc4b6eb040d82663c5d8d814f6e)

CentOS 8

Download the version 3.3.2 software for CentOS 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 696bb3d67b3aca379106a409a8de814174df5b8308a2d4500bee5fc89f40070)
- [PKCS #11 library](#) (SHA256 checksum c58992e14d75c0cc7ae9f57746b40be5a4dbc1f2769e9d387eae39107b560749)
- [JCE provider](#) (SHA256 checksum 95e519d2bf656446141cd227e50447b67d485008e8b38151ea31f2a9ca855b49)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
e1ab86404d162e1169bb80364510119ce2c072fe30fbebe0a06bd2497f980f840)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.3.2.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.3.2 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#) (SHA256 checksum 2cd5e0b022fe9091e027f019be1ea81923392ca6d065bfcca6532aa5b9492a99)
- [PKCS #11 library](#) (SHA256 checksum ae062f4675f7547639a8696d39cec1afc09db0b9fa6ded6e335b2a81025ab993)

- [OpenSSL Dynamic Engine](#) (SHA256 checksum
a0876a42f802c0fb67e5301045c435a4a9494ed84110b87c8fc3524e9afc29a)
- [JCE provider](#) (SHA256 checksum 640c7e3e43ca27178c003ca153a90814f7c78ada3e86a4aa4ce663bb784c4b8d)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
cf76cf044b01d9168a408d78aedae79295626bc4b6eb040d82663c5d8d814f6e)

RHEL 8

Download the version 3.3.2 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#) (SHA256 checksum 696bb3d67b3aca379106a409a8de814174df5b8308a2d4500bee5fc89f40070)
- [PKCS #11 library](#) (SHA256 checksum c58992e14d75c0cc7ae9f57746b40be5a4dbc1f2769e9d387eae39107b560749)
- [JCE provider](#) (SHA256 checksum 95e519d2bf656446141cd227e50447b67d485008e8b38151ea31f2a9ca855b49)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
e1ab86404d162e1169bb80364510119ce2c072fe30fbebe0a06bd2497f980f840)

Ubuntu 16.04 LTS

Download the version 3.3.2 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum 5797aa27b9ebe0aa52189b0a48a933a7a8ab404e651cbc43ec8dfe73fb03b66c)
- [PKCS #11 library](#) (SHA256 checksum 155cfb9f5e95ee03bcc298bd43ca3f5a883ae4ed69192d83ef34df2c6a117c7)
- [OpenSSL Dynamic Engine](#) (SHA256 checksum
b313ad3fc31fe6031707916567ee30d461a14511328db0b37c00ff0affa42608)
- [JCE provider](#) (SHA256 checksum 0c0307054bfbbf8c15830c5d431f9a21f96409a7969fda1b12379ce5444c56c1)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
e34e317cf3fe11a9b70fbfabcb35824f06800f3e2386d8ea02c977564eb3308a4)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.3.2 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#) (SHA256 checksum ccce515a4375e81b1493d641d523eac4599ad813d4fcc062bb872f16d57b094)
- [PKCS #11 library](#) (SHA256 checksum 25b54341d7efe594f13c7c7accdcac177241a610e82955bde24b82531236a52)
- [JCE provider](#) (SHA256 checksum 31cf9953ce86243b73863e1c5b0db21d9dcc4f26fa1c4741a9f0cc9489f389a2)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
1b6ff0c96cf7209c16ae69debb669f881a7a920f37226801456f6ec328c612d)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.3.2 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.3.2) software for Windows Server:

- [AWS CloudHSM for Windows Server](#) (SHA256 checksum
6e319786e92bbace07dd14e4bb9ce3d95d3ebf3c91434c73d8dab728a9e44fb2)
- [AWS CloudHSM Management Utility](#) (SHA256 checksum
fdb72c3d4c8f828f37e5e9e94c09b167806de556ff22dd1e3bfd58ed4f23f1b2)

Version 3.3.2 resolves an [issue](#) with the client_info script.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Updated the version for consistency.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.3.1

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 3.3.1 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Amazon Linux 2

Download the version 3.3.1 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.3.1.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.3.1 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

CentOS 8

Download the version 3.3.1 software for CentOS 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.3.1.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.3.1 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

RHEL 8

Download the version 3.3.1 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Ubuntu 16.04 LTS

Download the version 3.3.1 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)

- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Note

Due to the impending EOL of Ubuntu 16.04, we intend to drop support for this platform with the next release.

Ubuntu 18.04 LTS

Download the version 3.3.1 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.3.1 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.3.1) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)
- [AWS CloudHSM Management Utility](#)

Version 3.3.1 adds updates to all components.

AWS CloudHSM Client Software

- Improved stability and bug fixes.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.3.0

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 3.3.0 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Amazon Linux 2

Download the version 3.3.0 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

CentOS 6

AWS CloudHSM does not support CentOS 6 with Client SDK Version 3.3.0.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for CentOS 6 or choose a supported platform.

CentOS 7

Download the version 3.3.0 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

CentOS 8

Download the version 3.3.0 software for CentOS 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

RHEL 6

AWS CloudHSM does not support RedHat Enterprise Linux 6 with Client SDK Version 3.3.0.

Use [the section called “Version 3.2.1” \(p. 303\)](#) for RedHat Enterprise Linux 6 or choose a supported platform.

RHEL 7

Download the version 3.3.0 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

RHEL 8

Download the version 3.3.0 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Ubuntu 16.04 LTS

Download the version 3.3.0 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Ubuntu 18.04 LTS

Download the version 3.3.0 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)
- [AWS CloudHSM Management Utility](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.3.0 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.2.1) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)
- [AWS CloudHSM Management Utility](#)

Version 3.3.0 adds two-factor authentication (2FA) and other improvements.

AWS CloudHSM Client Software

- Added 2FA authentication for crypto officers (CO). For more information, see [Managing Two-Factor Authentication for Crypto Officers \(p. 70\)](#).
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see [Linux Support \(p. 315\)](#).
- Added a standalone version of CMU for use with Client SDK 5 or Client SDK 3. This is the same version of CMU included with the client daemon of version 3.3.0, and now you can download CMU without downloading the client daemon. For more information, see [Download and Install CMU \(p. 62\)](#).

PKCS #11 library

- Improved stability and bug fixes.
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see [Linux Support \(p. 315\)](#).

OpenSSL Dynamic Engine

- Updated the version for consistency
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see [Linux Support \(p. 315\)](#).

JCE provider

- Improved stability and bug fixes.
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see [Linux Support \(p. 315\)](#).

Windows (CNG and KSP providers)

- Updated the version for consistency

Version 3.2.1

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 3.2.1 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Amazon Linux 2

Download the version 3.2.1 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 6

Download the version 3.2.1 software for CentOS 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 7

Download the version 3.2.1 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 8

Download the version 3.2.1 software for CentOS 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)

RHEL 6

Download the version 3.2.1 software for RedHat Enterprise Linux 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 7

Download the version 3.2.1 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 8

Download the version 3.2.1 software for RedHat Enterprise Linux 8:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)

Ubuntu 16.04 LTS

Download the version 3.2.1 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Ubuntu 18.04 LTS

Download the version 3.2.1 software for Ubuntu 18.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [JCE provider](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.2.1 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.2.1) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)

Version 3.2.1 adds a compliance analysis between the AWS CloudHSM implementation of the PKCS #11 library and the PKCS #11 standard, new platforms, and other improvements.

AWS CloudHSM Client Software

- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see [??? \(p. 314\)](#).

PKCS #11 library

- [PKCS #11 library compliance report for client SDK 3.2.1](#)
- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see [??? \(p. 314\)](#).

OpenSSL Dynamic Engine

- No support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see [??? \(p. 521\)](#).

JCE provider

- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see [??? \(p. 314\)](#).

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.2.0

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 3.2.0 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Amazon Linux 2

Download the version 3.2.0 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 6

Download the version 3.2.0 software for CentOS 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 7

Download the version 3.2.0 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 6

Download the version 3.2.0 software for RedHat Enterprise Linux 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 7

Download the version 3.2.0 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Ubuntu 16.04 LTS

Download the version 3.2.0 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.2.0 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.2.0) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)

Version 3.2.0 adds support for masking passwords and other improvements.

AWS CloudHSM Client Software

- Adds support for hiding your password when using command-line tools. For more information, see [loginHSM and logoutHSM \(p. 136\)](#) (`cloudhsm_mgmt_util`) and [loginHSM and logoutHSM \(p. 220\)](#) (`key_mgmt_util`).

PKCS #11 library

- Adds support for hashing large data in software for some PKCS #11 mechanisms that were previously unsupported. For more information, see [Supported Mechanisms \(p. 329\)](#).

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Updated the version for consistency.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.1.2

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 3.1.2 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Amazon Linux 2

Download the version 3.1.2 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 6

Download the version 3.1.2 software for CentOS 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 7

Download the version 3.1.2 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 6

Download the version 3.1.2 software for RedHat Enterprise Linux 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 7

Download the version 3.1.2 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)

- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Ubuntu 16.04 LTS

Download the version 3.1.2 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.1.2 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.1.2) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)

Version 3.1.2 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency

PKCS #11 library

- Updated the version for consistency

OpenSSL Dynamic Engine

- Updated the version for consistency

JCE provider

- Update log4j to version 2.13.3

Windows (CNG and KSP providers)

- Updated the version for consistency

[**Version 3.1.1**](#)

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 3.1.1 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Amazon Linux 2

Download the version 3.1.1 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 6

Download the version 3.1.1 software for CentOS 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 7

Download the version 3.1.1 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 6

Download the version 3.1.1 software for RedHat Enterprise Linux 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 7

Download the version 3.1.1 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Ubuntu 16.04 LTS

Download the version 3.1.1 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.1.1 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.1.1) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 Library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Bug fixes and performance improvements.

Windows (CNG, KSP)

- Updated the version for consistency.

Version 3.1.0

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 3.1.0 software for Amazon Linux:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Amazon Linux 2

Download the version 3.1.0 software for Amazon Linux 2:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 6

Download the version 3.1.0 software for CentOS 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

CentOS 7

Download the version 3.1.0 software for CentOS 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 6

Download the version 3.1.0 software for RedHat Enterprise Linux 6:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

RHEL 7

Download the version 3.1.0 software for RedHat Enterprise Linux 7:

- [AWS CloudHSM Client](#)
- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Ubuntu 16.04 LTS

Download the version 3.1.0 software for Ubuntu 16.04 LTS:

- [AWS CloudHSM Client](#)

- [PKCS #11 Library](#)
- [OpenSSL Dynamic Engine](#)
- [JCE provider](#)

Windows Server

AWS CloudHSM supports 64-bit versions of Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016. The AWS CloudHSM 3.1.0 client software for Windows Server includes the required CNG and KSP providers. For details, see [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#). Download the latest version (3.1.0) software for Windows Server:

- [AWS CloudHSM for Windows Server](#)

Version 3.1.0 adds [standards-compliant AES key wrapping \(p. 95\)](#).

AWS CloudHSM Client Software

- A new requirement for upgrade: the version of your client must match the version of any software libraries you are using. To upgrade, you must use a batch command that upgrades the client and all the libraries at the same time. For more information, see [Client SDK 3 Upgrade \(p. 317\)](#).
- Key_mgmt_util (KMU) includes the following updates:
 - Added two new AES key wrap methods – standards-compliant AES key wrap with zero padding and AES key wrap with no padding. For more information, see [wrapKey \(p. 232\)](#) and [unwrapKey \(p. 226\)](#).
 - Disabled ability to specify custom IV when wrapping a key using AES_KEY_WRAP_PAD_PKCS5. For more information, see [AES Key Wrapping \(p. 95\)](#).

PKCS #11 Library

- Added two new AES key wrap methods - standards-compliant AES key wrap with zero padding and AES key wrap with no padding. For more information, see [AES Key Wrapping \(p. 95\)](#).
- You can configure salt length for RSA-PSS signatures. To learn how to use this feature, see [Configurable salt length for RSA-PSS signatures](#) on GitHub.

OpenSSL Dynamic Engine

- **BREAKING CHANGE:** TLS 1.0 and 1.2 cipher suites with SHA1 are not available in OpenSSL Engine 3.1.0. This issue will be resolved shortly.
- If you intend to install the OpenSSL Dynamic Engine library on RHEL 6 or CentOS 6, see a [known issue \(p. 521\)](#) about the default OpenSSL version installed on those operating systems.
- Improved stability and bug fixes

JCE provider

- **BREAKING CHANGE:** To address an issue with Java Cryptography Extension (JCE) compliance, AES wrap and unwrap now properly use the AESWrap algorithm instead of the AES algorithm. This means `Cipher.WRAP_MODE` and `Cipher.UNWRAP_MODE` no longer succeed for AES/ECB and AES/CBC mechanisms.

To upgrade to client version 3.1.0, you must update your code. If you have existing wrapped keys, you must pay particular attention to the mechanism you use to unwrap and how IV defaults have changed. If you wrapped keys with client version 3.0.0 or earlier, then in 3.1.1 you must use AESWrap/ECB/PKCS5Padding to unwrap your existing keys. For more information, see [AES Key Wrapping \(p. 95\)](#).

- You can list multiple keys with the same label from the JCE provider. To learn how to iterate through all available keys, see [Find all keys](#) on GitHub.
- You can set more restrictive values for attributes during key creation, including specifying different labels for public and private keys. For more information, see [Supported Java Attributes \(p. 360\)](#).

Windows (CNG, KSP)

- Improved stability and bug fixes.

Deprecated releases

Versions 3.0.1 and earlier are deprecated. We do not recommend using deprecated releases in production workloads. We do not provide backwards compatible updates for deprecated releases, nor do we host deprecated releases for download. If you experience production impact while using deprecated releases, you must upgrade to obtain software fixes.

End-of-life releases

AWS CloudHSM announces end of life for releases no longer compatible with the service. To preserve the safety of your application, we reserve the right to actively refuse connections from end-of-life releases.

- Currently no versions of the client SDK are end-of-life releases.

Supported platforms for the client SDKs

Base support is different for each version of the AWS CloudHSM Client SDK. Typically platform support for components in an SDK matches base support, but not always. To determine platform support for a given component, first make sure the platform you want appears in the base section for the SDK, then check for an exclusions or any other pertinent information in the component section.

Platform support changes over time. Earlier versions of the CloudHSM Client SDK may not support all the operating systems listed here. Use release notes to determine the operating system support for previous versions of the CloudHSM Client SDK. For more information, see [Download AWS CloudHSM Client SDK \(p. 260\)](#).

AWS CloudHSM supports only 64-bit operating systems.

Topics

- [Client SDK 3 platform support \(p. 314\)](#)
- [Client SDK 5 platform support \(p. 316\)](#)

Client SDK 3 platform support

Client SDK 3 requires a client daemon and offers command-line tools including, CloudHSM Management Utility (CMU), key management utility (KMU), and the configure tool.

Contents

- [Linux support \(p. 315\)](#)
- [Windows support \(p. 315\)](#)
- [Components support \(p. 315\)](#)
 - [PKCS #11 library \(p. 315\)](#)

- JCE provider ([p. 315](#))
- OpenSSL Dynamic Engine ([p. 315](#))
- CNG and KSP providers ([p. 316](#))

Linux support

- Amazon Linux
- Amazon Linux 2
- CentOS 6.10+ ²
- CentOS 7.3+
- CentOS 8 ^{1,4}
- Red Hat Enterprise Linux (RHEL) 6.10+ ²
- Red Hat Enterprise Linux (RHEL) 7.3+
- Red Hat Enterprise Linux (RHEL) 8 ¹
- Ubuntu 16.04 LTS ³
- Ubuntu 18.04 LTS ¹

[1] No support for OpenSSL Dynamic Engine. For more information, see [OpenSSL Dynamic Engine \(p. 315\)](#).

[2] No support for Client SDK 3.3.0 and later.

[3] SDK 3.4 is the last supported release on Ubuntu 16.04.

[4] SDK 3.4 is the last supported release on CentOS 8.3+.

Windows support

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016

Components support

PKCS #11 library

The PKCS #11 library is a Linux only component that matches Linux base support. For more information, see [the section called "Linux support" \(p. 315\)](#).

JCE provider

The JCE provider is a Linux only component that matches Linux base support. For more information, see [the section called "Linux support" \(p. 315\)](#).

- Requires OpenJDK 1.8

OpenSSL Dynamic Engine

The OpenSSL Dynamic Engine is Linux only component that does *not* match Linux base support. See the exclusions below.

- Requires OpenSSL 1.0.2[f+]

Unsupported platforms:

- CentOS 8
- Red Hat Enterprise Linux (RHEL) 8
- Ubuntu 18.04 LTS

These platforms ship with a version of OpenSSL incompatible with OpenSSL Dynamic Engine for Client SDK 3. AWS CloudHSM supports these platforms with OpenSSL Dynamic Engine for Client SDK 5.

CNG and KSP providers

The CNG and KSP providers is a Windows only component that matches Windows base support. For more information, see [Windows support \(p. 315\)](#).

Client SDK 5 platform support

Client SDK 5 does not require a client daemon.

Contents

- [Linux support for Client SDK 5 \(p. 316\)](#)
- [Windows support for Client SDK 5 \(p. 316\)](#)
- [Serverless support for Client SDK 5 \(p. 316\)](#)
- [Components support \(p. 317\)](#)
 - [PKCS #11 library \(p. 317\)](#)
 - [OpenSSL Dynamic Engine \(p. 317\)](#)
 - [JCE provider \(p. 317\)](#)

Linux support for Client SDK 5

- Amazon Linux
- Amazon Linux 2
- CentOS 7.8+
- CentOS 8.3+¹
- Red Hat Enterprise Linux (RHEL) 7.8+
- Red Hat Enterprise Linux (RHEL) 8.3+
- Ubuntu 18.04 LTS

[1] SDK 5.4.1 is the last supported release on CentOS 8.3+.

Windows support for Client SDK 5

- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Serverless support for Client SDK 5

- AWS Lambda

- Docker/ECS

Components support

PKCS #11 library

The PKCS #11 library is a cross-platform component that matches Linux and Windows Client SDK 5 base support. For more information, see [the section called “Linux support for Client SDK 5” \(p. 316\)](#) and [the section called “Windows support for Client SDK 5” \(p. 316\)](#).

OpenSSL Dynamic Engine

The OpenSSL Dynamic Engine is a Linux only component that requires OpenSSL 1.0.2 or 1.1.1.

JCE provider

The JCE provider is a Java SDK which is compatible with OpenJDK 8 and OpenJDK11 on all supported platforms.

Migrate from Client SDK 3 to Client SDK 5

Use this table to understand how to get started with Client SDK 5.

If you did this with Client SDK 3	You do this in Client SDK 5
Manage HSM users with CloudHSM Management Utility (CMU).	To manage HSM users with Client SDK 5, you must use a standalone version of CMU. For more information, see Download CMU (p. 62) and Manage HSM users with CMU (p. 64) .
Run a single HSM cluster.	To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting <code>disable_key_availability_check</code> to <code>True</code> . For more information, see Key Synchronization (p. 88) and Client SDK 5 Configure Tool (p. 245) .
Use Oracle Database Transparent data Encryption.	To use Oracle Database Transparent data Encryption, you must continue to use Client SDK 3. For more information, see Oracle Database Transparent data Encryption (p. 451) .
Use the same key handles across different runs of an application.	To successfully use key handles in Client SDK 5, you must obtain key handles each time you run an application. If you have existing applications that expect to use the same key handles across different runs, you must modify your code to obtain the key handle each time you run the application. This change is in compliance with the PKCS #11 2.40 specification .

Upgrade Client SDK 3 on Linux

With AWS CloudHSM Client SDK 3.1 and higher, the version of the client daemon and any components you install must match to upgrade. For all Linux-based systems, you must use a single command to

batch upgrade the client daemon with the same version of the PKCS #11 library, the Java Cryptographic Extension (JCE) provider, or the OpenSSL Dynamic Engine. This requirement does not apply to Windows-based systems because the binaries for the CNG and KSP providers are already included in the client daemon package.

To check the client daemon version

- On a Red Hat-based Linux system (including Amazon Linux and CentOS), use the following command:

```
rpm -qa | grep ^cloudhsm
```

- On a Debian-based Linux system, use the following command:

```
apt list --installed | grep ^cloudhsm
```

- On a Windows system, use the following command:

```
wmic product get name,version
```

Topics

- [Prerequisites \(p. 318\)](#)
- [Step 1: Stop the client daemon \(p. 321\)](#)
- [Step 2: Upgrade the client SDK \(p. 322\)](#)
- [Step 3: Start the client daemon \(p. 323\)](#)

Prerequisites

Download the latest version of AWS CloudHSM client daemon and choose your components.

Note

You do not have to install all the components. For every component you have installed, you must upgrade that component to match the version of the client daemon.

Latest Linux client daemon

Amazon Linux

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

Amazon Linux 2

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

CentOS 7

```
sudo yum install wget
```

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

CentOS 8

```
sudo yum install wget
```

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

RHEL 7

```
sudo yum install wget
```

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

RHEL 8

```
sudo yum install wget
```

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

Ubuntu 16.04 LTS

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

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

Latest PKCS #11 library

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

Latest OpenSSL Dynamic Engine

Amazon Linux

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

Amazon Linux 2

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

CentOS 7

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

RHEL 7

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

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-dyn_latest_amd64.deb
```

Latest JCE provider

Amazon Linux

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

Amazon Linux 2

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

CentOS 7

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

CentOS 8

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

RHEL 7

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

RHEL 8

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

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

Step 1: Stop the client daemon

Use the following command to stop the client daemon.

Amazon Linux

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

Amazon Linux 2

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

CentOS 7

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

CentOS 8

```
$ sudo service clouhdsm-client stop
```

RHEL 7

```
$ sudo service clouhdsm-client stop
```

RHEL 8

```
$ sudo service clouhdsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service clouhdsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service clouhdsm-client stop
```

Step 2: Upgrade the client SDK

The following command shows the syntax required to upgrade the client daemon and components. Before you run the command, remove any components you don't intend to upgrade.

Amazon Linux

```
$ sudo yum install ./clouhdsm-client-latest.el6.x86_64.rpm \
<./clouhdsm-client-pkcs11-latest.el6.x86_64.rpm> \
<./clouhdsm-client-dyn-latest.el6.x86_64.rpm> \
<./clouhdsm-client-jce-latest.el6.x86_64.rpm>
```

Amazon Linux 2

```
$ sudo yum install ./clouhdsm-client-latest.el7.x86_64.rpm \
<./clouhdsm-client-pkcs11-latest.el7.x86_64.rpm> \
<./clouhdsm-client-dyn-latest.el7.x86_64.rpm> \
<./clouhdsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 7

```
$ sudo yum install ./clouhdsm-client-latest.el7.x86_64.rpm \
<./clouhdsm-client-pkcs11-latest.el7.x86_64.rpm> \
<./clouhdsm-client-dyn-latest.el7.x86_64.rpm> \
<./clouhdsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 8

```
$ sudo yum install ./clouhdsm-client-latest.el8.x86_64.rpm \
<./clouhdsm-client-pkcs11-latest.el8.x86_64.rpm> \
<./clouhdsm-client-jce-latest.el8.x86_64.rpm>
```

RHEL 7

```
$ sudo yum install ./clouhdhsm-client-latest.el7.x86_64.rpm \
    <./clouhdhsm-client-pkcs11-latest.el7.x86_64.rpm> \
    <./clouhdhsm-client-dyn-latest.el7.x86_64.rpm> \
    <./clouhdhsm-client-jce-latest.el7.x86_64.rpm>
```

RHEL 8

```
$ sudo yum install ./clouhdhsm-client-latest.el8.x86_64.rpm \
    <./clouhdhsm-client-pkcs11-latest.el8.x86_64.rpm> \
    <./clouhdhsm-client-jce-latest.el8.x86_64.rpm>
```

Ubuntu 16.04 LTS

```
$ sudo apt install ./clouhdhsm-client_latest_amd64.deb \
    <clouhdhsm-client-pkcs11_latest_amd64.deb> \
    <clouhdhsm-client-dyn_latest_amd64.deb> \
    <clouhdhsm-client-jce_latest_amd64.deb>
```

Ubuntu 18.04 LTS

```
$ sudo apt install ./clouhdhsm-client_latest_u18.04_amd64.deb \
    <clouhdhsm-client-pkcs11_latest_amd64.deb> \
    <clouhdhsm-client-jce_latest_amd64.deb>
```

Step 3: Start the client daemon

Use the following command to start the client daemon.

Amazon Linux

```
$ sudo start clouhdhsm-client
```

Amazon Linux 2

```
$ sudo service clouhdhsm-client start
```

CentOS 7

```
$ sudo service clouhdhsm-client start
```

CentOS 8

```
$ sudo service clouhdhsm-client start
```

RHEL 7

```
$ sudo service clouhdhsm-client start
```

RHEL 8

```
$ sudo service clouhdhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

PKCS #11 library

PKCS #11 is a standard for performing cryptographic operations on hardware security modules (HSM). AWS CloudHSM offers implementations of the PKCS #11 library in Client SDK 3 and Client SDK 5, both of which are compliant with PKCS #11 version 2.40.

Client SDK 5

Client SDK 5 does not require a client daemon and offers cross-platform support on Windows and Linux.

Client SDK 3

Client SDK 3 requires a client daemon to connect to the cluster and is only supported on Linux.

Topics

- [Installing the PKCS #11 library \(p. 324\)](#)
- [Authenticating to the PKCS #11 library \(p. 329\)](#)
- [Supported key types \(p. 329\)](#)
- [Supported mechanisms \(p. 329\)](#)
- [Supported API operations \(p. 334\)](#)
- [Supported attributes \(p. 336\)](#)
- [Code samples for the PKCS #11 library \(p. 346\)](#)

Installing the PKCS #11 library

This topic provides PKCS #11 library installation instruction for Client SDK 5 and Client SDK 3. For more information about the Client SDK or PKCS #11 library, see [Using the Client SDK \(p. 258\)](#) and [PKCS #11 library \(p. 324\)](#).

Topics

- [Install PKCS #11 library Client SDK 3 \(p. 324\)](#)
- [Install PKCS #11 library Client SDK 5 \(p. 327\)](#)

Install PKCS #11 library Client SDK 3

Prerequisites for Client SDK 3

The PKCS #11 library requires 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 \(Linux\) \(p. 28\)](#). After you install and configure the client, use the following command to start it.

Amazon Linux

```
$ sudo start clouhdsm-client
```

Amazon Linux 2

```
$ sudo service clouhdsm-client start
```

CentOS 7

```
$ sudo service clouhdsm-client start
```

CentOS 8

```
$ sudo service clouhdsm-client start
```

RHEL 7

```
$ sudo service clouhdsm-client start
```

RHEL 8

```
$ sudo service clouhdsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service clouhdsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service clouhdsm-client start
```

Install the PKCS #11 library for Client SDK 3

The following command downloads and installs the PKCS #11 library.

Amazon Linux

```
$ wget https://s3.amazonaws.com/clouhdsmv2-software/CloudHsmClient/EL6/clouhdsm-client-pkcs11-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./clouhdsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/clouhdsmv2-software/CloudHsmClient/EL7/clouhdsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./clouhdsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

- If the EC2 instance on which you installed the PKCS #11 library has no other components from Client SDK 3 installed, you must bootstrap Client SDK 3. You only have to do this once on each instance with a component from Client SDK 3.

For more information about bootstrapping, see [Connecting to the cluster \(p. 38\)](#).

- You can find the PKCS #11 library files in the following locations:

Linux binaries, configuration scripts, certificates, and log files:

```
/opt/cloudhsm/lib
```

Install PKCS #11 library Client SDK 5

Install the PKCS #11 library for Client SDK 5

With Client SDK 5, you are not required to install or run a client daemon.

To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see [Key Synchronization \(p. 88\)](#) and [Client SDK 5 Configure Tool \(p. 245\)](#).

For more information about the PKCS #11 library in Client SDK 5, see [PKCS #11 library \(p. 324\)](#).

- Download and install the PKCS #11 library.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-pkcs11-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7.8+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8.3+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7.8+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8.3+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-pkcs11_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u18.04_amd64.deb
```

Windows Server 2016

1. Download [PKCS #11 library for Client SDK 5](#).
2. Run the PKCS #11 library installer ([AWSCloudHSMPKCS11-latest.msi](#)) with Windows administrative privilege.

Windows Server 2019

1. Download [PKCS #11 library for Client SDK 5](#).
2. Run the PKCS #11 library installer ([AWSCloudHSMPKCS11-latest.msi](#)) with Windows administrative privilege.

- You must bootstrap Client SDK 5. For more information about bootstrapping, see [Bootstrapping the Client SDK \(p. 38\)](#).
- You can find the PKCS #11 library files in the following locations:

Linux binaries, configuration scripts, and log files:

```
/opt/cloudhsm/lib
```

Windows binaries:

```
C:\ProgramFiles\Amazon\CloudHSM
```

Windows configuration scripts and log files:

```
C:\ProgramData\Amazon\CloudHSM
```

Authenticating to the PKCS #11 library

When you use the PKCS #11 library, your application runs as a particular [crypto user \(CU\) \(p. 59\)](#) 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 \(p. 64\)](#) for your application.

To specify the CU to PKCS #11 library, 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 library pin to the CU with user name `CryptoUser` and password `CUPassword123!`.

```
CryptoUser:CUPassword123!
```

Supported key types

The PKCS #11 library supports the following key types.

Key Type	Description
RSA	Generate 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	Generate 128, 192, and 256-bit AES keys.
Triple DES (3DES)	Generate 192-bit Triple DES keys.
GENERIC_SECRET	Generate 1 to 64 bytes generic secrets.

Supported mechanisms

The PKCS #11 library 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,^{4 (p. 334)} AES-GCM, RSA-AES, and RSA-OAEP
- **Key derivation** – ECDH,^{5 (p. 334)} SP800-108 CTR KDF

The PKCS #11 library mechanism-function table

The PKCS #11 library is compliant with version 2.40 of the PKCS #11 specification. 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 AWS CloudHSM supports the mechanism for the function. We do not support all possible functions listed in the PKCS #11 specification. A ✗ mark indicates that AWS 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 library mechanisms and functions

Mechanism	Functions							
	Generate Key or Key Pair	Sign & Verify	SR & VR	Digest	Encrypt & Decrypt	Derive Key	Wrap & UnWrap	
CKM_RSA_PKCS_KEY_PAIR_GEN	✓							
CKM_RSA_X9_31_KEY_PAIR_GEN	✓ ²							
CKM_RSA_PKCS		✓ ¹	✗		✓ ¹		✓ ¹	
CKM_RSA_PKCS_OAEP					✓ ¹		✓ ⁶	
CKM_SHA1_RSA_PKCS		✓ ^{3.2}						
CKM_SHA224_RSA_PKCS		✓ ^{3.2}						
CKM_SHA256_RSA_PKCS		✓ ^{3.2}						
CKM_SHA384_RSA_PKCS		✓ ^{2,3.2 (p. 332)}						
CKM_SHA512_RSA_PKCS		✓ ^{3.2}						
CKM_RSA_PKCS_PSS		✓ ¹						
CKM_SHA1_RSA_PKCS_PSS		✓ ^{3.2}						
CKM_SHA224_RSA_PKCS_PSS		✓ ^{3.2}						
CKM_SHA256_RSA_PKCS_PSS		✓ ^{3.2}						
CKM_SHA384_RSA_PKCS_PSS		✓ ^{2,3.2}						
CKM_SHA512_RSA_PKCS_PSS		✓ ^{3.2}						
CKM_EC_KEY_PAIR_GEN	✓							
CKM_ECDSA		✓ ¹						
CKM_ECDSA_SHA1		✓ ^{3.2}						
CKM_ECDSA_SHA224		✓ ^{3.2}						
CKM_ECDSA_SHA256		✓ ^{3.2}						
CKM_ECDSA_SHA384		✓ ^{3.2}						

Mechanism	Functions					
CKM_ECDSA_SHA512		✓ ^{3.2}				
CKM_ECDH1_DERIVE					✓ ⁵	
CKM_SP800_108_COUNTER_KDF					✓	
CKM_GENERIC_SECRET_KEY_GEN	✓					
CKM_AES_KEY_GEN	✓					
CKM_AES_ECB				✓		✗
CKM_AES_CTR				✓		✗
CKM_AES_CBC			✓ ^{3.3}			✗
CKM_AES_CBC_PAD			✓			✗
CKM_DES3_KEY_GEN	✓					
CKM_DES3_CBC			✓ ^{3.3}			✗
CKM_DES3_CBC_PAD			✓			✗
CKM_DES3_ECB			✓			✗
CKM_AES_GCM			✓ ^{3.3, 4}		✓ ^{7.1}	
CKM_CLOUDHSM_AES_GCM			✓ ^{7.1}		✓ ^{7.1}	
CKM_SHA_1			✓ ^{3.1}			
CKM_SHA_1_HMAC		✓ ^{3.3}				
CKM_SHA224			✓ ^{3.1}			
CKM_SHA224_HMAC		✓ ^{3.3}				
CKM_SHA256			✓ ^{3.1}			
CKM_SHA256_HMAC		✓ ^{3.3}				
CKM_SHA384			✓ ^{3.1}			
CKM_SHA384_HMAC		✓ ^{3.3}				
CKM_SHA512			✓ ^{3.1}			
CKM_SHA512_HMAC		✓ ^{3.3}				
CKM_RSA_AES_KEY_WRAP						✓
CKM_AES_KEY_WRAP						✓ ^{7.2}
CKM_AES_KEY_WRAP_PAD						✓ ^{7.2}

Mechanism	Functions						
CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD							✓ 7.1
CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD							✓ 7.1
CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD							✓ 7.1

Mechanism 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] AWS CloudHSM approaches hashing differently based on the Client SDK. For Client SDK 5, we do all hashing for these functions in software and we do not copy to the HSM, meaning that there is no data size limit. For Client SDK 3, where we do the hashing depends on data size and whether you're using single-part or multipart operations.

Single-part operations in Client SDK 3

Table 3.1 lists the maximum data set size for each mechanism for Client SDK 3. The entire hash is computed inside the HSM. No support for data sizes greater than 16KB.

Table 3.1, Maximum data set size for single-part operations

Mechanism	Maximum Data Size
CKM_SHA_1	16296
CKM_SHA224	16264
CKM_SHA256	16296
CKM_SHA384	16232
CKM_SHA512	16232

Multipart operations Client SDK 3

Support for data sizes greater than 16 KB, but data size determines where the hashing takes place. Data buffers less than 16 KB are hashed inside the HSM. Buffers between 16 KB and the maximum data size for your system are hashed locally in software. *Remember:* Hash functions do not require cryptographic secrets, so you can safely compute them outside of the HSM.

- [3.2] AWS CloudHSM approaches hashing differently based on the Client SDK. For Client SDK 5, we do all hashing for supported functions in software and we do not copy to the HSM, meaning that there is no data size limit. For Client SDK 3, where we do the hashing depends on data size and whether you're using single-part or multipart operations.

Single-part operations Client SDK 3

Table 3.2 lists the maximum data set size for each mechanism for Client SDK 3. No support for data sizes greater than 16KB.

Table 3.2, Maximum data set size for single-part operations

Mechanism	Maximum Data Size
CKM_SHA1_RSA_PKCS	16296
CKM_SHA224_RSA_PKCS	16264
CKM_SHA256_RSA_PKCS	16296
CKM_SHA384_RSA_PKCS	16232
CKM_SHA512_RSA_PKCS	16232
CKM_SHA1_RSA_PKCS_PSS	16296
CKM_SHA224_RSA_PKCS_PSS	16264
CKM_SHA256_RSA_PKCS_PSS	16296
CKM_SHA384_RSA_PKCS_PSS	16232
CKM_SHA512_RSA_PKCS_PSS	16232
CKM_ECDSA_SHA1	16296
CKM_ECDSA_SHA224	16264
CKM_ECDSA_SHA256	16296
CKM_ECDSA_SHA384	16232
CKM_ECDSA_SHA512	16232

Multipart operations Client SDK 3

Support for data sizes greater than 16 KB, but data size determines where the hashing takes place. Data buffers less than 16 KB are hashed inside the HSM. Buffers between 16 KB and the maximum data size for your system are hashed locally in software. *Remember:* Hash functions do not require cryptographic secrets, so you can safely compute them outside of the HSM.

- [3.3] 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. For these mechanisms, all the data processing must occur inside the HSM. The following table lists maximum data size set for each mechanism (**Note:** These data size limits apply to both Client SDK 3 and Client SDK 5):

Table 3.3, Maximum data set size

Mechanism	Maximum Data Size
CKM_SHA_1_HMAC	16288
CKM_SHA224_HMAC	16256
CKM_SHA256_HMAC	16288
CKM_SHA384_HMAC	16224
CKM_SHA512_HMAC	16224

Mechanism	Maximum Data Size
CKM_AES_CBC	16272
CKM_AES_GCM	16224
CKM_CLOUDHSM_AES_GCM	16224
CKM_DES3_CBC	16280

- [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] **Client SDK 3 only.** 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 library \(p. 515\)](#).
- [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:
 - CKM_SHA_1 using CKG_MGF1_SHA1
 - CKM_SHA224 using CKG_MGF1_SHA224
 - CKM_SHA256 using CKG_MGF1_SHA256
 - CKM_SHA384 using CKM_MGF1_SHA384
 - CKM_SHA512 using CKM_MGF1_SHA512
- [7.1] Vendor-defined mechanism. In order to use the CloudHSM vendor defined mechanisms, PKCS#11 applications must include /opt/cloudhsm/include/pkcs11t.h during compilation.

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.

CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD: AES Key Wrap with PKCS #5 Padding

CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD: AES Key Wrap with Zero Padding

For additional information regarding AES key wrapping, see [AES Key Wrapping \(p. 95\)](#).

- [7.2] To use the following mechanisms with Client SDK 5, use the vendor defined alternative instead.

CKM_AES_KEY_WRAP: use CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD instead.

CKM_AES_KEY_WRAP_PAD: use CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD instead.

Supported API operations

The PKCS #11 library supports the following PKCS #11 API operations.

- C_CloseAllSessions
- C_CloseSession
- C_CreateObject
- C_Decrypt

- `C_DecryptFinal`
- `C_DecryptInit`
- `C_DecryptUpdate`
- `C_DeriveKey`
- `C_DestroyObject`
- `C_Digest`
- `C_DigestFinal`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_Encrypt`
- `C_EncryptFinal`
- `C_EncryptInit`
- `C_EncryptUpdate`
- `C_Finalize`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_FindObjectsInit`
- `C_GenerateKey`
- `C_GenerateKeyValuePair`
- `C_GenerateRandom`
- `C_GetAttributeValue`
- `C_GetFunctionList`
- `C_GetInfo`
- `C_GetMechanismInfo`
- `C_GetMechanismList`
- `C_GetSessionInfo`
- `C_GetSlotInfo`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_Initialize`
- `C_Login`
- `C_Logout`
- `C_OpenSession`
- `C_Sign`
- `C_SignFinal`
- `C_SignInit`
- `C_SignRecover (Client SDK 3 support only)`
- `C_SignRecoverInit (Client SDK 3 support only)`
- `C_SignUpdate`
- `C_UnWrapKey`
- `C_Verify`
- `C_VerifyFinal`
- `C_VerifyInit`
- `C_VerifyRecover (Client SDK 3 support only)`
- `C_VerifyRecoverInit (Client SDK 3 support only)`
- `C_VerifyUpdate`

- C_WrapKey

Supported 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 the PKCS #11 library, we assign default values as specified by the PKCS #11 standard.

AWS 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 library attributes table

The PKCS #11 library 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.
- An empty cell in the Default Value column indicates that there is no specific default value assigned to the attribute.

GenerateKeyPair

Attribute	Key Type				Default Value
	EC private	EC public	RSA private	RSA public	
CKA_CLASS	✓	✓	✓	✓	
CKA_KEY_TYPE	✓	✓	✓	✓	
CKA_LABEL	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	True
CKA_TOKEN	✓	✓	✓	✓	False
CKA_PRIVATE	✓ ¹	✓ ¹	✓ ¹	✓ ¹	True
CKA_ENCRYPT	✗	✓	✗	✓	False
CKA_DECRYPT	✓	✗	✓	✗	False

Attribute	Key Type					Default Value
CKA_DERIVE	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓ ¹	True				
CKA_DESTROYABLE	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✗	✓	✗	✗	False
CKA_SIGN_RECOVER	✗	✗	✓ ³	✗	✗	
CKA_VERIFY	✗	✓	✗	✓	✓	False
CKA_VERIFY_RECOVER	✗	✗	✗	✗	✓ ⁴	
CKA_WRAP	✗	✓	✗	✓	✓	False
CKA_WRAP_TEMPLATE	✗	✓	✗	✓	✓	
CKA_TRUSTED	✗	✓	✗	✓	✓	False
CKA_WRAP_WITH_TRUSTED	✓	✗	✓	✗	✗	False
CKA_UNWRAP	✓	✗	✓	✗	✗	False
CKA_UNWRAP_TEMPLATE	✓	✗	✓	✗	✗	
CKA_SENSITIVE	✓	✗	✓	✗	✗	True
CKA_ALWAYS_SENSITIVE	R	✗	R	✗		
CKA_EXTRACTABLE	✓	✗	✓	✗	✗	True
CKA_NEVER_EXTRACTABLE	R	✗	R	✗		
CKA_MODULUS	✗	✗	✗	✗	✗	
CKA_MODULUS_BITS	✗	✗	✗	✗	✓ ²	
CKA_PRIME_1	✗	✗	✗	✗	✗	
CKA_PRIME_2	✗	✗	✗	✗	✗	
CKA_COEFFICIENT	✗	✗	✗	✗	✗	
CKA_EXPONENT_1	✗	✗	✗	✗	✗	
CKA_EXPONENT_2	✗	✗	✗	✗	✗	
CKA_PRIVATE_EXPONENT	✗	✗	✗	✗	✗	
CKA_PUBLIC_EXPONENT	✗	✗	✗	✗	✓ ²	
CKA_EC_PARAMS	✗	✓ ²	✗	✗	✗	
CKA_EC_POINT	✗	✗	✗	✗	✗	
CKA_VALUE	✗	✗	✗	✗	✗	

Attribute	Key Type					Default Value
CKA_VALUE_LEN	*	*	*	*	*	
CKA_CHECK_VALUE	R	R	R	R		

GenerateKey

Attribute	Key Type				Default Value
	AES	DES3	Generic Secret		
CKA_CLASS	✓	✓	✓		
CKA_KEY_TYPE	✓	✓	✓		
CKA_LABEL	✓	✓	✓		
CKA_ID	✓	✓	✓		
CKA_LOCAL	R	R	R	True	
CKA_TOKEN	✓	✓	✓	False	
CKA_PRIVATE	✓ ¹	✓ ¹	✓ ¹	True	
CKA_ENCRYPT	✓	✓	✗	False	
CKA_DECRYPT	✓	✓	✗	False	
CKA_DERIVE	✓	✓	✓	False	
CKA_MODIFIABLE	✓ ¹	✓ ¹	✓ ¹	True	
CKA_DESTROYABLE	✓	✓	✓	True	
CKA_SIGN	✓	✓	✓	True	
CKA_SIGN_RECOVER	✗	✗	✗		
CKA_VERIFY	✓	✓	✓	True	
CKA_VERIFY_RECOVER	✗	✗	✗		
CKA_WRAP	✓	✓	✗	False	
CKA_WRAP_TEMPLATE	✓	✓	✗		
CKA_TRUSTED	✓	✓	✗	False	
CKA_WRAP_WITH_TRUSTED	✓	✓	✓	False	
CKA_UNWRAP	✓	✓	✗	False	
CKA_UNWRAP_TEMPLATE	✓	✓	✗		
CKA_SENSITIVE	✓	✓	✓	True	
CKA_ALWAYS_SENSITIVE	✗	✗	✗		

Attribute	Key Type				Default Value
CKA_EXTRACTABLE	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	R	
CKA_MODULUS	✗	✗	✗	✗	
CKA_MODULUS_BITS	✗	✗	✗	✗	
CKA_PRIME_1	✗	✗	✗	✗	
CKA_PRIME_2	✗	✗	✗	✗	
CKA_COEFFICIENT	✗	✗	✗	✗	
CKA_EXPONENT_1	✗	✗	✗	✗	
CKA_EXPONENT_2	✗	✗	✗	✗	
CKA_PRIVATE_EXPONENT	✗	✗	✗	✗	
CKA_PUBLIC_EXPONENT	✗	✗	✗	✗	
CKA_EC_PARAMS	✗	✗	✗	✗	
CKA_EC_POINT	✗	✗	✗	✗	
CKA_VALUE	✗	✗	✗	✗	
CKA_VALUE_LEN	✓ ²	✗	✗	✓ ²	
CKA_CHECK_VALUE	R	R	R	R	

CreateObject

Attribute	Key Type							Default Value
	EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret	
CKA_CLASS	✓ ²							
CKA_KEY_TYPE	✓ ²							
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓ ¹	True						
CKA_ENCRYPT	✗	✗	✗	✓	✓	✓	✗	False
CKA_DECRYPT	✗	✗	✓	✗	✓	✓	✗	False

Attribute	Key Type								Default Value
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓ ¹	True							
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✗	✓	✗	✓	✓	✓	✓	False
CKA_SIGN_RECOVER	✗	✗	✓ ³	✗	✗	✗	✗	✗	False
CKA_VERIFY	✗	✓	✗	✓	✓	✓	✓	✓	False
CKA_VERIFY_RECOVER	✗	✗	✗	✓ ⁴	✗	✗	✗	✗	
CKA_WRAP	✗	✗	✗	✓	✓	✓	✓	✗	False
CKA_WRAP_TEMPLATE	✗	✓	✗	✓	✓	✓	✓	✗	
CKA_TRUSTED	✗	✓	✗	✓	✓	✓	✓	✗	False
CKA_WRAP_WITH_TRUSTED	✓	✗	✓	✗	✓	✓	✓	✓	False
CKA_UNWRAP	✗	✗	✓	✗	✓	✓	✓	✗	False
CKA_UNWRAP_TEMPLATE	✓	✗	✓	✗	✓	✓	✓	✗	
CKA_SENSITIVE	✓	✗	✓	✗	✓	✓	✓	✓	True
CKA_ALWAYS_SENSITIVE	R	✗	R	✗	R	R	R	R	
CKA_EXTRACTABLE	✓	✗	✓	✗	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	✗	R	✗	R	R	R	R	
CKA_MODULUS	✗	✗	✓ ²	✓ ²	✗	✗	✗	✗	
CKA_MODULUS_BITS	✗	✗	✗	✗	✗	✗	✗	✗	
CKA_PRIME_1	✗	✗	✓	✗	✗	✗	✗	✗	
CKA_PRIME_2	✗	✗	✓	✗	✗	✗	✗	✗	
CKA_COEFFICIENT	✗	✗	✓	✗	✗	✗	✗	✗	
CKA_EXPONENT_1	✗	✗	✓	✗	✗	✗	✗	✗	
CKA_EXPONENT_2	✗	✗	✓	✗	✗	✗	✗	✗	
CKA_PRIVATE_EXPONENT	✗	✗	✓ ²	✗	✗	✗	✗	✗	
CKA_PUBLIC_EXPONENT	✗	✗	✓ ²	✓ ²	✗	✗	✗	✗	
CKA_EC_PARAMS	✓ ²	✓ ²	✗	✗	✗	✗	✗	✗	
CKA_EC_POINT	✗	✓ ²	✗	✗	✗	✗	✗	✗	
CKA_VALUE	✓ ²	✗	✗	✗	✓ ²	✓ ²	✓ ²	✓ ²	

Attribute	Key Type							Default Value
CKA_VALUE_LEN	*	*	*	*	*	*	*	
CKA_CHECK_VALUE	R	R	R	R	R	R	R	

UnwrapKey

Attribute	Key Type					Default Value
	EC private	RSA private	AES	DES3	Generic Secret	
CKA_CLASS	✓ ²					
CKA_KEY_TYPE	✓ ²					
CKA_LABEL	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	False
CKA_PRIVATE	✓ ¹	True				
CKA_ENCRYPT	*	*	✓	✓	*	False
CKA_DECRYPT	*	✓	✓	✓	*	False
CKA_DERIVE	✓	✓	✓	✓	✓	False
CKA_MODIFIABLE	✓ ¹	True				
CKA_DESTROYABLE	✓	✓	✓	✓	✓	True
CKA_SIGN	✓	✓	✓	✓	✓	False
CKA_SIGN_RECOVER	*	✓ ³	*	*	*	False
CKA_VERIFY	*	*	✓	✓	✓	False
CKA_VERIFY_RECOVER	*	*	*	*	*	
CKA_WRAP	*	*	✓	✓	*	False
CKA_UNWRAP	*	✓	✓	✓	*	False
CKA_SENSITIVE	✓	✓	✓	✓	✓	True
CKA_EXTRACTABLE	✓	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE	R	R	R	R	R	
CKA_ALWAYS_SENSITIVE	R	R	R	R	R	

Attribute	Key Type					Default Value
CKA_MODULUS	✗	✗	✗	✗	✗	
CKA_MODULUS_BITS	✗	✗	✗	✗	✗	
CKA_PRIME_1	✗	✗	✗	✗	✗	
CKA_PRIME_2	✗	✗	✗	✗	✗	
CKA_COEFFICIENT	✗	✗	✗	✗	✗	
CKA_EXPONENT_1	✗	✗	✗	✗	✗	
CKA_EXPONENT_2	✗	✗	✗	✗	✗	
CKA_PRIVATE_EXPONENT	✗	✗	✗	✗	✗	
CKA_PUBLIC_EXPONENT	✗	✗	✗	✗	✗	
CKA_EC_PARAMS	✗	✗	✗	✗	✗	
CKA_EC_POINT	✗	✗	✗	✗	✗	
CKA_VALUE	✗	✗	✗	✗	✗	
CKA_VALUE_LEN	✗	✗	✗	✗	✗	
CKA_CHECK_VALUE	R	R	R	R	R	

DeriveKey

Attribute	Key Type			Default Value
	AES	DES3	Generic Secret	
CKA_CLASS	✓ ²	✓ ²	✓ ²	
CKA_KEY_TYPE	✓ ²	✓ ²	✓ ²	
CKA_LABEL	✓	✓	✓	
CKA_ID	✓	✓	✓	
CKA_LOCAL	R	R	R	True
CKA_TOKEN	✓	✓	✓	False
CKA_PRIVATE	✓ ¹	✓ ¹	✓ ¹	True
CKA_ENCRYPT	✓	✓	✗	False
CKA_DECRYPT	✓	✓	✗	False
CKA_DERIVE	✓	✓	✓	False
CKA_MODIFIABLE	✓ ¹	✓ ¹	✓ ¹	True

Attribute	Key Type				Default Value
CKA_DESTROYABLE	✓ ¹	✓ ¹	✓ ¹		True
CKA_SIGN	✓	✓	✓		False
CKA_SIGN_RECOVER	✗	✗	✗		
CKA_VERIFY	✓	✓	✓		False
CKA_VERIFY_RECOVER	✗	✗	✗		
CKA_WRAP	✓	✓	✗		False
CKA_UNWRAP	✓	✓	✗		False
CKA_SENSITIVE	✓	✓	✓		True
CKA_EXTRACTABLE	✓	✓	✓		True
CKA_NEVER_EXTRACTABLE	R	R	R		
CKA_ALWAYS_SENSITIVE	R	R	R		
CKA_MODULUS	✗	✗	✗		
CKA_MODULUS_BITS	✗	✗	✗		
CKA_PRIME_1	✗	✗	✗		
CKA_PRIME_2	✗	✗	✗		
CKA_COEFFICIENT	✗	✗	✗		
CKA_EXPONENT_1	✗	✗	✗		
CKA_EXPONENT_2	✗	✗	✗		
CKA_PRIVATE_EXPONENT	✗	✗	✗		
CKA_PUBLIC_EXPONENT	✗	✗	✗		
CKA_EC_PARAMS	✗	✗	✗		
CKA_EC_POINT	✗	✗	✗		
CKA_VALUE	✗	✗	✗		
CKA_VALUE_LEN	✓ ²	✗	✓ ²		
CKA_CHECK_VALUE	R	R	R		

GetAttributeValue

Attribute	Key Type								
		EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret	
CKA_CLASS		✓	✓	✓	✓	✓	✓	✓	

Attribute	Key Type							
CKA_KEY_TYPE	✓	✓	✓	✓	✓	✓	✓	✓
CKA_LABEL	✓	✓	✓	✓	✓	✓	✓	✓
CKA_ID	✓	✓	✓	✓	✓	✓	✓	✓
CKA_LOCAL	✓	✓	✓	✓	✓	✓	✓	✓
CKA_TOKEN	✓	✓	✓	✓	✓	✓	✓	✓
CKA_PRIVATE	✓ ¹							
CKA_ENCRYPT	✗	✗	✗	✓	✓	✓	✓	✗
CKA_DECRYPT	✗	✗	✓	✗	✓	✓	✓	✗
CKA_DERIVE	✓	✓	✓	✓	✓	✓	✓	✓
CKA_MODIFIABLE	✓	✓	✓	✓	✓	✓	✓	✓
CKA_DESTROYABLE	✓	✓	✓	✓	✓	✓	✓	✓
CKA_SIGN	✓	✗	✓	✗	✓	✓	✓	✓
CKA_SIGN_RECOVER	✗	✗	✓	✗	✗	✗	✗	✗
CKA_VERIFY	✗	✓	✗	✓	✓	✓	✓	✓
CKA_VERIFY_RECOVER	✗	✗	✗	✓	✗	✗	✗	✗
CKA_WRAP	✗	✗	✗	✓	✓	✓	✓	✗
CKA_WRAP_TEMPLATE	✗	✓	✗	✓	✓	✓	✓	✗
CKA_TRUSTED	✗	✓	✗	✓	✓	✓	✓	✓
CKA_WRAP_WITH_TRUSTED	✓	✗	✓	✗	✓	✓	✓	✓
CKA_UNWRAP	✗	✗	✓	✗	✓	✓	✓	✗
CKA_UNWRAP_TEMPLATE	✓	✗	✓	✗	✓	✓	✓	✗
CKA_SENSITIVE	✓	✗	✓	✗	✓	✓	✓	✓
CKA_EXTRACTABLE	✓	✗	✓	✗	✓	✓	✓	✓
CKA_NEVER_EXTRACTABLE	✓	✗	✓	✗	✓	✓	✓	✓
CKA_ALWAYS_SENSITIVE	R	R;	R	R	R	R	R	R
CKA_MODULUS	✗	✗	✓	✓	✗	✗	✗	✗
CKA_MODULUS_BITS	✗	✗	✗	✓	✓	✗	✗	✗
CKA_PRIME_1	✗	✗	S	✗	✗	✗	✗	✗
CKA_PRIME_2	✗	✗	S	✗	✗	✗	✗	✗
CKA_COEFFICIENT	✗	✗	S	✗	✗	✗	✗	✗
CKA_EXPONENT_1	✗	✗	S	✗	✗	✗	✗	✗

Attribute	Key Type						
CKA_EXPONENT_2	✗	✗	S	✗	✗	✗	✗
CKA_PRIVATE_EXPONENT	✗	✗	S	✗	✗	✗	✗
CKA_PUBLIC_EXPONENT	✗	✗	✓	✓	✗	✗	✗
CKA_EC_PARAMS	✓	✓	✗	✗	✗	✗	✗
CKA_EC_POINT	✗	✓	✗	✗	✗	✗	✗
CKA_VALUE	S	✗	✗	✗	✓ ²	✓ ²	✓ ²
CKA_VALUE_LEN	✗	✗	✗	✗	✓	✗	✓
CKA_CHECK_VALUE	✓	✓	✓	✓	✓	✓	✗

Attribute annotations

- [1] This attribute is partially supported by the firmware and must be explicitly set only to the default value.
- [2] Mandatory attribute.
- [3] **Client SDK 3 only.** 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] **Client SDK 3 only.** 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 \(p. 142\)](#) command from `cloudhsm_mgmt_util`. You can also derive a list of attributes and the constants that represent them by using the [listAttribute \(p. 133\)](#) 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 \(p. 221\)](#) 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

Apply this attribute to an exportable data key to specify that you can only wrap this key with keys marked as `CKA_TRUSTED`. Once you set `CKA_WRAP_WITH_TRUSTED` to true, the attribute becomes read-only and you cannot change or remove the attribute.

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:

Error Code	Description
CKR_TEMPLATE_INCONSISTENT	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.
CKR_ATTRIBUTE_TYPE_INVALID	You receive this error when you retrieve value for an attribute, which complies with the PKCS #11 specification, but is not supported by CloudHSM.
CKR_ATTRIBUTE_INCOMPLETE	You receive this error when you do not specify the mandatory attribute in the attribute template.
CKR_ATTRIBUTE_READ_ONLY	You receive this error when you specify a read-only attribute in the attribute template.

Code samples for the PKCS #11 library

The code samples on GitHub show you how to accomplish basic tasks using the PKCS #11 library.

Sample code prerequisites

Before running the samples, perform the following steps to set up your environment:

- Install and configure the [PKCS #11 library \(p. 324\)](#) for Client SDK 5 or Client SDK 3.
- Set up a [cryptographic user \(CU\) \(p. 59\)](#). Your application uses this HSM account to run the code samples on the HSM.

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.

- [Generate keys \(AES, RSA, EC\)](#)
- [List key attributes](#)
- [Encrypt and decrypt data with AES GCM](#)
- [Encrypt and decrypt data with AES_CTR](#)
- [Encrypt and decrypt data with 3DES](#)
- [Sign and verify data with RSA](#)
- [Derive keys using HMAC KDF](#)
- [Wrap and unwrap keys with AES using PKCS #5 padding](#)
- [Wrap and unwrap keys with AES using no padding](#)
- [Wrap and unwrap keys with AES using zero padding](#)
- [Wrap and unwrap keys with AES-GCM](#)
- [Wrap and unwrap keys with RSA](#)

OpenSSL Dynamic Engine

AWS CloudHSM offers two implementations of the OpenSSL Dynamic Engine: Client SDK 3 and Client SDK 5.

Client SDK 3 requires a client daemon to connect to the cluster. It supports:

- 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.

Client SDK 5 doesn't require a client daemon to connect to the cluster. It supports:

- RSA key generation for 2048, 3072, and 4096-bit keys.
- RSA sign/verify. Verification is offloaded to OpenSSL software.
- ECDSA sign/verify for P-256, P-384, and secp256k1 key types.

To generate EC keys that are interoperable with the OpenSSL engine, see [the section called "getCaviumPrivKey" \(p. 197\)](#).

For information about platform support for SDKs, see [the section called "Supported platforms" \(p. 314\)](#).

Topics

- [OpenSSL Dynamic Engine Client SDK 3 \(p. 348\)](#)
- [OpenSSL Dynamic Engine Client SDK 5 \(p. 350\)](#)

OpenSSL Dynamic Engine Client SDK 3

Prerequisites

For information about platform support, see [the section called "Supported platforms" \(p. 314\)](#).

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. To install and configure the AWS CloudHSM client, see [Install the client \(Linux\) \(p. 28\)](#). Then use the following command to start it.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 6

```
$ sudo service start cloudhsm-client
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

RHEL 6

```
$ sudo service start cloudhsm-client
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Install the OpenSSL Dynamic Engine for Client SDK 3

The following steps describe how to install and configure the AWS CloudHSM dynamic engine for OpenSSL. For information about upgrading, see [Upgrade your Client SDK 3 \(p. 317\)](#).

To install 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 ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

Amazon Linux 2

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

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

CentOS 6

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

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

CentOS 7

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

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

RHEL 6

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

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

RHEL 7

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

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

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-dyn_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-dyn_latest_amd64.deb
```

The OpenSSL engine is installed at /opt/cloudhsm/lib/libcloudhsm_openssl.so.

2. Use the following command to set an environment variable named n3fips_password that contains the credentials of a crypto user (CU).

```
$ export n3fips_password=<HSM user name>:<password>
```

Use the OpenSSL Dynamic Engine for Client SDK 3

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`.

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:

```
$ openssl s_server -cert server.crt -key server.key -engine cloudhsm
```

OpenSSL Dynamic Engine Client SDK 5

For Client SDK 5, you don't need to install or run a client daemon.

To install and configure the OpenSSL Dynamic 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-dyn-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el6.x86_64.rpm
```

Amazon Linux 2

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

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

CentOS 7

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

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

CentOS 8

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

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

RHEL 7

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

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

RHEL 8

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

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

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-dyn_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u18.04_amd64.deb
```

You have installed the shared library for the dynamic engine at /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so.

2. Set an environment variable with the credentials of a crypto user (CU). For information about creating CUs, see [Using CMU to manage users \(p. 61\)](#).

```
$ export CLOUDHSM_PIN=<HSM user name>:<password>
```

Note

Client SDK 5 introduces the CLOUDHSM_PIN environment variable for storing the credentials of the CU. In Client SDK 3 you store the CU credentials in the n3fips_password environment variable. Client SDK 5 supports both environment variables, but we recommend using CLOUDHSM_PIN.

3. Connect your installation of OpenSSL Dynamic Engine to the cluster. For more information, see [Connect to the Cluster \(p. 38\)](#).
4. Bootstrap the Client SDK 5. For more information, see [the section called “Bootstrap the Client SDK” \(p. 39\)](#).

Verify the OpenSSL Dynamic Engine for Client SDK 5

Use the following command to verify your installation of OpenSSL Dynamic Engine.

```
$ openssl engine -t cloudhsm
```

The following output verifies your configuration:

```
(cloudhsm) CloudHSM OpenSSL Engine  
[ available ]
```

JCE provider

The AWS CloudHSM JCE provider is a provider implementation built from the Java Cryptographic Extension (JCE) provider framework. The JCE provides a framework for performing cryptographic operations using the Java Development Kit (JDK). In this guide, the AWS CloudHSM JCE provider is sometimes referred to as the JCE provider. Use the JCE provider and the JDK to offload cryptographic operations to the HSM.

There are two versions of the JCE provider. For information on which version is best suited for your use case, see [the section called "Choose a Client SDK version" \(p. 258\)](#). For information about platform support, see [the section called "Supported platforms" \(p. 314\)](#).

Choose one of the following:

- [JCE provider Client SDK 3 \(p. 352\)](#)
- [JCE provider Client SDK 5 \(p. 369\)](#)

JCE provider Client SDK 3

Topics

- [Install and use the AWS CloudHSM JCE provider for Client SDK 3 \(p. 352\)](#)
- [Supported mechanisms for Client SDK 3 \(p. 357\)](#)
- [Supported Java attributes for Client SDK 3 \(p. 360\)](#)
- [Code samples for the AWS CloudHSM software library for Java for Client SDK 3 \(p. 366\)](#)
- [Using AWS CloudHSM KeyStore Java class for Client SDK 3 \(p. 367\)](#)

Install and use the AWS CloudHSM JCE provider for Client SDK 3

Before you can use the JCE provider, 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 JCE provider 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 \(Linux\) \(p. 28\)](#). After you install and configure the client, use the following command to start it.

Note that the JCE provider is supported only on Linux and compatible operating systems.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Topics

- [Installing the JCE provider \(p. 353\)](#)
- [Validating the installation \(p. 354\)](#)
- [Providing credentials to the JCE provider \(p. 356\)](#)
- [Key management basics in the JCE provider \(p. 356\)](#)

Installing the JCE provider

Use the following commands to download and install the JCE provider. This provider is supported only on Linux and compatible operating systems.

Note

For upgrading, see [Upgrade your Client SDK 3 \(p. 317\)](#).

Amazon Linux

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

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

Amazon Linux 2

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

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

CentOS 7

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

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

CentOS 8

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

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

RHEL 7

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

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

RHEL 8

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

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

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_u18.04_amd64.deb
```

After you run the preceding commands, you can find the following JCE provider 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-2.17.1.jar
- /opt/cloudhsm/java/log4j-core-2.17.1.jar
- /opt/cloudhsm/lib/libcaviumjca.so

Validating the installation

Perform basic operations on the HSM to validate the installation.

To validate JCE provider 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)

```
$ sudo yum install java-1.8.0-openjdk
```

Ubuntu

```
$ 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).

```
$ 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.

```
$ java8 -classpath "/opt/cloudhsm/java/*" org.junit.runner.JUnitCore
TestBasicFunctionality

JUnit version 4.11
.2018-08-20 17:53:48,514 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:33) - Adding provider.
2018-08-20 17:53:48,612 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:42) - Logging in.
2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:104) - Looking
for credentials in HsmCredentials.properties
2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:122) - Looking
for credentials in System.properties
2018-08-20 17:53:48,613 INFO [main] cfm2.LoginManager (LoginManager.java:130) - Looking
for credentials in System.env
SDK Version: 2.03
2018-08-20 17:53:48,655 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:54) - Generating AES Key with key size 256.
2018-08-20 17:53:48,698 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:63) - Encrypting with AES Key.
2018-08-20 17:53:48,705 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:84) - Deleting AES Key.
2018-08-20 17:53:48,707 DEBUG [main] TestBasicFunctionality
(TestBasicFunctionality.java:92) - Logging out.

Time: 0.205

OK (1 test)
```

Providing credentials to the JCE provider

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 -DHSM_PARTITION=PARTITION_1 -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
```

```
System.setProperty("HSM_PARTITION", "PARTITION_1");
System.setProperty("HSM_USER", "<HSM user name>");
System.setProperty("HSM_PASSWORD", "<password>");
```

- **Environment variables** – 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 JCE provider

The basics on key management in the JCE provider 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 JCE provider code examples at [Java samples \(p. 366\)](#).

Supported mechanisms for Client SDK 3

For information about the Java Cryptography Architecture (JCA) interfaces and engine classes supported by AWS CloudHSM, see the following topics.

Topics

- [Supported keys \(p. 357\)](#)
- [Supported ciphers \(p. 357\)](#)
- [Supported digests \(p. 359\)](#)
- [Supported hash-based message authentication code \(HMAC\) algorithms \(p. 359\)](#)
- [Supported sign/verify mechanisms \(p. 360\)](#)

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.

Note

Java library version 3.1 provides the ability to specify parameters in greater detail. For more information, see [Supported Java Attributes \(p. 360\)](#).

Supported ciphers

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

Algorithm	Mode	Padding	Notes
AES	CBC	AES/CBC/NoPadding AES/CBC/ PKCS5Padding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
AES	ECB	AES/ECB/NoPadding AES/ECB/ PKCS5Padding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> . Use Transformation AES.
AES	CTR	AES/CTR/NoPadding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .

Algorithm	Mode	Padding	Notes
AES	GCM	AES/GCM/NoPadding	<p>Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code>, <code>Cipher.WRAP_MODE</code>, and <code>Cipher.UNWRAP_MODE</code>.</p> <p>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 <code>Cipher.getIV()</code> to get the IV.</p>
AESWrap	ECB	AESWrap/ECB/ ZeroPadding AESWrap/ECB/ NoPadding AESWrap/ECB/ PKCS5Padding	<p>Implements <code>Cipher.WRAP_MODE</code>, and <code>Cipher.UNWRAP_MODE</code>. Use Transformation AES.</p>
DESede (Triple DES)	CBC	DESede/CBC/ NoPadding DESede/CBC/ PKCS5Padding	<p>Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code>.</p> <p>The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.</p>
DESede (Triple DES)	ECB	DESede/ECB/ NoPadding DESede/ECB/ PKCS5Padding	<p>Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code>.</p> <p>The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.</p>
RSA	ECB	RSA/ECB/NoPadding RSA/ECB/ PKCS1Padding	<p>Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code>.</p>

Algorithm	Mode	Padding	Notes
RSA	ECB	RSA/ECB/ OAEPPadding RSA/ECB/ OAEPWithSHA-1ANDMGF1Padding RSA/ECB/ OAEPPadding RSA/ECB/ OAEPWithSHA-224ANDMGF1Padding RSA/ECB/ OAEPWithSHA-256ANDMGF1Padding RSA/ECB/ OAEPWithSHA-384ANDMGF1Padding RSA/ECB/ OAEPWithSHA-512ANDMGF1Padding	Implements <code>Cipher.ENCRYPT_MODE</code> , <code>Cipher.DECRYPT_MODE</code> , <code>Cipher.WRAP_MODE</code> , <code>Cipher.UNWRAP_MODE</code> . OAEPPadding is OAEP with the SHA-1 padding type.
RSAAESWrap	ECB	OAEP_PADDING	Implements <code>Cipher.WRAP_Mode</code> and <code>Cipher.UNWRAP_MODE</code> .

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`

Supported Java attributes for Client SDK 3

This topic describes how to use a proprietary extension for the Java library version 3.1 to set key attributes. Use this extension to set supported key attributes and their values during these operations:

- Key generation
- Key import
- Key unwrap

Note

The extension for setting custom key attributes is an optional feature. If you already have code that functions in Java library version 3.0, you do not need to modify that code. Keys you create will continue to contain the same attributes as before.

Topics

- [Understanding attributes \(p. 361\)](#)
- [Supported attributes \(p. 361\)](#)
- [Setting attributes for a key \(p. 363\)](#)
- [Putting it all together \(p. 365\)](#)

Understanding attributes

You use key attributes to specify what actions are permitted on key objects, including public, private or secret keys. You define key attributes and values during key object creation operations.

However, the Java Cryptography Extension (JCE) does not specify how you should set values on key attributes, so most actions were permitted by default. In contrast, the PKCS# 11 standard defines a comprehensive set of attributes with more restrictive defaults. Starting with the Java library version 3.1, CloudHSM provides a proprietary extension that enables you to set more restrictive values for commonly used attributes.

Supported attributes

You can set values for the attributes listed in the table below. As a best practice, only set values for attributes you wish to make restrictive. If you don't specify a value, CloudHSM uses the default value specified in the table below. An empty cell in the Default Value columns indicates that there is no specific default value assigned to the attribute.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
CKA_TOKEN	FALSE	FALSE	FALSE	A permanent key which is replicated across all HSMs in the cluster and included in backups. CKA_TOKEN = FALSE implies a session key, which is only loaded onto one HSM and automatically erased when the connection to the HSM is broken.
CKA_LABEL				A user-defined string. It allows you to conveniently identify keys on your HSM.
CKA_EXTRACTABLE	TRUE		TRUE	True indicates you can export this key out of the HSM.
CKA_ENCRYPT	TRUE	TRUE		True indicates you can use the key to encrypt any buffer.
CKA_DECRYPT	TRUE		TRUE	True indicates you can use the key to decrypt any buffer. You generally

Attribute	Default Value			Notes
				set this to FALSE for a key whose CKA_WRAP is set to true.
CKA_WRAP	TRUE	TRUE		True indicates you can use the key to wrap another key. You will generally set this to FALSE for private keys.
CKA_UNWRAP	TRUE		TRUE	True indicates you can use the key to unwrap (import) another key.
CKA_SIGN	TRUE		TRUE	True indicates you can use the key to sign a message digest. This is generally set to FALSE for public keys and for private keys that you have archived.
CKA_VERIFY	TRUE	TRUE		True indicates you can use the key to verify a signature. This is generally set to FALSE for private keys.
CKA_PRIVATE	TRUE	TRUE	TRUE	True indicates that a user may not access the key until the user is authenticated. For clarity, users cannot access any keys on CloudHSM until they are authenticated, even if this attribute is set to FALSE.

Note

You get broader support for attributes in the PKCS#11 library. For more information, see [Supported PKCS #11 Attributes \(p. 336\)](#).

Setting attributes for a key

`CloudHsmKeyAttributesMap` is a [Java Map-like](#) object, which you can use to set attribute values for key objects. The methods for `CloudHsmKeyAttributesMap` function similar to the methods used for Java map manipulation.

To set custom values on attributes, you have two options:

- Use the methods listed in the following table
- Use builder patterns demonstrated later in this document

Attribute map objects support the following methods to set attributes:

Operation	Return Value	<code>CloudHsmKeyAttributesMap</code> method
Get the value of a key attribute for an existing key	Object (containing the value) or <code>null</code>	<code>get(keyAttribute)</code>
Populate the value of one key attribute	The previous value associated with key attribute, or <code>null</code> if there was no mapping for a key attribute	<code>put(keyAttribute, value)</code>
Populate values for multiple key attributes	N/A	<code>putAll(keyAttributesMap)</code>
Remove a key-value pair from the attribute map	The previous value associated with key attribute, or <code>null</code> if there was no mapping for a key attribute	<code>remove(keyAttribute)</code>

Note

Any attributes you do not explicitly specify are set to the defaults listed in the preceding table in the section called “[Supported attributes](#)” (p. 361).

Builder pattern example

Developers will generally find it more convenient to utilize classes through the Builder pattern. As examples:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

CloudHsmKeyAttributesMap keyAttributesSessionDecryptionKey =
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_LABEL, "ExtractableSessionKeyEncryptDecrypt")
        .put(CloudHsmKeyAttributes.CKA_WRAP, false)
        .put(CloudHsmKeyAttributes.CKA_UNWRAP, false)
        .put(CloudHsmKeyAttributes.CKA_SIGN, false)
        .put(CloudHsmKeyAttributes.CKA_VERIFY, false)
        .build();

CloudHsmKeyAttributesMap keyAttributesTokenWrappingKey =
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_LABEL, "TokenWrappingKey")
```

```
.put(CloudHsmKeyAttributes.CKA_TOKEN, true)
.put(CloudHsmKeyAttributes.CKA_ENCRYPT, false)
.put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
.put(CloudHsmKeyAttributes.CKA_SIGN, false)
.put(CloudHsmKeyAttributes.CKA_VERIFY, false)
.build();
```

Developers may also utilize pre-defined attribute sets as a convenient way to enforce best practices in key templates. As an example:

```
//best practice template for wrapping keys

CloudHsmKeyAttributesMap commonKeyAttrs = new CloudHsmKeyAttributesMap.Builder()
    .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, false)
    .put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
    .build();

// initialize a new instance of CloudHsmKeyAttributesMap by copying commonKeyAttrs
// but with an appropriate label

CloudHsmKeyAttributesMap firstKeyAttrs = new CloudHsmKeyAttributesMap(commonKeyAttrs);
firstKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, "key label");

// alternatively, putAll() will overwrite existing values to enforce conformance

CloudHsmKeyAttributesMap secondKeyAttrs = new CloudHsmKeyAttributesMap();
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_DECRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_ENCRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, "safe wrapping key");
secondKeyAttrs.putAll(commonKeyAttrs); // will overwrite CKA_DECRYPT to be FALSE
```

Setting attributes for a key pair

Use the Java class `CloudHsmKeyPairAttributesMap` to handle key attributes for a key pair. `CloudHsmKeyPairAttributesMap` encapsulates two `CloudHsmKeyAttributesMap` objects; one for a public key and one for a private key.

To set individual attributes for the public key and private key separately, you can use the `put()` method on corresponding `CloudHsmKeyAttributes` map object for that key. Use the `getPublic()` method to retrieve the attribute map for the public key, and use `getPrivate()` to retrieve the attribute map for the private key. Populate the value of multiple key attributes together for both public and private key pairs using the `putAll()` with a key pair attributes map as its argument.

Builder pattern example

Developers will generally find it more convenient to set key attributes through the Builder pattern. For example:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

//specify attributes up-front
CloudHsmKeyAttributesMap keyAttributes =
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_SIGN, false)
        .put(CloudHsmKeyAttributes.CKA_LABEL, "PublicCertSerial12345")
    .build();

CloudHsmKeyPairAttributesMap keyPairAttributes =
    new CloudHsmKeyPairAttributesMap.Builder()
```

```

.withPublic(keyAttributes)
.withPrivate(
    new CloudHsmKeyAttributesMap.Builder() //or specify them inline
        .put(CloudHsmKeyAttributes.CKA_LABEL, "PrivateCertSerial12345")
        .put(CloudHsmKeyAttributes.CKA_WRAP, FALSE)
        .build()
)
.build();

```

Note

For more information about this proprietary extension, see the [Javadoc](#) archive and the [sample](#) on GitHub. To explore the Javadoc, download and expand the archive.

Putting it all together

To specify key attributes with your key operations, follow these steps:

1. Instantiate `CloudHsmKeyAttributesMap` for symmetric keys or `CloudHsmKeyPairAttributesMap` for key pairs.
2. Define the attributes object from step 1 with the required key attributes and values.
3. Instantiate a `Cavium*ParameterSpec` class, corresponding to your specific key type, and pass into its constructor this configured attributes object.
4. Pass this `Cavium*ParameterSpec` object into a corresponding crypto class or method.

For reference, the following table contains the `Cavium*ParameterSpec` classes and methods which support custom key attributes.

Key Type	Parameter Spec Class	Example Constructors
Base Class	<code>CaviumKeyGenAlgorithmParameterSpec</code>	<code>CaviumKeyGenAlgorithmParameterSpec(CloudHsmKeyAttributesMap)</code>
DES	<code>CaviumDESKeyGenParameterSpec</code>	<code>CaviumDESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap keyAttributesMap)</code>
RSA	<code>CaviumRSAKeyGenParameterSpec</code>	<code>CaviumRSAKeyGenParameterSpec(int keysize, BigInteger publicExponent, CloudHsmKeyPairAttributesMap keyPairAttributesMap)</code>
Secret	<code>CaviumGenericSecretKeyGenParameterSpec</code>	<code>CaviumGenericSecretKeyGenParameterSpec(int size, CloudHsmKeyAttributesMap keyAttributesMap)</code>
AES	<code>CaviumAESKeyGenParameterSpec</code>	<code>CaviumAESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap keyAttributesMap)</code>
EC	<code>CaviumECGenParameterSpec</code>	<code>CaviumECGenParameterSpec(String stdName, CloudHsmKeyPairAttributesMap keyPairAttributesMap)</code>

Sample code: Generate and wrap a key

These brief code samples demonstrate the steps for two different operations: Key Generation and Key Wrapping:

```
// Set up the desired key attributes

KeyGenerator keyGen = KeyGenerator.getInstance("AES", "Cavium");
CaviumAESKeyGenParameterSpec keyAttributes = new CaviumAESKeyGenParameterSpec(
    256,
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_LABEL, "MyPersistentAESKey")
        .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, true)
        .put(CloudHsmKeyAttributes.CKA_TOKEN, true)
        .build()
);

// Assume we already have a handle to the myWrappingKey
// Assume we already have the wrappedBytes to unwrap

// Unwrap a key using Custom Key Attributes

CaviumUnwrapParameterSpec unwrapSpec = new
    CaviumUnwrapParameterSpec(myInitializationVector, keyAttributes);

Cipher unwrapCipher = Cipher.getInstance("AESWrap", "Cavium");
unwrapCipher.init(Cipher.UNWRAP_MODE, myWrappingKey, unwrapSpec);
Key unwrappedKey = unwrapCipher.unwrap(wrappedBytes, "AES", Cipher.SECRET_KEY);
```

Code samples for the AWS CloudHSM software library for Java for Client SDK 3

Prerequisites

Before running the samples, you must set up your environment:

- Install and configure the [Java Cryptographic Extension \(JCE\) provider \(p. 353\)](#) and the [AWS CloudHSM client package \(p. 28\)](#).
- Set up a valid [HSM user name and password \(p. 59\)](#). 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 provide credentials to the [JCE provider \(p. 356\)](#).

Code samples

The following code samples show you how to use the [AWS CloudHSM JCE provider \(p. 352\)](#) to perform basic tasks. More code samples are available on [GitHub](#).

- [Log in to an HSM](#)
- [Manage keys](#)
- [Generate an AES key](#)
- [Encrypt and decrypt with AES-GCM](#)
- [Encrypt and decrypt with AES-CTR](#)
- [Encrypt and decrypt with D3DES-ECB](#)
- [Wrap and unwrap keys with AES-GCM](#)
- [Wrap and unwrap keys with AES](#)

- Wrap and unwrap keys with RSA
- Use supported key attributes
- Enumerate keys in the key store
- Use the CloudHSM key store
- Sign messages in a multi-threaded sample
- Sign and Verify with EC Keys

Using AWS CloudHSM KeyStore Java class for Client SDK 3

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 Cryptographic Extension (JCE) provider 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 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:

```
KeyStore ks = KeyStore.getInstance("Cavium");
```

Load the special-purpose CloudHSM KeyStore as follows:

```
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. 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 provider, 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:

```
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:

```
ks.load(inputStream, password);
```

Using AWS 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.

```
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:

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
for(Enumeration<String> entry = ksAliases(); 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.

```
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.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
```

```
String alias = ks.getCertificateAlias(cert)
```

- **GetCertificateChain**

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.

```
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**

This API is currently unsupported with Client SDK 3.

- **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).

JCE provider Client SDK 5

Topics

- [Install and use the AWS CloudHSM JCE provider for Client SDK 5 \(p. 370\)](#)
- [Supported mechanisms for Client SDK 5 \(p. 373\)](#)

- [Supported Java attributes for Client SDK 5 \(p. 376\)](#)
- [Code samples for the AWS CloudHSM software library for Java for Client SDK 5 \(p. 380\)](#)
- [Using AWS CloudHSM KeyStore Java class for Client SDK 5 \(p. 380\)](#)

Install and use the AWS CloudHSM JCE provider for Client SDK 5

The JCE provider is compatible with OpenJDK 8 and OpenJDK11. You can download both from the [OpenJDK website](#).

Topics

- [Install the JCE provider \(p. 370\)](#)
- [Provide credentials to the JCE provider \(p. 372\)](#)
- [Key management basics in the JCE provider \(p. 373\)](#)

Install the JCE provider

Use the following commands to download and install the JCE provider.

Amazon Linux

Amazon Linux on x86_64 architecture:

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

```
$ sudo yum install ./cloudhsm-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

Amazon Linux 2 on x86_64 architecture:

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

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

Amazon Linux 2 on ARM64 architecture:

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

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

CentOS 7

CentOS 7 on x86_64 architecture:

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

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

CentOS 8

CentOS 8 on x86_64 architecture:

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

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

RHEL 7

RHEL 7 on x86_64 architecture:

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

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

RHEL 8

RHEL 8 on x86_64 architecture:

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

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

Ubuntu 18.04 LTS

Ubuntu 18.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-jce_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u18.04_amd64.deb
```

Windows Server 2016

For Windows Server 2016 on x86_64 architecture, open PowerShell as an administrator and run the following command:

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

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

Windows Server 2019

For Windows Server 2019 on x86_64 architecture, open PowerShell as an administrator and run the following command:

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

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

You must bootstrap Client SDK 5. For more information, see [the section called “Bootstrap the Client SDK” \(p. 39\)](#).

After you run the preceding commands, you can find the following JCE provider files:

Linux

- /opt/cloudhsm/java/cloudhsm-*version*.jar
- /opt/cloudhsm/bin/configure-jce
- /opt/cloudhsm/bin/jce-info

Windows

- C:\Program Files\Amazon\CloudHSM\java\cloudhsm-*version*.jar>
- C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe
- C:\Program Files\Amazon\CloudHSM\bin\jce_info.exe

Provide credentials to the JCE provider

Before your Java application can use an HSM, the HSM needs to first authenticate the application. HSMs authenticate using either an explicit login or implicit login method.

Explicit login – This method lets you provide AWS CloudHSM credentials directly in the application. It uses the method from the [AuthProvider](#), where you pass a CU username and password in the pin pattern. For more information, see [Login to an HSM](#) code example.

Implicit login – This method lets you set AWS CloudHSM credentials either in a new property file, system properties, or as environment variables.

- **System properties** – Set credentials through system properties when running your application. The following examples show two different ways that you can do this:

Linux

```
$ java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
```

```
System.setProperty("HSM_USER", "<HSM user name>");  
System.setProperty("HSM_PASSWORD", "<password>");
```

Windows

```
PS C:\> java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
```

```
System.setProperty("HSM_USER", "<HSM user name>");  
System.setProperty("HSM_PASSWORD", "<password>");
```

- **Environment variables** – Set credentials as environment variables.

Linux

```
$ export HSM_USER=<HSM user name>
$ export HSM_PASSWORD=<password>
```

Windows

```
PS C:\> $Env:HSM_USER=<HSM user name>
PS C:\> $Env:HSM_PASSWORD=<password>
```

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. System properties
2. Environment variables

Key management basics in the JCE provider

The basics on key management in the JCE provider 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 JCE provider code examples at [Java samples \(p. 380\)](#).

Supported mechanisms for Client SDK 5

For information about the Java Cryptography Architecture (JCA) interfaces and engine classes supported by AWS CloudHSM, see the following topics.

Topics

- [Supported keys \(p. 373\)](#)
- [Supported ciphers \(p. 373\)](#)
- [Supported digests \(p. 375\)](#)
- [Supported hash-based message authentication code \(HMAC\) algorithms \(p. 375\)](#)
- [Supported sign/verify mechanisms \(p. 376\)](#)

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).
- DESede (Triple DES)
- HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support.

Supported ciphers

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

Algorithm	Mode	Padding	Notes
AES	CBC	AES/CBC/NoPadding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
		AES/CBC/ PKCS5Padding	
AES	ECB	AES/ECB/ PKCS5Padding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
		AES/ECB/NoPadding	
AES	CTR	AES/CTR/NoPadding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
AES	GCM	AES/GCM/NoPadding	Implements <code>Cipher.WRAP_MODE</code> , <code>Cipher.UNWRAP_MODE</code> , <code>Cipher.ENCRYPT_MODE</code> , and <code>Cipher.DECRYPT_MODE</code> . 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 <code>Cipher.getIV()</code> to get the IV.
AESWrap	ECB	AESWrap/ECB/ NoPadding	Implements <code>Cipher.WRAP_MODE</code> and <code>Cipher.UNWRAP_MODE</code> .
		AESWrap/ECB/ PKCS5Padding	
		AESWrap/ECB/ ZeroPadding	
DESede (Triple DES)	CBC	DESede/CBC/ PKCS5Padding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
		DESede/CBC/ NoPadding	
DESede (Triple DES)	ECB	DESede/ECB/ NoPadding	Implements <code>Cipher.ENCRYPT_MODE</code> and <code>Cipher.DECRYPT_MODE</code> .
		DESede/ECB/ PKCS5Padding	
RSA	ECB	RSA/ECB/ PKCS1Padding	Implements <code>Cipher.WRAP_MODE</code> , <code>Cipher.UNWRAP_MODE</code> ,

Algorithm	Mode	Padding	Notes
		RSA/ECB/ OAEPPadding RSA/ECB/ OAEPwithSHA-1ANDMGF1Padding RSA/ECB/ OAEPwithSHA-224ANDMGF1Padding RSA/ECB/ OAEPwithSHA-256ANDMGF1Padding RSA/ECB/ OAEPwithSHA-384ANDMGF1Padding RSA/ECB/ OAEPwithSHA-512ANDMGF1Padding	<code>Cipher.ENCRYPT_MODE</code> , and <code>Cipher.DECRYPT_MODE</code> .
RSAAESWrap	ECB	RSAAESWrap/ECB/ OAEPPadding RSAAESWrap/ECB/ OAEPwithSHA-1ANDMGF1Padding RSAAESWrap/ECB/ OAEPwithSHA-224ANDMGF1Padding RSAAESWrap/ECB/ OAEPwithSHA-256ANDMGF1Padding RSAAESWrap/ECB/ OAEPwithSHA-384ANDMGF1Padding RSAAESWrap/ECB/ OAEPwithSHA-512ANDMGF1Padding	Implements <code>Cipher.WRAP_MODE</code> and <code>Cipher.UNWRAP_MODE</code> .

Supported digests

The AWS CloudHSM software library for Java supports the following message digests. With Client SDK 5, the data is hashed locally in software. This means there is no limit on the size of the data that can be hashed by the SDK.

- `SHA-1`
- `SHA-224`
- `SHA-256`
- `SHA-384`
- `SHA-512`

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. With Client SDK 5 and signature algorithms with hashing, the data is hashed locally in software before being sent to the HSM for the signature/verification. This means there is no limit on the size of the data that can be hashed by the SDK.

RSA Signature Types

- NONEwithRSA
- RSASSA-PSS
- SHA1withRSA
- SHA1withRSA/PSS
- SHA1withRSAandMGF1
- SHA224withRSA
- SHA224withRSAandMGF1
- SHA224withRSA/PSS
- SHA256withRSA
- SHA256withRSAandMGF1
- SHA256withRSA/PSS
- SHA384withRSA
- SHA384withRSAandMGF1
- SHA384withRSA/PSS
- SHA512withRSA
- SHA512withRSAandMGF1
- SHA512withRSA/PSS

ECDSA Signature Types

- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA

Supported Java attributes for Client SDK 5

This topic describes how to use a proprietary extension for the JCE provider to set key attributes. Use this extension to set supported key attributes and their values during these operations:

- Key generation
- Key import

For examples of how to use key attributes, see [the section called “Java samples” \(p. 380\)](#).

Topics

- [Understanding attributes \(p. 377\)](#)
- [Supported attributes \(p. 377\)](#)
- [Setting attributes for a key \(p. 379\)](#)

Understanding attributes

Use key attributes to specify what actions are permitted on key objects, including public, private or secret keys. Define key attributes and values during key object creation operations.

The Java Cryptography Extension (JCE) does not specify how you should set values on key attributes, so most actions were permitted by default. In contrast, the PKCS# 11 standard defines a comprehensive set of attributes with more restrictive defaults. Starting with the JCE provider 3.1, AWS CloudHSM provides a proprietary extension that enables you to set more restrictive values for commonly used attributes.

Supported attributes

You can set values for the attributes listed in the following table. As a best practice, only set values for attributes you wish to make restrictive. If you don't specify a value, AWS CloudHSM uses the default value specified in the table below. An empty cell in the Default Value columns indicates that there is no specific default value assigned to the attribute.

Attribute	Default Value			Notes
DECRYPT	TRUE		TRUE	True indicates you can use the key to decrypt any buffer. You generally set this to FALSE for a key whose WRAP is set to true.
ENCRYPT	TRUE	TRUE		True indicates you can use the key to encrypt any buffer.
EXTRACTABLE	TRUE		TRUE	True indicates you can export this key out of the HSM.
LABEL				A user-defined string. It allows you to conveniently identify keys on your HSM.
PRIVATE	TRUE	TRUE	TRUE	True indicates that a user may not access the key until the user is authenticated. For clarity, users cannot access any keys on AWS CloudHSM until they are

Attribute	Default Value			Notes
				authenticated, even if this attribute is set to FALSE.
SIGN	TRUE		TRUE	True indicates you can use the key to sign a message digest. This is generally set to FALSE for public keys and for private keys that you have archived.
TOKEN	FALSE	FALSE	FALSE	A permanent key which is replicated across all HSMs in the cluster and included in backups. TOKEN = FALSE implies an ephemeral key which is automatically erased when the connection to the HSM is broken or logged out.
UNWRAP	TRUE		TRUE	True indicates you can use the key to unwrap (import) another key.
VERIFY	TRUE	TRUE		True indicates you can use the key to verify a signature. This is generally set to FALSE for private keys.
WRAP	TRUE	TRUE		True indicates you can use the key to wrap another key. You will generally set this to FALSE for private keys.

Attribute	Default Value			Notes
WRAP_WITH_TRUSTED	FALSE		FALSE	True indicates a key can only be wrapped and unwrapped with keys that have the TRUSTED attribute set to true. Once a key has WRAP_WITH_TRUSTED set to true, that attribute is read-only and can't be set to false. To read about trust wrapping, see Using trusted keys to control key unwraps .

Note

You get broader support for attributes in the PKCS#11 library. For more information, see [Supported PKCS #11 Attributes \(p. 336\)](#).

Setting attributes for a key

`KeyAttributesMap` is a Java Map-like object, which you can use to set attribute values for key objects. The methods for `KeyAttributesMap` function similar to the methods used for Java map manipulation.

To set custom values on attributes, you have two options:

- Use the methods listed in the following table
- Use builder patterns demonstrated later in this document

Attribute map objects support the following methods to set attributes:

Operation	Return Value	<code>KeyAttributesMap</code> method
Get the value of a key attribute for an existing key	Object (containing the value) or <code>null</code>	<code>get(keyAttribute)</code>
Populate the value of one key attribute	The previous value associated with key attribute, or <code>null</code> if there was no mapping for a key attribute	<code>put(keyAttribute, value)</code>
Populate values for multiple key attributes	N/A	<code>putAll(keyAttributesMap)</code>
Remove a key-value pair from the attribute map	The previous value associated with key attribute, or <code>null</code> if there was no mapping for a key attribute	<code>remove(keyAttribute)</code>

Note

Any attributes you do not explicitly specify are set to the defaults listed in the preceding table in the section called "Supported attributes" (p. 377).

Setting attributes for a key pair

Use the Java class `KeyPairAttributesMap` to handle key attributes for a key pair.

`KeyPairAttributesMap` encapsulates two `KeyAttributesMap` objects; one for a public key and one for a private key.

To set individual attributes for the public key and private key separately, you can use the `put()` method on corresponding `KeyAttributes` map object for that key. Use the `getPublic()` method to retrieve the attribute map for the public key, and use `getPrivate()` to retrieve the attribute map for the private key. Populate the value of multiple key attributes together for both public and private key pairs using the `putAll()` with a key pair attributes map as its argument.

Code samples for the AWS CloudHSM software library for Java for Client SDK 5

Prerequisites

Before running the samples, you must set up your environment:

- Install and configure the [Java Cryptographic Extension \(JCE\) provider \(p. 370\)](#).
- Set up a valid [HSM user name and password \(p. 59\)](#). 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 provide credentials to the [JCE provider \(p. 372\)](#).

Code samples

The following code samples show you how to use the [AWS CloudHSM JCE provider \(p. 352\)](#) to perform basic tasks. More code samples are available on [GitHub](#).

- [Log in to an HSM](#)
- [Manage keys](#)
- [Generate Symmetric Keys](#)
- [Generate Asymmetric Keys](#)
- [Encrypt and decrypt with AES-GCM](#)
- [Encrypt and decrypt with AES-CTR](#)
- [Encrypt and decrypt with DESede-ECB](#)
- [Sign and Verify with RSA Keys](#)
- [Sign and Verify with EC Keys](#)
- [Use supported key attributes](#)
- [Use the CloudHSM key store](#)

Using AWS CloudHSM KeyStore Java class for Client SDK 5

The AWS CloudHSM `KeyStore` class provides a special-purpose PKCS12 key store. This key store can store certificates along with your key data and correlate them to key data stored on AWS CloudHSM. 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](#).

Note

Because certificates are public information, and to maximize storage capacity for cryptographic keys, AWS CloudHSM does not support storing certificates on HSMs.

Choosing the appropriate key store

The AWS CloudHSM Java Cryptographic Extension (JCE) provider offers a special-purpose AWS CloudHSM KeyStore. The AWS CloudHSM KeyStore class supports offloading key operations to the HSM, local storage of certificates and certificate-based operations.

Load the special-purpose CloudHSM KeyStore as follows:

```
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. 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 provider, 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:

```
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:

```
ks.load(inputStream, password);
```

Using AWS CloudHSM KeyStore

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 an new empty key store.

```
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:

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
for(Enumeration<String> entry = ksAliases(); 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 using the *destroy* method of the [Destroyable](#) interface.

```
((Destroyable) key).destroy();
```

- **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.

```
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.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
String alias = ks.getCertificateAlias(cert);
```

- **GetCertificateChain**

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.

```
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

This API is currently unsupported with Client SDK 5.

- `SetKeyEntry` with `Key` object

Assigns the given key to the given alias and stores it inside the HSM. If the key does not already exist inside the HSM, it will be 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).

Cryptography API: Next Generation (CNG) and key storage providers (KSP) for Microsoft Windows

The AWS CloudHSM client for Windows includes CNG and KSP providers.

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 choose to create a new private key for your certificate authority (CA), you can choose the KSP that will manage key storage. When you configure the AD CS role, you can choose this KSP. For more information, see [Create Windows Server CA \(p. 449\)](#).

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, the AWS CloudHSM implementation of CNG provides the following functionality:

- **Cryptographic Primitives** - enable you to perform fundamental cryptographic operations.
- **Key Import and Export** - enables you to import and export 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.

Topics

- [Verify the KSP and CNG Providers for Windows \(p. 383\)](#)
- [Windows AWS CloudHSM prerequisites \(p. 385\)](#)
- [Associate a AWS CloudHSM key with a certificate \(p. 386\)](#)
- [Code sample for CNG provider \(p. 387\)](#)

Verify the KSP and CNG Providers for Windows

The 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 \(Windows\) \(p. 30\)](#).

Configure and run the Windows AWS CloudHSM client

To start the Windows CloudHSM client, you must first satisfy the [Prerequisites \(p. 385\)](#). 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 \(p. 238\)](#) 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
```

To verify that the KSP and CNG providers are installed on your Windows Server EC2 instance, you should see the following entries in the list:

```
Cavium CNG Provider  
Cavium Key Storage Provider
```

If the CNG provider is missing, run the following command.

```
C:\Program Files\Amazon\CloudHSM>cng_config.exe -register
```

If the KSP provider 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.

```
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 \(p. 60\)](#) (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 \(p. 60\)](#) (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. To create a new CU, use [createUser \(p. 115\)](#). To find existing CUs, use [listUsers \(p. 134\)](#).

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:  
72000000047f7f7a9d41851b4e000000000004Issuer: CN=Enterprise-CANotBefore: 10/8/2019 11:50  
AM NotAfter: 11/8/2020 12:00 PMSubject: CN=www.example.com, OU=Certificate Management,  
O=Information Technology, L=Seattle, S=Washington, C=USNon-root CertificateCert  
Hash(sh1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45 75 bc 65No key provider  
information Simple container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c Unique
```

```
container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
```

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 \(p. 152\)](#). 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.

```
$> import_key.exe -RSA CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c  
Successfully opened Microsoft Software Key Storage Provider : 0NCryptOpenKey failed :  
80090016
```

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 "72000000047f7f7a9d41851b4e000000000004"my "Personal"  
  
===== Certificate 1 =====Serial Number:  
72000000047f7f7a9d41851b4e000000000004  
Issuer: CN=Enterprise-CA  
NotBefore: 10/8/2019 11:50 AM  
NotAfter: 11/8/2020 12:00 PM  
Subject: CN=www.example.com, OU=Certificate Management, O=Information Technology,  
L=Seattle, S=Washington, C=US  
Non-root CertificateCert Hash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45 75  
bc 65  
SDK Version: 3.0  
Key Container = CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c  
Provider = Cavium Key Storage ProviderPrivate key is NOT exportableEncryption test  
passedCertUtil: -repairstore command completed successfully.
```

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 CNG provider

** Example code only - Not for production use **

This sample code is for illustrative purposes only. Do not run this code in production.

The following sample shows how to enumerate the registered cryptographic providers on your system to find the CNG provider installed with CloudHSM client for Windows. 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 \(p. 385\)](#).

```

// CloudHsmCngExampleConsole.cpp : Console application that demonstrates CNG capabilities.
// This example contains the following functions.
//
// VerifyProvider()           - Enumerate the registered providers and retrieve Cavium KSP
// and CNG providers.
// GenerateKeyPair()          - Create an RSA key pair.
// SignData()                 - Sign and verify data.
//

#include "stdafx.h"
#include <Windows.h>

#ifndef NT_SUCCESS
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)
#endif

#define CAVIUM_CNG_PROVIDER L"Cavium CNG Provider"
#define CAVIUM_KEYSTORE_PROVIDER L"Cavium Key Storage Provider"

// Enumerate the registered providers and determine whether the Cavium CNG provider
// and the Cavium KSP provider exist.
//
bool VerifyProvider()
{
    NTSTATUS status;
    ULONG cbBuffer = 0;
    PCRYPT_PROVIDERS pBuffer = NULL;
    bool foundCng = false;
    bool foundKeystore = false;

    // Retrieve information about the registered providers.
    // cbBuffer - the size, in bytes, of the buffer pointed to by pBuffer.
    // pBuffer - pointer to a buffer that contains a CRYPT_PROVIDERS structure.
    status = BCryptEnumRegisteredProviders(&cbBuffer, &pBuffer);

    // If registered providers exist, enumerate them and determine whether the
    // Cavium CNG provider and Cavium KSP provider have been registered.
    if (NT_SUCCESS(status))
    {
        if (pBuffer != NULL)
        {
            for (ULONG i = 0; i < pBuffer->cProviders; i++)
            {
                // Determine whether the Cavium CNG provider exists.
                if (wcscmp(CAVIUM_CNG_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
                {
                    printf("Found %S\n", CAVIUM_CNG_PROVIDER);
                    foundCng = true;
                }

                // Determine whether the Cavium KSP provider exists.
                else if (wcscmp(CAVIUM_KEYSTORE_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
                {
                    printf("Found %S\n", CAVIUM_KEYSTORE_PROVIDER);
                    foundKeystore = true;
                }
            }
        }
    }
    else
    {
        printf("BCryptEnumRegisteredProviders failed with error code 0x%08x\n", status);
    }
}

// Free memory allocated for the CRYPT_PROVIDERS structure.

```

```

        if (NULL != pBuffer)
        {
            BCryptFreeBuffer(pBuffer);
        }

        return foundCng == foundKeystore == true;
    }

    // Generate an asymmetric key pair. As used here, this example generates an RSA key pair
    // and returns a handle. The handle is used in subsequent operations that use the key
    // pair.
    // The key material is not available.
    //
    // The key pair is used in the SignData function.
    //
NTSTATUS GenerateKeyPair(BCRYPT_ALG_HANDLE hAlgorithm, BCRYPT_KEY_HANDLE *hKey)
{
    NTSTATUS status;

    // Generate the key pair.
    status = BCryptGenerateKeyPair(hAlgorithm, hKey, 2048, 0);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptGenerateKeyPair failed with code 0x%08x\n", status);
        return status;
    }

    // Finalize the key pair. The public/private key pair cannot be used until this
    // function is called.
    status = BCryptFinalizeKeyPair(*hKey, 0);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptFinalizeKeyPair failed with code 0x%08x\n", status);
        return status;
    }

    return status;
}

// Sign and verify data using the RSA key pair. The data in this function is hardcoded
// and is for example purposes only.
//
NTSTATUS SignData(BCRYPT_KEY_HANDLE hKey)
{
    NTSTATUS status;
    PBYTE sig;
    ULONG sigLen;
    ULONG resLen;
    BCRYPT_PKCS1_PADDING_INFO pInfo;

    // Hardcode the data to be signed (for demonstration purposes only).
    PBYTE message = (PBYTE)"d83e7716bed8a20343d8dc6845e57447";
    ULONG messageLen = strlen((char*)message);

    // Retrieve the size of the buffer needed for the signature.
    status = BCryptSignHash(hKey, NULL, message, messageLen, NULL, 0, &sigLen, 0);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptSignHash failed with code 0x%08x\n", status);
        return status;
    }

    // Allocate a buffer for the signature.
    sig = (PBYTE)HeapAlloc(GetProcessHeap(), 0, sigLen);
    if (sig == NULL)
    {

```

```

        return -1;
    }

    // Use the SHA256 algorithm to create padding information.
    pInfo.pszAlgId = BCRYPT_SHA256_ALGORITHM;

    // Create a signature.
    status = BCryptSignHash(hKey, &pInfo, message, messageLen, sig, sigLen, &resLen,
BCRYPT_PAD_PKCS1);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptSignHash failed with code 0x%08x\n", status);
        return status;
    }

    // Verify the signature.
    status = BCryptVerifySignature(hKey, &pInfo, message, messageLen, sig, sigLen,
BCRYPT_PAD_PKCS1);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptVerifySignature failed with code 0x%08x\n", status);
        return status;
    }

    // Free the memory allocated for the signature.
    if (sig != NULL)
    {
        HeapFree(GetProcessHeap(), 0, sig);
        sig = NULL;
    }

    return 0;
}

// Main function.
// 
int main()
{
    NTSTATUS status;
    BCRYPT_ALG_HANDLE hRsaAlg;
    BCRYPT_KEY_HANDLE hKey = NULL;

    // Enumerate the registered providers.
    printf("Searching for Cavium providers...\n");
    if (VerifyProvider() == false) {
        printf("Could not find the CNG and Keystore providers\n");
        return 1;
    }

    // Get the RSA algorithm provider from the Cavium CNG provider.
    printf("Opening RSA algorithm\n");
    status = BCryptOpenAlgorithmProvider(&hRsaAlg, BCRYPT_RSA_ALGORITHM, CAVIUM_CNG_PROVIDER,
0);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptOpenAlgorithmProvider RSA failed with code 0x%08x\n", status);
        return status;
    }

    // Generate an asymmetric key pair using the RSA algorithm.
    printf("Generating RSA Keypair\n");
    GenerateKeyPair(hRsaAlg, &hKey);
    if (hKey == NULL)
    {
        printf("Invalid key handle returned\n");
        return 0;
    }
}

```

```
    }
    printf("Done!\n");

    // Sign and verify [hardcoded] data using the RSA key pair.
    printf("Sign/Verify data with key\n");
    SignData(hKey);
    printf("Done!\n");

    // Remove the key handle from memory.
    status = BCryptDestroyKey(hKey);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptDestroyKey failed with code 0x%08x\n", status);
        return status;
    }

    // Close the RSA algorithm provider.
    status = BCryptCloseAlgorithmProvider(hRsaAlg, NULL);
    if (!NT_SUCCESS(status))
    {
        printf("BCryptCloseAlgorithmProvider RSA failed with code 0x%08x\n", status);
        return status;
    }

    return 0;
}
```

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 security with SSL/TLS offload in AWS CloudHSM \(p. 392\)](#)
- [Configure Windows Server as a certificate authority \(CA\) with AWS CloudHSM \(p. 448\)](#)
- [Oracle database transparent data encryption \(TDE\) with AWS CloudHSM \(p. 451\)](#)
- [Use Microsoft SignTool with AWS CloudHSM to sign files \(p. 455\)](#)
- [Java Keytool and Jarsigner \(p. 459\)](#)
- [Other third-party vendor integrations \(p. 477\)](#)

Improve your web server 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

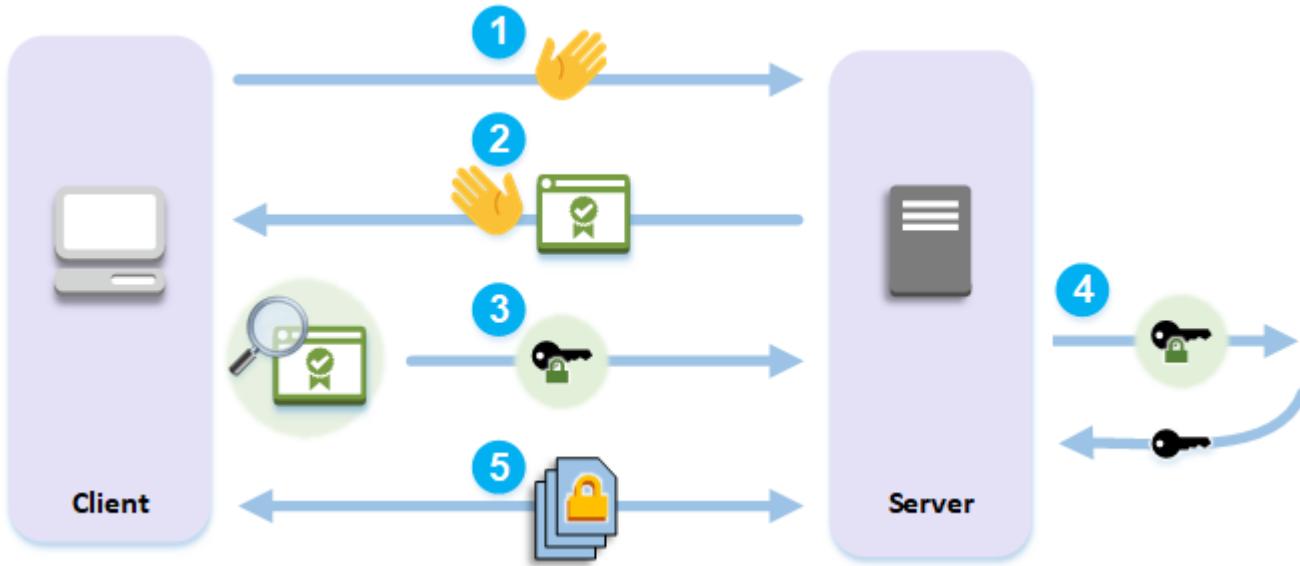
- [How SSL/TLS offload with AWS CloudHSM works \(p. 392\)](#)
- [Tutorial: Using SSL/TLS offload with AWS CloudHSM on Linux \(p. 393\)](#)
- [Tutorial: Using SSL/TLS offload with AWS CloudHSM on Windows \(p. 436\)](#)

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 \(p. 393\)](#)
- [Tutorial: Using SSL/TLS offload with AWS CloudHSM on Windows \(p. 436\)](#)

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 \(p. 394\)](#)

- [Step 1: Set up the prerequisites \(p. 394\)](#)
- [Step 2: Generate or import a private key and SSL/TLS certificate \(p. 398\)](#)
- [Step 3: Configure the web server \(p. 404\)](#)
- [Step 4: Enable HTTPS traffic and verify the certificate \(p. 430\)](#)
- [Step 5 \(Optional\): Add a load balancer with Elastic Load Balancing \(p. 432\)](#)

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 \(p. 347\)](#) 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 \(p. 394\)](#).

Step 1: Set up the prerequisites

Different platforms require different prerequisites. Use the prerequisites section below that matches your platform.

Topics

- [Prerequisites for Client SDK 5 \(p. 394\)](#)
- [Prerequisites for Client SDK 3 \(p. 397\)](#)

Prerequisites for Client SDK 5

To set up web server SSL/TLS offload with Client SDK 5, you need the following:

- An active AWS CloudHSM cluster with at least two hardware security modules (HSM)

Note

You can use a single HSM cluster, but you must first disable client key durability. For more information, see [Manage Client Key Durability Settings \(p. 90\)](#) and [Client SDK 5 Configure Tool \(p. 245\)](#).

- An Amazon EC2 instance running a Linux operating system with the following software installed:
 - A web server (either NGINX or Apache)
 - The OpenSSL Dynamic Engine for Client SDK 5
 - A [crypto user \(p. 60\)](#) (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. Install and configure the OpenSSL Dynamic Engine for AWS CloudHSM. For more information about installing OpenSSL Dynamic Engine, see [OpenSSL Dynamic Engine for Client SDK 5 \(p. 350\)](#).
2. On an EC2 Linux instance that has access to your cluster, install either NGINX or Apache web server:

Amazon Linux

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd24 mod24_ssl
```

Amazon Linux 2

- For information on how to download the latest version of NGINX on Amazon Linux 2, see the [NGINX website](#).

The latest version of NGINX available for Amazon Linux 2 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 7

- For information on how to download the latest version of NGINX on CentOS 7, see the [NGINX website](#).

The latest version of NGINX available for CentOS 7 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 7

- For information on how to download the latest version of NGINX on Red Hat 7, see the [NGINX website](#).

The latest version of NGINX available for Red Hat 7 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Ubuntu 18.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

3. Use CloudHSM Management Utility (CMU) to create a CU. For more information about managing HSM users, see [Understanding HSM User Management with CMU \(p. 61\)](#).

Tip

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 \(p. 398\)](#).

Notes

- To use Security-Enhanced Linux (SELinux) and web servers, you must allow outbound TCP connections on port 2223, which is the port Client SDK 5 uses to communicate with the HSM.
- To create and activate a cluster and give an EC2 instance access to the cluster, complete the steps in [Getting Started with AWS CloudHSM \(p. 10\)](#). The getting started offers step-by-step instruction for creating an active cluster with one HSM and an Amazon EC2 client instance with the Client SDK 3 command line tools. You can use this client instance as your web server.
- To avoid disabling client key durability, add more than one HSM to your cluster. For more information, see [Adding an HSM \(p. 42\)](#).
- To connect to your client instance, you can use SSH or PuTTY. 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.

Prerequisites for Client SDK 3

To set up web server SSL/TLS offload with Client SDK 3, 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 \(p. 60\)](#) (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 \(p. 10\)](#). 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.
3. On an EC2 Linux instance that has access to your cluster, install either NGINX or Apache web server:

Amazon Linux

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd24 mod24_ssl
```

Amazon Linux 2

- NGINX version 1.19 is the latest version of NGINX compatible with the Client SDK 3 engine on Amazon Linux 2.

For more information and to download NGINX version 1.19, see the [NGINX website](#).

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 7

- NGINX version 1.19 is the latest version of NGINX compatible with the Client SDK 3 engine on CentOS 7.

For more information and to download NGINX version 1.19, see the [NGINX website](#).

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 7

- NGINX version 1.19 is the latest version of NGINX compatible with the Client SDK 3 engine on Red Hat 7.

For more information and to download NGINX version 1.19, see the [NGINX website](#).

- Apache

```
$ sudo yum install httpd mod_ssl
```

Ubuntu 16.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

4. (Optional) Add more HSMs to your cluster. For more information, see [Adding an HSM \(p. 42\)](#).
5. Use cloudfsm_mgmt_util to create a CU. For more information, see [Managing HSM users \(p. 59\)](#). 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 \(p. 398\)](#).

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 generate a private key in a HSM. You use the private key to create a certificate signing request (CSR), which you use to create the SSL/TLS certificate.
- If you already have a private key and corresponding certificate, you import the private key into a HSM.

Regardless of which of the preceding methods you choose, you export a *fake PEM private key* from the HSM, which is a private key file in PEM format which contains a reference to the private key stored on the HSM (it's not the actual private key). Your web server uses the fake PEM private key file to identify the private key on the HSM during SSL/TLS offload.

Do one of the following:

- [Generate a private key and certificate \(p. 399\)](#)
- [Import an existing private key and certificate \(p. 402\)](#)

Generate a private key and certificate

Generate a private key

This section shows you how to generate a keypair using the [Key Management Utility \(KMU\) \(p. 152\)](#) from Client SDK 3. Once you have a key pair generated inside the HSM, you can export it as a fake PEM file, and generate the corresponding certificate.

Private keys generated with the Key Management Utility (KMU) can be used with both Client SDK 3 and Client SDK 5.

Install and configure the Key Management Utility (KMU)

1. Connect to your client instance.
2. [Install and Configure \(p. 28\)](#) Client SDK 3
3. Run the following command to start the AWS CloudHSM client.

Amazon Linux

```
$ sudo start cloudfsm-client
```

Amazon Linux 2

```
$ sudo service cloudfsm-client start
```

CentOS 7

```
$ sudo service cloudfsm-client start
```

CentOS 8

```
$ sudo service cloudfsm-client start
```

RHEL 7

```
$ sudo service cloudfsm-client start
```

RHEL 8

```
$ sudo service cloudfsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudfsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

- Run the following command to start the key_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

- 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).

```
Command: loginHSM -u CU -s <user name> -p <password>>
```

Generate a Private Key

Depending on your use case, you can either generate an RSA or an EC key pair. Do one of the following:

- To generate an RSA private key on an HSM

Use the genRSAKeyPair command to generate an RSA key pair. This example generates an RSA key pair with a modulus of 2048, a public exponent of 65537, and a label of *tls_rsa_keypair*.

```
Command: genRSAKeyPair -m 2048 -e 65537 -l tls_rsa_keypair
```

If the command was successful, you should see the following output indicating that you've successfully generated an RSA key pair.

```
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS
Cfm3GenerateKeyPair:    public key handle: 7    private key handle: 8
Cluster Status:
Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

- To generate an EC private key on an HSM

Use the genECCKeyPair command to generate an EC key pair. This example generates an EC key pair with a curve ID of 2 (corresponding to the NID_X9_62_prime256v1 curve) and a label of *tls_ec_keypair*.

```
Command: genECCKeyPair -i 2 -l tls_ec_keypair
```

If the command was successful, you should see the following output indicating that you've successfully generated an EC key pair.

```
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS
Cfm3GenerateKeyPair:    public key handle: 7    private key handle: 8
Cluster Status:
Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

Export a fake PEM private key file

Once you have a private key on the HSM, you must export a fake PEM private key file. This file does not contain the actual key data, but it allows the OpenSSL Dynamic Engine to identify the private key on the HSM. You can then use the private key to create a certificate signing request (CSR) and sign the CSR to create the certificate.

Note

Fake PEM files generated with the Key Management Utility (KMU) can be used with both Client SDK 3 and Client SDK 5.

Identify the key handle that corresponds to the key that you would like to export as a fake PEM, then 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 generated private key. This handle was generated by one of the key generation commands 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 your fake PEM key will be written to.

```
Command: getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
```

Exit

Run the following command to stop the key_mgmt_util.

```
Command: exit
```

You should now have a new file on your system, located at the path specified by <*web_server_fake_PEM.key*> in the preceding command. This file is the fake PEM private key file.

Generate a self-signed certificate

Once you have generated a fake PEM private key, you can use this file to generate a certificate signing request (CSR) and certificate.

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 to them and use signed SSL/TLS certificate that they provide you in your web server for HTTPS.

As an alternative to using a CA, you can use the AWS CloudHSM OpenSSL Dynamic Engine 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.

Install and configure the OpenSSL Dynamic Engine

1. Connect to your client instance.
2. To install and configure, do one of the following:
 - [the section called “Client SDK 5” \(p. 350\)](#)
 - [the section called “Client SDK 3” \(p. 348\)](#)

Generate a certificate

1. Obtain a copy of your fake PEM file generated in an earlier step.

2. Create a CSR

Run the following command to use the AWS CloudHSM OpenSSL Dynamic Engine 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 clouhdsm -new -key <web_server_fake_PEM.key> -  
out <web_server.csr>
```

3. Create a self-signed certificate

Run the following command to use the AWS CloudHSM OpenSSL Dynamic Engine 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 clouhdsm -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 \(p. 404\)](#).

Import an existing private key and certificate

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 following the steps in this section.

Note

Some notes on private key imports and Client SDK compatibility:

- Importing an existing private key requires Client SDK 3.
- You can use private keys from Client SDK 3 with Client SDK 5.
- OpenSSL Dynamic Engine for Client SDK 3 does not support the latest Linux platforms, but the implementation of OpenSSL Dynamic Engine for Client SDK 5 does. You can import an existing private key using the Key Management Utility (KMU) provided with Client SDK 3, then *use that private key* and the implementation of OpenSSL Dynamic Engine with Client SDK 5 to support SSL/TLS offload on the latest Linux platforms.

To import an existing private key into an HSM with Client SDK 3

1. Connect to your Amazon EC2 client instance. If necessary, copy your existing private key and certificate to the instance.
2. [Install and Configure \(p. 28\)](#) Client SDK 3
3. Run the following command to start the AWS CloudHSM client.

Amazon Linux

```
$ sudo start clouhdsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

4. Run the following command to start the key_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

5. 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).

```
Command: loginHSM -u CU -s <user name> -p <password>
```

6. 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.

```
Command: genSymKey -t 31 -s 16 -sess -l wrapping_key_for_import

Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Created. Key Handle: 6
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

- b. 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.

```
Command: importPrivateKey -f <web_server_existing.key> -l <web_server_imported_key>
-w <wrapping_key_handle>

BER encoded key length is 1219
Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS
Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS
Private Key Unwrapped. Key Handle: 8
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

7. 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.

```
Command: getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
```

8. Run the following command to stop key_mgmt_util.

```
Command: exit
```

After you complete these steps, go to [Step 3: Configure the web server \(p. 404\)](#).

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 \(p. 398\)](#). Remember to backup your existing certificates and keys before you start. This will finish setting up your Linux web server software for SSL/TLS offload with AWS CloudHSM.

Complete the steps from one of the following sections.

Topics

- [Configure NGINX web server \(p. 404\)](#)
- [Configure Apache web server \(p. 418\)](#)

Configure NGINX web server

Use this section to configure NGINX on supported platforms.

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.

```
$ 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. Update the NGINX configuration.

Note

Each cluster can support a maximum of 1000 NGINX worker processes across all NGINX web servers.

Amazon Linux

Use a text editor to edit the /etc/nginx/nginx.conf file. This requires Linux root permissions. At the top of the file, add the following lines:

- If using Client SDK 3

```
ssl_engine cloudhsm;
env n3fips_password;
```

- If using Client SDK 5

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
```

```

ssl_session_timeout 10m;
ssl_protocols TLSv1.2;
ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA";
ssl_prefer_server_ciphers on;

# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

location / {
}

error_page 404 /404.html;
location = /40x.html {
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
}
}

```

Amazon Linux 2

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

- If using Client SDK 3

```

ssl_engine cloudhsm;
env n3fips_password;

```

- If using Client SDK 5

```

ssl_engine cloudhsm;
env CLOUDHSM_PIN;

```

Then add the following to the TLS section of the file:

```

# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-

```

```
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-  
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-  
AES256-SHA:ECDHE-ECDSA-AES128-SHA";  
    ssl_prefer_server_ciphers on;  
  
    # Load configuration files for the default server block.  
    include /etc/nginx/default.d/*.conf;  
  
    location / {  
    }  
  
    error_page 404 /404.html;  
    location = /40x.html {  
    }  
  
    error_page 500 502 503 504 /50x.html;  
    location = /50x.html {  
    }  
}
```

CentOS 7

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

- If using Client SDK 3

```
ssl_engine cloudhsm;  
env n3fips_password;
```

- If using Client SDK 5

```
ssl_engine cloudhsm;  
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.  
server {  
    listen      443 ssl http2 default_server;  
    listen      [::]:443 ssl http2 default_server;  
    server_name _;  
    root        /usr/share/nginx/html;  
  
    ssl_certificate "/etc/pki/nginx/server.crt";  
    ssl_certificate_key "/etc/pki/nginx/private/server.key";  
    # It is *strongly* recommended to generate unique DH parameters  
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048  
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";  
    ssl_session_cache shared:SSL:1m;  
    ssl_session_timeout 10m;  
    ssl_protocols TLSv1.2;  
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-  
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-  
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-  
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-  
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-  
AES256-SHA:ECDHE-ECDSA-AES128-SHA";  
    ssl_prefer_server_ciphers on;
```

```
# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

location / {
}

error_page 404 /404.html;
location = /40x.html {
}

error_page 500 502 503 504 /50x.html;
location = /50x.html {
}
}
```

CentOS 8

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {

    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Red Hat 7

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

- If using Client SDK 3

```
ssl_engine cloudhsm;
env n3fips_password;
```

- If using Client SDK 5

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Red Hat 8

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Ubuntu 16.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env n3fips_password;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
```

```

server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}

```

Ubuntu 18.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```

ssl_engine cloudhsm;
env CLOUDHSM_PIN;

```

Then add the following to the TLS section of the file:

```

# Settings for a TLS enabled server.
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_protocols TLSv1.2 TLSv1.3;

```

```
ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-  
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-  
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-  
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-  
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-  
AES256-SHA:ECDHE-ECDSA-AES128-SHA";  
ssl_prefer_server_ciphers on;  
  
# Load configuration files for the default server block.  
include /etc/nginx/default.d/*.conf;  
  
location / {  
}  
  
error_page 404 /404.html;  
location = /40x.html {  
}  
  
error_page 500 502 503 504 /50x.html;  
location = /50x.html {  
}  
}
```

Save the file.

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.

```
$ 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 7

No action required.

CentOS 8

1. Back up the `nginx.service` file.

```
$ 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
```

Red Hat 7

No action required.

Red Hat 8

1. Back up the `nginx.service` file.

```
$ 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
```

Ubuntu 16.04

1. Back up the `nginx.service` file.

```
$ 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
```

Ubuntu 18.04

1. Back up the `nginx.service` file.

```
$ 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:

```
$ 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.

10. Configure the NGINX environment.

Note

Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the

`n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

Amazon Linux

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace `<CU user name>` and `<password>` with the CU credentials.

Save the file.

Amazon Linux 2

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace `<CU user name>` and `<password>` with the CU credentials.

Save the file.

CentOS 7

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace `<CU user name>` and `<password>` with the CU credentials.

Save the file.

CentOS 8

Open the /etc/sysconfig/nginx file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Red Hat 7

Open the /etc/sysconfig/nginx file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Red Hat 8

Open the /etc/sysconfig/nginx file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 16.04 LTS

Open the /etc/sysconfig/nginx file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

```
n3fips_password=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 18.04 LTS

Open the /etc/sysconfig/nginx file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

11. Start the NGINX web server.

Amazon Linux

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions.
Add the Cryptography User (CU) credentials:

```
$ sudo service nginx start
```

Amazon Linux 2

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

CentOS 7

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

CentOS 8

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Red Hat 7

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the systemd configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Red Hat 8

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the systemd configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 16.04

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the systemd configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 18.04

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

12. (Optional) Configure your platform to start NGINX at start-up.

Amazon Linux

```
$ sudo chkconfig nginx on
```

Amazon Linux 2

```
$ sudo systemctl enable nginx
```

CentOS 7

No action required.

CentOS 8

```
$ sudo systemctl enable nginx
```

Red Hat 7

No action required.

Red Hat 8

```
$ sudo systemctl enable nginx
```

Ubuntu 16.04

```
$ sudo systemctl enable nginx
```

Ubuntu 18.04

```
$ sudo systemctl enable nginx
```

After you update your web server configuration, go to [Step 4: Enable HTTPS traffic and verify the certificate \(p. 430\)](#).

Configure Apache web server

Use this section to configure Apache on supported platforms.

To update the web server configuration for Apache

1. Connect to your Amazon EC2 client instance.
2. Define default locations for certificates and private keys for your platform.

Amazon Linux

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

Amazon Linux 2

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

CentOS 7

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

CentOS 8

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

Red Hat 7

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

Red Hat 8

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/pki/tls/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/pki/tls/private/localhost.key</code>

Ubuntu 16.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/ssl/certs/localhost.crt</code>
<code>SSLCertificateKeyFile</code>	<code>/etc/ssl/private/localhost.key</code>

Ubuntu 18.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

<code>SSLCertificateFile</code>	<code>/etc/ssl/certs/localhost.crt</code>
---------------------------------	---

```
SSLCertificateKeyFile /etc/ssl/private/localhost.key
```

3. Copy your web server certificate to the required location for your platform.

Amazon Linux

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Amazon Linux 2

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

CentOS 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

CentOS 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Red Hat 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Red Hat 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 16.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 18.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

4. Copy your fake PEM private key to the required location for your platform.

Amazon Linux

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

Amazon Linux 2

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

CentOS 7

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

CentOS 8

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

Red Hat 7

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

Red Hat 8

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

Ubuntu 16.04 LTS

```
$ sudo cp <web_server_fake_PEM.key> /etc/ssl/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

Ubuntu 18.04 LTS

```
$ sudo cp <web_server_fake_PEM.key> /etc/ssl/private/localhost.key
```

Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key.

5. Change ownership of these files if required by your platform.

Amazon Linux

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

Amazon Linux 2

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

CentOS 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

CentOS 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

Red Hat 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

Red Hat 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

Provides read permission to the user named *apache*.

Ubuntu 16.04 LTS

No action required.

Ubuntu 18.04 LTS

No action required.

6. Configure Apache directives for your platform.

Amazon Linux

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Amazon Linux 2

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

CentOS 7

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

CentOS 8

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

Save the file.

Red Hat 7

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Red Hat 8

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

Save the file.

Ubuntu 16.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 18.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-
```

```
RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-  
AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-  
AES256-SHA:ECDHE-ECDSA-AES128-SHA  
SSLProtocol TLSv1.2 TLSv1.3
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl  
$ sudo a2ensite default-ssl
```

7. Configure an environment-values file for your platform.

Amazon Linux

No action required. Environment values go in /etc/sysconfig/httpd

Amazon Linux 2

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the [Service] section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 7

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the [Service] section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 8

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the [Service] section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 7

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the [Service] section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 8

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the [Service] section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Ubuntu 16.04 LTS

No action required. Environment values go in /etc/sysconfig/httpd

Ubuntu 18.04 LTS

No action required. Environment values go in /etc/sysconfig/httpd

8. In the file that stores environment variables for your platform, set an environment variable that contains the credentials of the cryptographic user (CU):

Amazon Linux

Use a text editor to edit the /etc/sysconfig/httpd.

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Amazon Linux 2

Use a text editor to edit the /etc/sysconfig/httpd.

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

CentOS 7

Use a text editor to edit the /etc/sysconfig/httpd.

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

CentOS 8

Use a text editor to edit the /etc/sysconfig/httpd.

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Red Hat 7

Use a text editor to edit the /etc/sysconfig/httpd.

- If using Client SDK 3

```
n3fips_password=<CU user name>:<password>
```

- If using Client SDK 5

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Red Hat 8

Use a text editor to edit the /etc/sysconfig/httpd.

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Note

Client SDK 5 introduces the CLOUDHSM_PIN environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the n3fips_password environment variable. Client SDK 5 supports both environment variables, but we recommend using CLOUDHSM_PIN.

Ubuntu 16.04 LTS

Use a text editor to edit the /etc/apache2/envvars.

```
export n3fips_password=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Ubuntu 18.04 LTS

Use a text editor to edit the /etc/apache2/envvars.

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

Replace `<CU user name>` and `<password>` with the CU credentials.

Note

Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the `n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

9. Start the Apache web server.

Amazon Linux

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Amazon Linux 2

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Ubuntu 16.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 18.04 LTS

```
$ sudo service apache2 start
```

10. (Optional) Configure your platform to start Apache at start-up.

Amazon Linux

```
$ sudo chkconfig httpd on
```

Amazon Linux 2

```
$ sudo chkconfig httpd on
```

CentOS 7

```
$ sudo chkconfig httpd on
```

CentOS 8

```
$ systemctl enable httpd
```

Red Hat 7

```
$ sudo chkconfig httpd on
```

Red Hat 8

```
$ systemctl enable httpd
```

Ubuntu 16.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 18.04 LTS

```
$ sudo systemctl enable apache2
```

After you update your web server configuration, go to [Step 4: Enable HTTPS traffic and verify the certificate \(p. 430\)](#).

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 \(p. 431\)](#)
- [Verify that HTTPS uses the certificate that you configured \(p. 431\)](#)

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.

```
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 [Step 5 \(Optional\): Add a load balancer with Elastic Load Balancing \(p. 432\)](#).

Step 5 (Optional): 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 \(p. 432\)](#)
- [Create the second web server \(p. 433\)](#)
- [Create the load balancer \(p. 435\)](#)

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.

subnet-1f358d78 | CloudHSM Public subnet

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-68ee440c

- 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](#)

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**.

▼ AMI Details



amzn-ami-hvm-2017.09.1.20171120-x86_64-gp2 - ami-a51f27c5

Amazon Linux AMI 2017.09.1.20171120 x86_64 HVM GP2

Root Device Type: ebs Virtualization type: hvm

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 \(p. 12\)](#). 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 \(p. 431\)](#).
 - 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.

```
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.

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 \(p. 437\)](#)
- [Step 1: Set up the prerequisites \(p. 437\)](#)

- [Step 2: Create a certificate signing request \(CSR\) and certificate \(p. 438\)](#)
- [Step 3: Configure the web server \(p. 440\)](#)
- [Step 4: Enable HTTPS traffic and verify the certificate \(p. 442\)](#)
- [\(Optional\) Step 5: Add a load balancer with Elastic Load Balancing \(p. 443\)](#)

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 \(CNG\) \(p. 383\)](#) 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 \(p. 437\)](#).

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 \(p. 60\)](#) (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 \(p. 10\)](#). 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. [Create a crypto user \(CU\) \(p. 64\)](#). 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 \(p. 385\)](#), 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 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 \(p. 438\)](#).

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 \(CNG\) \(p. 383\)](#) 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 \(p. 438\)](#)
- [Get a signed certificate and import it \(p. 439\)](#)

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 daemon.
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.

```
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.

```
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.

```
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
Signature ok
subject=/ST=IIS-HSM/L=IIS-HSM/OU=IIS-HSM/O=IIS-HSM/CN=IIS-HSM/C=IIS-HSM
Getting CA Private Key
Enter pass phrase for SelfSignedCA.key:
```

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 \(p. 440\)](#).

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 \(p. 438\)](#). 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 \(p. 438\)](#), 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 daemon.
3. Copy your web server's signed certificate—the one that you created at the end of [this tutorial's previous step \(p. 438\)](#)—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 \(p. 438\)](#).

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 \(p. 442\)](#).

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 \(p. 442\)](#)
- [Verify that HTTPS uses the certificate that you configured \(p. 442\)](#)

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.

```
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 \(p. 443\)](#).

(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 \(p. 443\)](#)
- [Create the second web server \(p. 444\)](#)
- [Create the load balancer \(p. 446\)](#)

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.

subnet-1f358d78 | CloudHSM Public subnet

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-68ee440c

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](#)

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**.

▼ AMI Details

 amzn-ami-hvm-2017.09.1.20171120-x86_64-gp2 - ami-a51f27c5
Amazon Linux AMI 2017.09.1.20171120 x86_64 HVM GP2
Root Device Type: ebs Virtualization type: hvm

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 \(p. 12\)](#). 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 \(p. 442\)](#).
 - 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.

```
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 \(p. 448\)](#)
- [Windows Server CA step 2: Create a Windows Server CA with AWS CloudHSM \(p. 449\)](#)
- [Windows Server CA step 3: Sign a certificate signing request \(CSR\) with your Windows Server CA with AWS CloudHSM \(p. 450\)](#)

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 \(p. 10\)](#). 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. [Create a crypto user \(CU\) \(p. 64\)](#). 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 \(p. 385\)](#), 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_clouhdsm_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 \(p. 449\)](#).

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:
 - a. Select **Active Directory Certificate Services**.
 - 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 \(p. 450\)](#).

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.

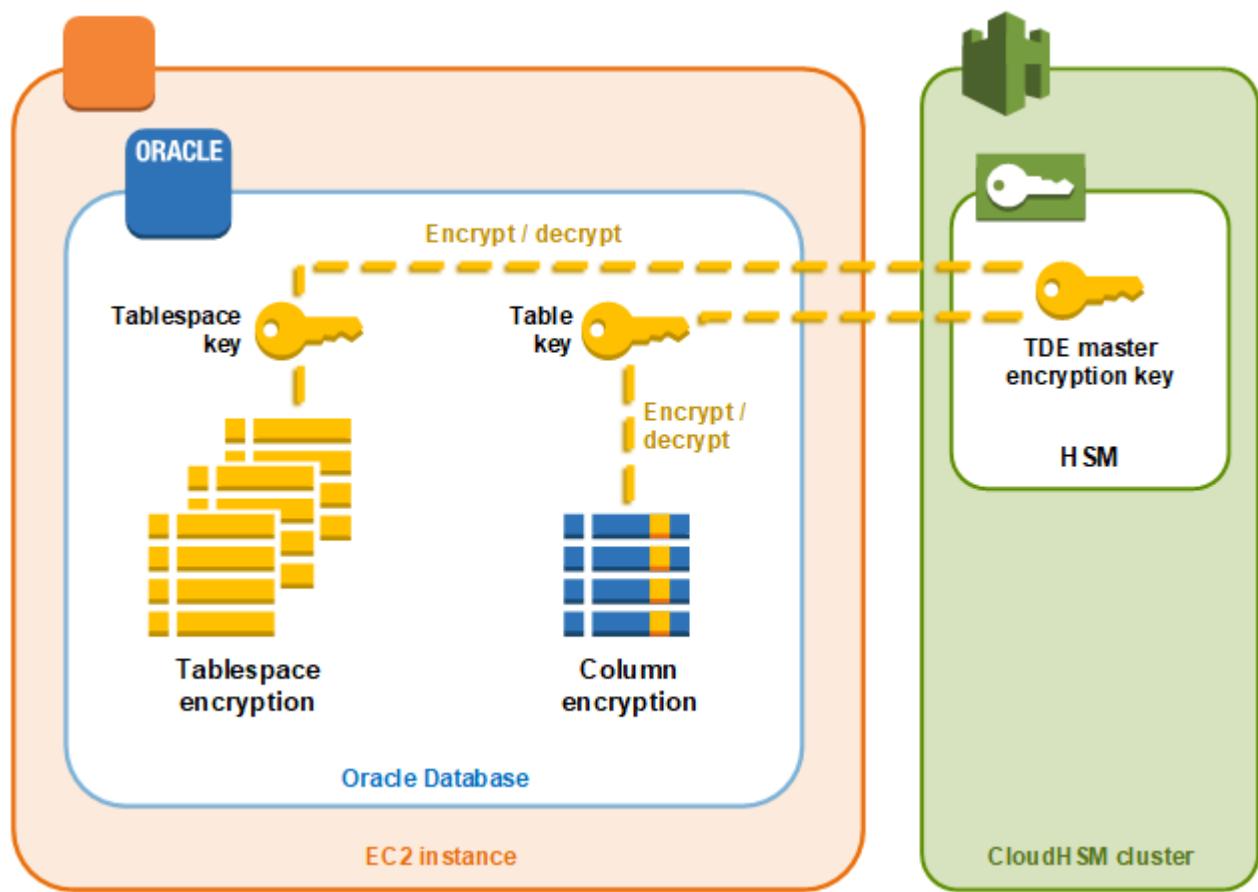
To sign a CSR with your Windows Server CA

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 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.



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 \(p. 324\)](#) 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.
- You can't use Client SDK 5 for Oracle TDE integration. For more information, see [???](#) (p. 258).

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 \(p. 452\)](#) to prepare your environment.
2. Follow the steps in [Configure the database \(p. 453\)](#) 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 \(p. 10\)](#). 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 \(p. 324\)](#).
 - 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. Use the `cloudhsm_mgmt_util` command line tool to create a cryptographic user (CU) on your cluster. For more information about creating a CU, see [How to Manage HSM Users with CMU \(p. 64\)](#) and [Managing HSM users \(p. 59\)](#).

After you complete these steps, you can [Configure the database \(p. 453\)](#).

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 \(p. 453\)](#) 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 \(p. 454\)](#) on the HSMs in your cluster.

Topics

- [Update the Oracle database configuration \(p. 453\)](#)
- [Generate the Oracle TDE master encryption key \(p. 454\)](#)

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. Run the following command to copy the AWS CloudHSM software library for PKCS #11 file to the directory that you created in the previous step.

```
sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11.so /opt/oracle/extapi/64/hsm/
```

Note

The `/opt/oracle/extapi/64/hsm` directory must contain only one library file. Remove any other files that exist in that directory.

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 following command to open Oracle SQL*Plus and set the `CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE` environment variable. When prompted, type the system password that you set when you installed Oracle Database.

```
CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE=true sqlplus / as sysdba
```

Note

You must set the `CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE` environment variable each time you generate a master key. This variable is only needed for master key generation. For more information, see "Issue: Oracle sets the PKCS #11 attribute `CKA_MODIFIABLE` during master key generation, but the HSM does not support it" in [Known Issues for Integrating Third-Party Applications \(p. 523\)](#).

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> alter system set encryption key identified by "<CU user name>:<password>";
```

- For Oracle Database version 12, run the following SQL statement.

```
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> 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> alter system set encryption wallet open identified by "<CU user name>:<password>";
```

To manually close the wallet, run the following command.

```
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> administer key management set keystore open identified by "<CU user name>:<password>";
```

To manually close the wallet, run the following command.

```
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 \(p. 456\)](#)
- [Microsoft SignTool with AWS CloudHSM step 2: Create a signing certificate \(p. 457\)](#)
- [Microsoft SignTool with AWS CloudHSM step 3: Sign a file \(p. 458\)](#)

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 \(p. 10\)](#) 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 \(p. 448\)](#). 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 \(p. 457\)](#).

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](#).

```
[Version]
Signature= $Windows NT$
[NewRequest]
Subject = "C=<Country>,CN=<www.website.com>,\n
          O=<Organization>,OU=<Organizational-Unit>,\n
          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 \(p. 458\)](#).

Microsoft SignTool with AWS CloudHSM step 3: Sign a file

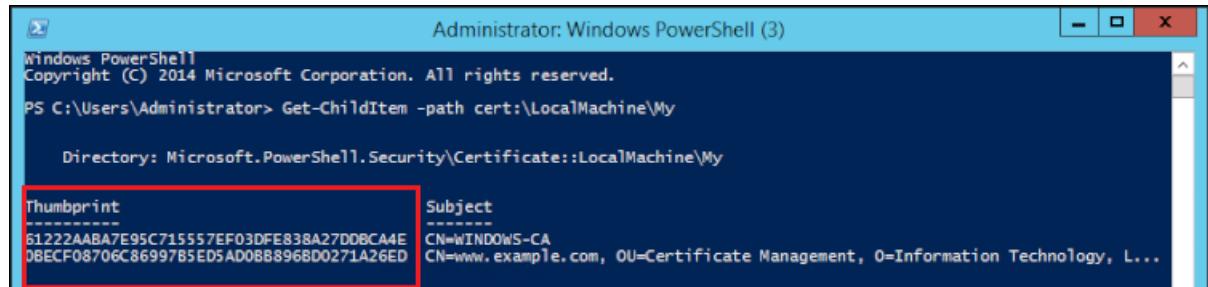
You are now ready to use SignTool and your imported certificate to sign your example file. In order to do so, you need to know the certificate's SHA-1 hash, or *thumbprint*. The thumbprint is used to ensure that SignTool only uses certificates that are verified by AWS CloudHSM. In this example, we use 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.



```
Administrator: Windows PowerShell (3)
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

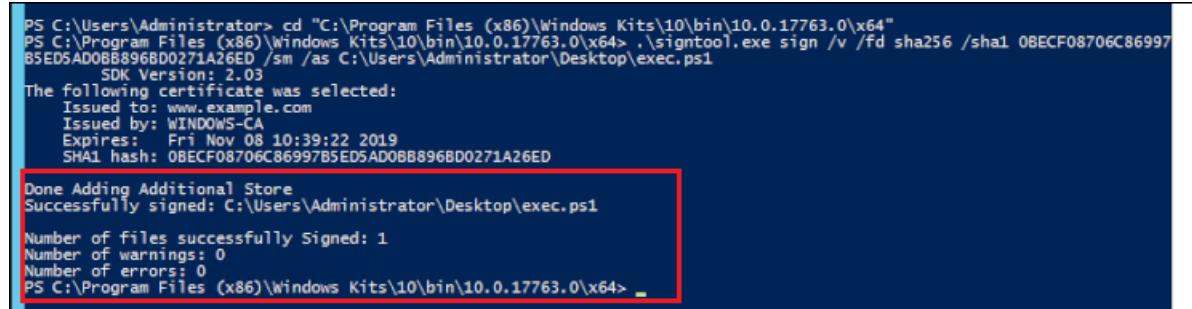
PS C:\Users\Administrator> Get-ChildItem -path cert:\LocalMachine\My

Directory: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint          Subject
-----            -----
6122AA8A7E95C715557EF03DFE838A27DD8CA4E CN=WINDWS-CA
D8ECF08706C86997B5ED5AD0B8896BD0271A26ED CN=www.example.com, OU=Certificate Management, O=Information Technology, L...
```

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.
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
```



```
PS C:\Users\Administrator> cd "C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64"
PS C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64> .\signtool.exe sign /v /fd sha256 /sha1 0BECF08706C86997B5ED5AD0B8896BD0271A26ED
SDK Version: 2.03
The following certificate was selected:
Issued to: www.example.com
Issued by: WINDWS-CA
Expires: Fri Nov 08 10:39:22 2019
SHA1 hash: 0BECF08706C86997B5ED5AD0B8896BD0271A26ED

Done Adding Additional Store
Successfully signed: C:\Users\Administrator\Desktop\exec.ps1

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
PS C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64>
```

4. (Optional) To verify the signature on the file, use the following command:

```
signtool.exe verify /v /pa C:\Users\Administrator\Desktop\<test>.ps1
```

Java Keytool and Jarsigner

AWS CloudHSM offers integration with the Java Keytool and Jarsigner utilities through Client SDK 3 and Client SDK 5. The steps to use these tools will vary depending on the version of the client SDK in which you currently have downloaded:

- [Using Client SDK 3 to integrate with Java Keytool and Jarsigner \(p. 459\)](#)
- [Using Client SDK 5 to integrate with Java Keytool and Jarsigner \(p. 469\)](#)

Using Client SDK 3 to integrate with Java Keytool and Jarsigner

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 \(p. 459\)](#)
- [Using AWS CloudHSM key store with keytool \(p. 460\)](#)
- [Using AWS CloudHSM key store with jarsigner \(p. 463\)](#)
- [Known issues \(p. 464\)](#)
- [Registering pre-existing keys with AWS CloudHSM key store \(p. 464\)](#)

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 \(p. 352\)](#).

Step 2: Add HSM login credentials to environment variables

Set up environment variables to contain your HSM login credentials.

```
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 \(p. 367\)](#).

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 KeyStore \(p. 368\)](#). 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 \(p. 357\)](#) 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 \(p. 464\)](#). 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 \(p. 461\)](#). 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 -genseckeypair -alias my_secret -keyalg aes \
    -keysize 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 \(p. 360\)](#) 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 \(p. 160\)](#).

Generate a CSR using keytool

You receive the greatest flexibility in generating a certificate signing request (CSR) if you use the [OpenSSL Dynamic Engine \(p. 347\)](#). 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 \(p. 462\)](#).

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 \
-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, 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.

```
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 \(p. 464\)](#).

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.

```
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 \(p. 459\)](#).
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 \(p. 464\)](#) 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 \(p. 460\)](#). 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 \(p. 463\)](#) 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 \(p. 462\)](#).

Sign a JAR file using AWS CloudHSM and jarsigner

Use the following command to sign a JAR file:

```
jarsigner -keystore my_keystore.store \
    -signedjar signthisclass_signed.jar \
    -sigalg sha512withrsa \
    -storetype CloudHSM \
    -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
    -J-Djava.library.path=/opt/cloudhsm/lib \
    signthisclass.jar my_key_pair
```

Use the following command to verify a signed JAR:

```
jarsigner -verify \
    -keystore my_keystore.store \
    -sigalg sha512withrsa \
    -storetype CloudHSM \
    -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
    -J-Djava.library.path=/opt/cloudhsm/lib \
    signthisclass_signed.jar 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 \(p. 459\)](#).
- 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 \(p. 98\)](#). 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 stored 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.

```
//
// Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this
// software and associated documentation files (the "Software"), to deal in the Software
```

```
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
// OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
// SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
//

package com.amazonaws.cloudhsm.examples;

import com.cavium.key.CaviumKey;
import com.cavium.key.parameter.CaviumAESKeyGenParameterSpec;
import com.cavium.key.parameter.CaviumRSAKeyGenParameterSpec;
import com.cavium.asn1.Encoder;
import com.cavium.cfm2.Util;

import javax.crypto.KeyGenerator;

import java.io.ByteArrayInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;

import java.math.BigInteger;

import java.security.*;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.KeyStore.PasswordProtection;
import java.security.KeyStore.PrivateKeyEntry;
import java.security.KeyStore.Entry;

import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;

//
// KeyStoreExampleRunner demonstrates how to load a keystore, and associate a certificate
// with a
// key in that keystore.
//
// This example relies on implicit credentials, so you must setup your environment
// correctly.
//
// https://docs.aws.amazon.com/cloudhsm/latest/userguide/java-library-install.html#java-
library-credentials
//

public class KeyStoreExampleRunner {

    private static byte[] COMMON_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04, (byte)
0x03 };
    private static byte[] COUNTRY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04, (byte)
0x06 };
    private static byte[] LOCALITY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
(byte) 0x07 };
    private static byte[] STATE_OR_PROVINCE_NAME_OID = new byte[] { (byte) 0x55, (byte)
0x04, (byte) 0x08 };
```

```
private static byte[] ORGANIZATION_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
(byte) 0x0A };
private static byte[] ORGANIZATION_UNIT_OID = new byte[] { (byte) 0x55, (byte) 0x04,
(byte) 0x0B };

private static String helpString = "KeyStoreExampleRunner%n" +
"This sample demonstrates how to load and store keys using a keystore.%n%n" +
"Options%n" +
"\t--help\t\tDisplay this message.%n" +
"\t--store <filename>\t\tPath of the keystore.%n" +
"\t--password <password>\t\tPassword for the keystore (not your CU password).
%n" +
"\t--label <label>\t\tLabel to store the key and certificate under.%n" +
"\t--list\t\tList all the keys in the keystore.%n%n";

public static void main(String[] args) throws Exception {
    Security.addProvider(new com.cavium.provider.CaviumProvider());
    KeyStore keyStore = KeyStore.getInstance("CloudHSM");

    String keystoreFile = null;
    String password = null;
    String label = null;
    boolean list = false;
    for (int i = 0; i < args.length; i++) {
        String arg = args[i];
        switch (args[i]) {
            case "--store":
                keystoreFile = args[++i];
                break;
            case "--password":
                password = args[++i];
                break;
            case "--label":
                label = args[++i];
                break;
            case "--list":
                list = true;
                break;
            case "--help":
                help();
                return;
        }
    }
    if (null == keystoreFile || null == password) {
        help();
        return;
    }

    if (list) {
        listKeys(keystoreFile, password);
        return;
    }

    if (null == label) {
        label = "Keystore Example Keypair";
    }

    //
    // This call to keyStore.load() will open the pkcs12 keystore with the supplied
    // password and connect to the HSM. The CU credentials must be specified using
    // standard CloudHSM login methods.
    //
    try {
        FileInputStream instream = new FileInputStream(keystoreFile);
        keyStore.load(instream, password.toCharArray());
    }
}
```

```

        } catch (FileNotFoundException ex) {
            System.err.println("Keystore not found, loading an empty store");
            keyStore.load(null, null);
        }

        PasswordProtection passwd = new PasswordProtection(password.toCharArray());
        System.out.println("Searching for example key and certificate...");

        PrivateKeyEntry keyEntry = (PrivateKeyEntry) keyStore.getEntry(label, passwd);
        if (null == keyEntry) {
            //
            // No entry was found, so we need to create a key pair and associate a
            certificate.
            //
            // The private key will get the label passed on the command line. The keystore
            alias
            // needs to be the same as the private key label. The public key will have
            ":public"
            // appended to it. The alias used in the keystore will be associated with 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
    privateKeyLabel)
        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, privateKeyLabel, isExtractable, isPersistent);

keyPairGen.initialize(spec);

return keyPairGen.generateKeyPair();
}

// Generate a certificate signed by a given keypair.
//
private static Certificate generateCert(KeyPair kp) throws CertificateException {
    CertificateFactory cf = CertificateFactory.getInstance("X509");
    PublicKey publicKey = kp.getPublic();
    PrivateKey privateKey = kp.getPrivate();
    byte[] version = Encoder.encodeConstructed((byte) 0,
Encoder.encodePositiveBigInteger(new BigInteger("2"))); // version 1
    byte[] serialNo = Encoder.encodePositiveBigInteger(new BigInteger(1,
Util.computeKCV(publicKey.getEncoded())));

    // Use the SHA512 OID and algorithm.
    byte[] signatureOid = new byte[] {
        (byte) 0x2A, (byte) 0x86, (byte) 0x48, (byte) 0x86, (byte) 0xF7, (byte) 0x0D,
(byte) 0x01, (byte) 0x01, (byte) 0x0D };
    String sigAlgoName = "SHA512WithRSA";

    byte[] signatureId = Encoder.encodeSequence(
        Encoder.encodeOid(signatureOid),
        Encoder.encodeNull());

    byte[] issuer = Encoder.encodeSequence(
        encodeName(COUNTRY_NAME_OID, "<Country>"),
        encodeName(STATE_OR_PROVINCE_NAME_OID, "<State>"),
        encodeName(LOCALITY_NAME_OID, "<City>"),
        encodeName(ORGANIZATION_NAME_OID, "<Organization>"),
        encodeName(ORGANIZATION_UNIT_OID, "<Unit>"),
        encodeName(COMMON_NAME_OID, "<CN>")
    );

    Calendar c = Calendar.getInstance();
    c.add(Calendar.DAY_OF_YEAR, -1);
    Date notBefore = c.getTime();
    c.add(Calendar.YEAR, 1);
    Date notAfter = c.getTime();
    byte[] validity = Encoder.encodeSequence(
        Encoder.encodeUTCTime(notBefore),
        Encoder.encodeUTCTime(notAfter)
    );
    byte[] key = publicKey.getEncoded();

    byte[] certificate = Encoder.encodeSequence(
        version,
        serialNo,
        signatureId,
        issuer,
        validity,
        issuer,
        key);

    Signature sig;
    byte[] signature = null;
    try {
        sig = Signature.getInstance(sigAlgoName, "Cavium");
        sig.initSign(privateKey);
```

```
        sig.update(certificate);
        signature = Encoder.encodeBitstring(sig.sign());

    } catch (Exception e) {
        System.err.println(e.getMessage());
        return null;
    }

    byte [] x509 = Encoder.encodeSequence(
        certificate,
        signatureId,
        signature
    );
    return cf.generateCertificate(new ByteArrayInputStream(x509));
}

// Simple OID encoder.
// Encode a value with OID in ASN.1 format
//
private static byte[] encodeName(byte[] nameOid, String value) {
    byte[] name = null;
    name = Encoder.encodeSet(
        Encoder.encodeSequence(
            Encoder.encodeOid(nameOid),
            Encoder.encodePrintableString(value)
        )
    );
    return name;
}

// List all the keys in the keystore.
//
private static void listKeys(String keystoreFile, String password) throws Exception {
    KeyStore keyStore = KeyStore.getInstance("CloudHSM");

    try {
        FileInputStream instream = new FileInputStream(keystoreFile);
        keyStore.load(instream, password.toCharArray());
    } catch (FileNotFoundException ex) {
        System.err.println("Keystore not found, loading an empty store");
        keyStore.load(null, null);
    }

    for(Enumeration<String> entry = keyStore.aliases(); entry.hasMoreElements();) {
        System.out.println(entry.nextElement());
    }
}
}
```

Using Client SDK 5 to integrate with Java Keytool and Jarsigner

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 \(p. 470\)](#)
- [Using AWS CloudHSM key store with keytool \(p. 471\)](#)
- [Using AWS CloudHSM key store with Jarsigner \(p. 463\)](#)
- [Known issues \(p. 477\)](#)

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 \(p. 370\)](#).

Step 2: Add HSM login credentials to environment variables

Set up environment variables to contain your HSM login credentials.

Linux

```
$ export HSM_USER=<HSM user name>
```

```
$ export HSM_PASSWORD=<HSM password>
```

Windows

```
PS C:\> $Env:HSM_USER=<HSM user name>
```

```
PS C:\> $Env:HSM_PASSWORD=<HSM password>
```

Note

The AWS 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 \(p. 380\)](#).

Step 3: Registering the JCE provider

To register the JCE provider in the Java CloudProvider configuration, follow these steps:

1. Open the `java.security` configuration file in your Java installation for editing.
2. In the `java.security` configuration file, add `com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider` 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:

```
security.provider.10=com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider
```

Note

Adding the AWS CloudHSM provider as a higher priority may negatively impact your system's performance because the AWS CloudHSM provider will be prioritized for operations that may be safely offloaded to software. As a best practice, **always** specify the provider you wish to use for an operation, whether it is the AWS CloudHSM or a software-based provider.

Note

Specifying `-providerName`, `-providerclass`, and `-providerpath` command line options when generating keys using keytool with the AWS CloudHSM key store may cause errors.

Using AWS CloudHSM key store with keytool

[Keytool](#) is a popular command line utility for common key and certificate tasks. 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:

Linux

```
-storetype CLOUDHSM -J-classpath< '-J/opt/cloudhsm/java/*'>
```

Windows

```
-storetype CLOUDHSM -J-classpath<'-J"C:\Program Files\Amazon\CloudHSM\java\*"'>
```

If you want to create a new key store file using AWS CloudHSM key store, see [Using AWS CloudHSM KeyStore \(p. 368\)](#). 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 RSA, AES, and DESede type of key supported by AWS CloudHSM's JCE SDK.

Important

A key generated through keytool is generated in software, and then imported into AWS CloudHSM as an extractable, persistent key.

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 AWS 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 [the section called "Generate a CSR using keytool" \(p. 472\)](#). 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.

Linux

```
$ keytool -genseckeypair -alias my_secret -keyalg aes \
-keysize 256 -keystore my_keystore.store \
-storetype CloudHSM -J-classpath '-J/opt/cloudhsm/java/*' \
```

Windows

```
PS C:\> keytool -genseckeypair -alias my_secret -keyalg aes \
-keysize 256 -keystore my_keystore.store \
-storetype CloudHSM -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"' \
```

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.

Linux

```
$ keytool -genkeypair -alias my_rsa_key_pair \
-keyalg rsa -keysize 2048 \
-signalg sha512withrsa \
-keystore my_keystore.store \
-storetype CLOUDHSM \
-J-classpath '-J/opt/cloudhsm/java/*' \
```

Windows

```
PS C:\> keytool -genkeypair -alias my_rsa_key_pair \
-keyalg rsa -keysize 2048 \
-signalg sha512withrsa \
-keystore my_keystore.store \
-storetype CLOUDHSM \
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"' \
```

You can find a list of [supported signature algorithms \(p. 376\)](#) in the Java library.

Delete a key using keytool

The AWS CloudHSM key store doesn't support deleting keys. You can delete keys using the `destroy` method of the [Destroyable interface](#).

```
((Destroyable) key).destroy();
```

Generate a CSR using keytool

You receive the greatest flexibility in generating a certificate signing request (CSR) if you use the [OpenSSL Dynamic Engine \(p. 347\)](#). The following command uses keytool to generate a CSR for a key pair with the alias, `my-key-pair`.

Linux

```
$ keytool -certreq -alias my_key_pair \
-file my_csr.csr \
-keystore my_keystore.store \
```

```
-storetype CLOUDHSM \
-J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -certreq -alias my_key_pair ^
-file my_csr.csr ^
-keystore my_keystore.store ^
-storetype CLOUDHSM ^
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

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 [the section called "Using keytool to import intermediate and root certificates into AWS CloudHSM key store" \(p. 473\)](#).

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.

Linux

```
$ keytool -import -trustcacerts -alias rootCAcert \
-file rootCAcert.cert -keystore my_keystore.store \
-storetype CLOUDHSM \
-J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -import -trustcacerts -alias rootCAcert ^
-file rootCAcert.cert -keystore my_keystore.store ^
-storetype CLOUDHSM ^
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

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.

Linux

```
$ keytool -delete -alias mydomain -keystore \
-keystore my_keystore.store \
-storetype CLOUDHSM \
-J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -delete -alias mydomain -keystore ^
```

```
-keystore my_keystore.store `  
-storetype CLOUDHSM `  
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

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.

Linux

```
$ keytool -importcert -noprompt -alias my_key_pair `  
-file my_certificate.crt `  
-keystore my_keystore.store `  
-storetype CLOUDHSM `  
-J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -importcert -noprompt -alias my_key_pair `  
-file my_certificate.crt `  
-keystore my_keystore.store `  
-storetype CLOUDHSM `  
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

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.

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.

Linux

```
$ keytool -exportcert -alias my_key_pair `  
-file my_exported_certificate.crt `  
-keystore my_keystore.store `  
-storetype CLOUDHSM `  
-J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -exportcert -alias my_key_pair `  
-file my_exported_certificate.crt `  
-keystore my_keystore.store `  
-storetype CLOUDHSM `  
-J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

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 \(p. 470\)](#).
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. If you want to use keytool to set up the keys and certificates, see [the section called "Create new keys with keytool" \(p. 471\)](#). 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 [the section called "Sign a JAR file using AWS CloudHSM and Jarsigner" \(p. 475\)](#) 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 [the section called "Using keytool to import intermediate and root certificates into AWS CloudHSM key store" \(p. 473\)](#).

Sign a JAR file using AWS CloudHSM and Jarsigner

Use the following command to sign a JAR file:

Amazon Linux

For OpenJDK8

```
jarsigner -keystore my_keystore.store \
-signedjar signthisclass_signed.jar \
-sigalg sha512withrsa \
-storetype CloudHSM \
-J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
-J-Djava.library.path=/opt/cloudhsm/lib \
signthisclass.jar my_key_pair
```

For OpenJDK11

```
jarsigner -keystore my_keystore.store \
-signedjar signthisclass_signed.jar \
-sigalg sha512withrsa \
-storetype CloudHSM \
-J-classpath '-J/opt/cloudhsm/java/*' \
-J-Djava.library.path=/opt/cloudhsm/lib \
signthisclass.jar my_key_pair
```

Windows

For OpenJDK8

```
jarsigner -keystore my_keystore.store `  
-signedjar signthisclass_signed.jar `  
-sigalg sha512withrsa `  
-storetype CloudHSM `  
-J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java  
\jdk1.8.0_331\lib\tools.jar' `  
"-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `  
signthisclass.jar my_key_pair
```

For OpenJDK11

```
jarsigner -keystore my_keystore.store `  
-signedjar signthisclass_signed.jar `  
-sigalg sha512withrsa `  
-storetype CloudHSM `  
-J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*' `  
"-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `  
signthisclass.jar my_key_pair
```

Use the following command to verify a signed JAR:

Amazon Linux

For OpenJDK8

```
jarsigner -verify `  
-keystore my_keystore.store `  
-sigalg sha512withrsa `  
-storetype CloudHSM `  
-J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' `  
-J-Djava.library.path=/opt/cloudhsm/lib `  
signthisclass_signed.jar my_key_pair
```

For OpenJDK11

```
jarsigner -verify `  
-keystore my_keystore.store `  
-sigalg sha512withrsa `  
-storetype CloudHSM `  
-J-classpath '-J/opt/cloudhsm/java/*' `  
-J-Djava.library.path=/opt/cloudhsm/lib `  
signthisclass_signed.jar my_key_pair
```

Windows

For OpenJDK8

```
jarsigner -verify `  
-keystore my_keystore.store `  
-sigalg sha512withrsa `  
-storetype CloudHSM `  
-J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java  
\jdk1.8.0_331\lib\tools.jar' `  
"-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `  
signthisclass_signed.jar my_key_pair
```

For OpenJDK11

```
jarsigner -verify `  
-keystore my_keystore.store`  
-sigalg sha512withrsa`  
-storetype CloudHSM`  
-J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*'`  
"-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'"`  
signthisclass_signed.jar my_key_pair
```

Known issues

1. We do not support EC keys with Keytool and Jarsigner.

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.
- [Venafi Trust Protection Platform](#) provides comprehensive machine identity management for TLS, SSH, and code signing with AWS CloudHSM key generation and protection.

Monitoring AWS CloudHSM

In addition to the logging features built into the Client SDK, you can also use AWS CloudTrail, Amazon CloudWatch Logs, and Amazon CloudWatch to monitor AWS CloudHSM.

Client SDK logs

Use Client SDK logging to monitor diagnostic and troubleshooting information from the applications you create.

CloudTrail

Use CloudTrail to monitor all API calls in your AWS account, including the calls you make to create and delete clusters, hardware security modules (HSM), and resource tags.

CloudWatch Logs

Use CloudWatch Logs to monitor the logs from your HSM instances, which include events for create and delete HSM users, change user passwords, create and delete keys, and more.

CloudWatch

Use CloudWatch to monitor the health of your cluster in real time.

Topics

- [Working with client SDK logs \(p. 478\)](#)
- [Working with AWS CloudTrail and AWS CloudHSM \(p. 482\)](#)
- [Working with Amazon CloudWatch Logs and AWS CloudHSM \(p. 484\)](#)
- [Getting CloudWatch metrics for AWS CloudHSM \(p. 500\)](#)

Working with client SDK logs

You can retrieve logs generated by the Client SDK. AWS CloudHSM offers an implementation of logging with Client SDK 3 and Client SDK 5.

Topics

- [Client SDK 3 logging \(p. 478\)](#)
- [Client SDK 5 logging \(p. 480\)](#)

Client SDK 3 logging

Client SDK 3 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 client daemon.

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 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
```

CentOS 8

In CentOS 8, 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 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
```

RHEL 8

In Red Hat Enterprise Linux 8, 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 16.04, 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 18.04

In Ubuntu 18.04, 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.

Client SDK 5 logging

Client SDK 5 logs contain information for each component in a file named for the component. You can use the configure tool for Client SDK 5 to configure logging for each component. For more information about the configure tool, see [Client SDK 5 Configure Tool \(p. 245\)](#).

How to configure logging for Client SDK 5

You can configure the name of the log file, how much information Client SDK 5 components include in the logs, and how often the system rotates the logs.

PKCS #11 library

To configure the name of the logging file

- Use the `log-file` option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-file path/to/log
```

For example, use the following command to set the log file name to `cloudhsm-pkcs11.log`.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-file cloudhsm-pkcs11.log
```

If you do not specify a location for the file, the system writes logs to the default location.

- Linux:

```
/opt/cloudhsm/run
```

- Windows:

```
C:\ProgramData\Amazon\CloudHSM
```

To configure the logging level

- Use the `log-level` option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level warn
```

To configure log rotation

- Use the `log-rotation` option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-rotation daily
```

OpenSSL Dynamic Engine

To configure the name of the logging file

- Use the `log-file` option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-file path/to/log
```

For example, use the following command to set the log file name to `cloudhsm-dyn.log`.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-file cloudhsm-dyn.log
```

If you do not specify a location for the file, the system writes logs to `stderr`

To configure the logging level

- Use the `log-level` option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-level warn
```

To configure log rotation

- Use the `log-rotation` option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --log-rotation daily
```

JCE provider

To configure the name of the logging file

- Use the `log-file` option to change the name or location of the log file.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-file path/to/log
```

For example, use the following command to set the log file name to `cloudhsm-dyn.log`.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-file cloudhsm-dyn.log
```

If you do not specify a location for the file, the system writes logs to the default location:

- Linux:

```
/opt/cloudhsm/run
```

- Windows:

```
C:\ProgramData\Amazon\CloudHSM
```

To configure the logging level

- Use the `log-level` option to establish how much information Client SDK 5 should log.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-level <error | warn | info | debug | trace>
```

For example, use the following command to set the log level to receive error and warning messages in logs.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-level warn
```

To configure log rotation

- Use the `log-rotation` option to establish how often Client SDK 5 should rotate the logs.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-rotation <daily | hourly | never>
```

For example, use the following command to rotate the logs daily. Each day the system creates a new log and appends a time stamp to the file name.

```
$ sudo /opt/cloudhsm/bin/configure-jce --log-rotation daily
```

Working with AWS CloudTrail and AWS CloudHSM

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.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#). For a full list of AWS CloudHSM API operations, see [Actions](#) in the [AWS CloudHSM API Reference](#).

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.

```
{
```

```

    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAJZVM5NEGZSTCITAMM:ExampleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/AdminRole/ExampleSession",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIY22AX6VRYNBGJSA",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2017-07-11T03:48:44Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAJZVM5NEGZSTCITAMM",
                "arn": "arn:aws:iam::111122223333:role/AdminRole",
                "accountId": "111122223333",
                "userName": "AdminRole"
            }
        }
    },
    "eventTime": "2017-07-11T03:50:45Z",
    "eventSource": "clouhdsm.amazonaws.com",
    "eventName": "CreateHsm",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.179",
    "userAgent": "aws-internal/3",
    "requestParameters": {
        "availabilityZone": "us-west-2b",
        "clusterId": "cluster-fw7mh6mayb5"
    },
    "responseElements": {
        "hsm": {
            "eniId": "eni-65338b5a",
            "clusterId": "cluster-fw7mh6mayb5",
            "state": "CREATE_IN_PROGRESS",
            "eniIp": "10.0.2.7",
            "hsmId": "hsm-61z2hfmnzbx",
            "subnetId": "subnet-02c28c4b",
            "availabilityZone": "us-west-2b"
        }
    },
    "requestID": "1dae0370-65ec-11e7-a770-6578d63de907",
    "eventID": "b73a5617-8508-4c3d-900d-aa8ac9b31d08",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}

```

Working with Amazon CloudWatch Logs and AWS CloudHSM

When an HSM in your account receives a command from the AWS CloudHSM [command line tools \(p. 105\)](#) or [software libraries \(p. 258\)](#), it records its execution of the command in audit log form. The HSM audit logs include all client-initiated [management commands \(p. 498\)](#), 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 HSM audit logging works \(p. 485\)](#)
- [Viewing HSM audit logs in CloudWatch Logs \(p. 485\)](#)
- [Interpreting HSM audit logs \(p. 488\)](#)
- [HSM audit log reference \(p. 498\)](#)

How HSM 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 \(p. 507\)](#). 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 \(p. 507\)](#), as well as [Creating a Service-Linked Role](#) in the [IAM User Guide](#).

Viewing HSM 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-nwbbiqbj4jk`.

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.

The screenshot shows the AWS CloudWatch Log Groups page. At the top, there is a breadcrumb navigation: CloudWatch > Log Groups. Below the breadcrumb, there are two buttons: "Create Metric Filter" (highlighted in blue) and "Actions". A "Filter" input field is set to "Log Group Name Prefix". On the right, there are navigation icons for back, forward, and refresh, along with a gear icon for settings. The main table has columns: Log Groups, Expire Events After, Metric Filters, and Subscriptions. One row is visible, showing a log group named `/aws/cloudhsm/cluster-likphkxygsn`, which never expires, has 0 filters, and no subscriptions.

Log Groups	Expire Events After	Metric Filters	Subscriptions
<input type="radio"/> /aws/cloudhsm/cluster-likphkxygsn	Never Expire	0 filters	None

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.

The screenshot shows the AWS CloudWatch Log Streams page. At the top, there is a breadcrumb navigation: CloudWatch > Log Groups > Streams for `/aws/cloudhsm/cluster-likphkxygsn`. Below the breadcrumb, there are three buttons: "Search Log Group" (highlighted in blue), "Create Log Stream", and "Delete Log Stream". A "Filter" input field is set to "Log Stream Name Prefix". On the right, there are navigation icons for back, forward, and refresh, along with a gear icon for settings. The main table has columns: Log Streams and Last Event Time. Two log streams are listed: `hsm-aht4p3sgs3c` (last event time: 2017-12-28 06:12 UTC-8) and `hsm-xkvjp4wk5o3` (last event time: 2017-12-28 06:12 UTC-8).

Log Streams	Last Event Time
<input type="checkbox"/> hsm-aht4p3sgs3c	2017-12-28 06:12 UTC-8
<input type="checkbox"/> hsm-xkvjp4wk5o3	2017-12-28 06:12 UTC-8

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.

Filter events		
	Time (UTC +00:00)	Message
	2017-12-19	
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)		

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`.

CN_CREATE_USER	
Time (UTC +00:00)	Message
2017-12-20	No older events
▼ 00:04:53	Time: 12/20/17 00:04:53.635826, usecs:1513728293635826 Sequence No : 0x13a Reboot counter : 0xe8 Command Type(hex) : CN_MGMT_CMD (0x0) Opcode : CN_CREATE_USER (0x3) Session Handle : 0x1014006 Response : 0:HSM Return: SUCCESS Log type : MGMT_USER_DETAILS_LOG (2) User Name : testuser User Type : CN_CRYPTO_USER (1)

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 : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

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 [HSM audit log reference \(p. 498\)](#).

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 \(p. 490\)](#)
- [Login and logout events \(p. 491\)](#)
- [Example: Create and delete users \(p. 490\)](#)
- [Example: Create and delete a key pair \(p. 493\)](#)
- [Example: Generate and synchronize a key \(p. 494\)](#)
- [Example: Export a key \(p. 496\)](#)
- [Example: Import a key \(p. 497\)](#)
- [Example: Share and unshare a key \(p. 498\)](#)

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 \(p. 536\)](#).

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 \(AU\) \(p. 60\)](#), 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
```

```
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INIT_DONE (0x95)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

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 \(PRECO\) \(p. 59\)](#). 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 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.

When a user logs into an HSM, the other HSMs in the cluster also record a login event for the user. You can find the corresponding login events in the log streams of other HSMs in the cluster shortly after the initial login 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 : 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.

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:1516815699037255
Sequence No : 0x4
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 157:HSM Error: user isn't initialized or user with this name doesn't exist
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : ExampleUser
User Type : CN_CRYPTO_USER (1)
```

If a login attempt fails because the password is invalid, the HSM records a CN_LOGIN event with the user name and type provided in the login command. The response displays the error message with the RET_USER_LOGIN_FAILURE error code.

```
Time: 01/24/18 17:44:25.013218, usecs:1516815865013218
Sequence No : 0x5
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 163:HSM Error: RET_USER_LOGIN_FAILURE
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

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 : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_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)
```

```
Opcode : CN_CREATE_USER (0x3)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : bob
User Type : CN_CRYPTO_USER (1)
```

Then, the CO creates another crypto officer, alice. The sequence number indicates that this action followed the previous one with no intervening actions.

```
Time: 01/16/18 01:49:55.993404, usecs:1516067395993404
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_CO (0x4)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

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.

```
Time: 01/23/18 19:58:23.451420, usecs:1516737503451420
Sequence No : 0xb
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_DELETE_USER (0xa1)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

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.

```
Time: 12/13/17 23:09:04.648952, usecs:1513206544648952
Sequence No: 0x28
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_LOGIN (0xd)
Session Handle: 0x2014005
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

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.

```
Time: 12/13/17 23:09:04.761594, usecs:1513206544761594
Sequence No: 0x29
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_GENERATE_KEY_PAIR (0x19)
```

```
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
Public Key Handle: 131078
```

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 (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131078
Public Key Handle: 0
```

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 (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
Public Key Handle: 0
```

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 (0x0)
Opcode: CN_LOGOUT (0xe)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

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 \(p. 189\)](#) 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.

```
Time: 01/24/18 00:39:30.408950, usecs:1516754370408950
Sequence No : 0xe
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

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 \(p. 152\)](#).

```
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 \(p. 166\)](#) 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 \(p. 209\)](#) 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
Sequence No : 0xb
```

```
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0xc008004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

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 \(p. 145\)](#) 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:1549654539480168
Sequence No : 0x3f
Reboot counter : 0x38
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_SHARE_OBJECT (0x12)
Session Handle : 0x3014007
Response : 0:HSM Return: SUCCESS
Log type : UNKNOWN_LOG_TYPE (5)
```

HSM 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.

Operation Code (Opcode)	Description
User Login: These events include the user name and user type.	
CN_LOGIN (0xd)	User login (p. 136) (excludes appliance user [AU]).
CN_LOGOUT (0xe)	User logout (p. 136) (excludes appliance user [AU]).
CN_APP_FINALIZE	App finalize (logged only when user did not explicitly log out)
CN_CLOSE_SESSION	Close session (logged only when user did not explicitly log out)
User Management: These events include the user name and user type.	
CN_CREATE_USER (0x3)	Create a crypto user (CU) (p. 115)
CN_CREATE_CO	Create a crypto officer (CO) (p. 115)
CN_CREATE_APPLIANCE_USER	Create an appliance user (AU) (p. 115)
CN_DELETE_USER	Delete a user (p. 118)
CN_CHANGE_PSWD	Change a user password (p. 111)

Operation Code (Opcode)	Description
CN_SET_M_VALUE	Set quorum authentication (M of N) for a user action.
CN_APPROVE_TOKEN	Approve a quorum authentication token for a user action.
Key Management: These events include the key handle.	
CN_GENERATE_KEY	Generate a symmetric key (p. 189)
CN_GENERATE_KEY_PAIR (0x19)	Generate a key pair (DSA (p. 176) , ECC (p. 180) , or RSA (p. 184))
CN_CREATE_OBJECT	Import a public key (without wrapping)
CN_MODIFY_OBJECT	Set a key attribute in key_mgmt_util (p. 221) or cloudhsm_mgmt_util (p. 142) .
CN_DESTROY_OBJECT (0x11)	Delete a key (p. 160)
CN_TOMBSTONE_OBJECT	Mark the key for deletion, but do not remove it
CN_SHARE_OBJECT	Share or unshare a key (p. 145)
CN_WRAP_KEY	Export an encrypted copy of a key (wrapKey (p. 232))
CN_UNWRAP_KEY	Import an encrypted copy of a key (unwrapKey (p. 226))
CN_NIST_AES_WRAP	Encrypt or decrypt a file (aesWrapUnwrap (p. 157))
CN_INSERT_MASKED_OBJECT_USER	Receive a key (as a masked object) from another HSM in the cluster; this event is recorded when a client action synchronizes the key
CN_EXTRACT_MASKED_OBJECT_USER	Send a key (as a masked object) to other HSMs in the cluster; this event is recorded when a client action synchronizes the key
Clone HSMs	
CN_CLONE_SOURCE_INIT	Clone source start
CN_CLONE_SOURCE_STAGE1	Clone source end
CN_CLONE_TARGET_INIT	Clone target start
CN_CLONE_TARGET_STAGE1	Clone target end
Certificate-Based Authentication	
CN_CERT_AUTH_STORE_CERT	Store a certificate
CN_CERT_AUTH_VALIDATE_PEER_CERTS	Validate a certificate
CN_CERT_AUTH_SOURCE_KEY_EXCHANGE	Source key exchange
CN_CERT_AUTH_TARGET_KEY_EXCHANGE	Target key exchange

Operation Code (Opcode)	Description
HSM Instance Commands	
CN_INIT_TOKEN (0x1)	Initialize the HSM: Start
CN_INIT_DONE	Initialize the HSM: Complete
CN_GEN_KEY_ENC_KEY	Generate a key encryption key (KEK)
CN_GEN_PSWD_ENC_KEY (0x1d)	Generate a password encryption key (PEK)
CN_CLOSE_PARTITION_SESSIONS	Close a session on the HSM
CN_STORE_KBK_SHARE	Store the key backup key (KBK)
CN_SET_NODEID	Set the node ID of the HSM in the cluster
CN_ZEROIZE	Zeroize the HSM

Getting CloudWatch metrics for AWS CloudHSM

Use CloudWatch to monitor your AWS CloudHSM cluster in real time. The metrics can be grouped by region, cluster ID, or cluster ID and HSM ID.

The `AWS/CloudHSM` namespace includes the following metrics:

Metric	Description
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	The junction temperature of the hardware processor. The system shuts down if temperature reaches 110 degrees Centigrade.
HsmKeysSessionOccupied	The number of session keys being used by the HSM instance.
HsmKeysTokenOccupied	The number of token keys being used by the HSM instance and the cluster.
HsmSslCtxsOccupied	The number of end-to-end encrypted channels currently established for the HSM instance. Up to 2,048 channels are allowed.
HsmSessionCount	The number of open connections to the HSM instance. Up to 2,048 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	The number of additional users that can be created. This equals the maximum number of users (listed in HsmUsersMax) minus the users created to date.
HsmUsersMax	The maximum number of users that can be created on the HSM instance. Currently this is 1,024.
InterfaceEth2OctetsInput	The cumulative sum of incoming traffic to the HSM to date.
InterfaceEth2OctetsOutput	The cumulative sum of outgoing traffic to the HSM to date.

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 \(p. 501\)](#)
- [Identity and access management for AWS CloudHSM \(p. 503\)](#)
- [FIPS validation \(p. 508\)](#)
- [Resilience in AWS CloudHSM \(p. 509\)](#)
- [Infrastructure security in AWS CloudHSM \(p. 509\)](#)
- [AWS CloudHSM and VPC endpoints \(p. 510\)](#)
- [Update management in AWS CloudHSM \(p. 511\)](#)

Data protection in AWS CloudHSM

The AWS [shared responsibility model](#) applies to data protection in AWS CloudHSM. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

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). That way 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. We recommend TLS 1.2 or later.
- 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.

- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or 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 tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

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 \(p. 6\)](#).

Encryption in transit

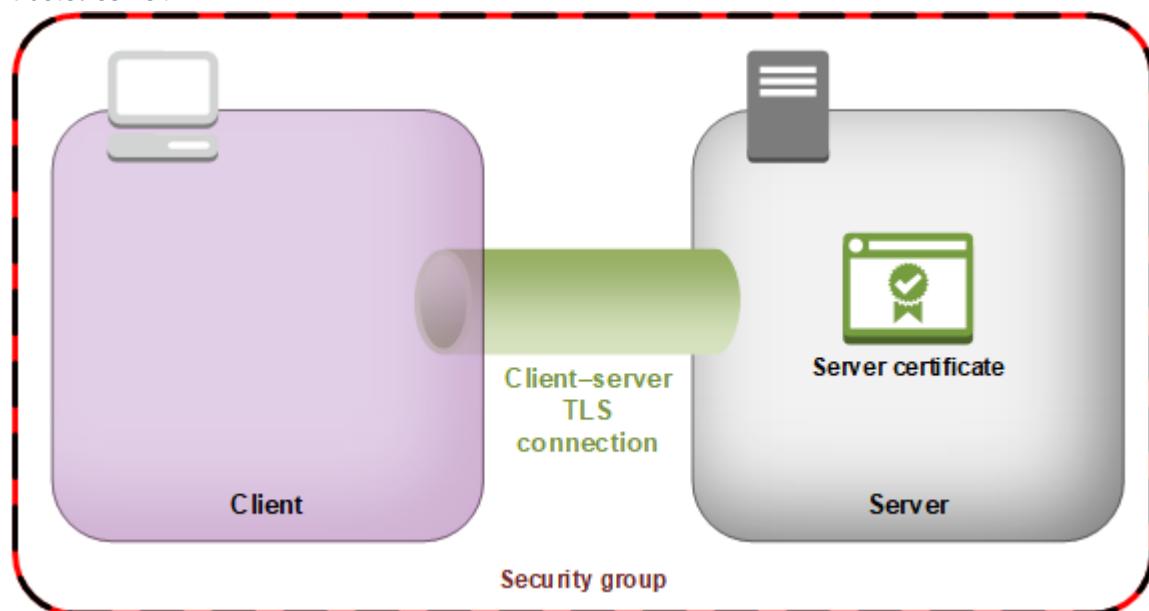
Communication between the AWS CloudHSM client and the HSM in your cluster is encrypted from end to end. This communication can be decrypted only by your client and your HSM. For more information, see [End-to-end encryption \(p. 502\)](#).

AWS CloudHSM client end-to-end encryption

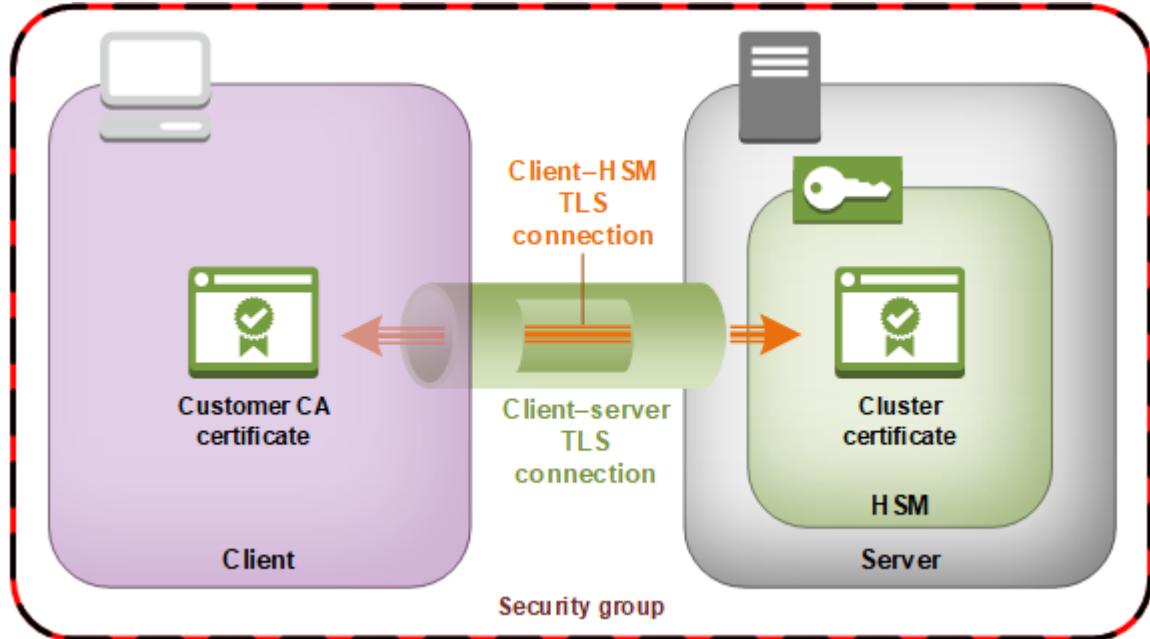
Communication between the client instance and the HSMs in your cluster is encrypted from end to end. Only your client and your HSMs can decrypt the communication.

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.



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 \(p. 504\)](#).
- **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 \(p. 505\)](#).

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 \(p. 505\)](#)
- [Power User Permissions \(p. 506\)](#)
- [Admin Permissions \(p. 506\)](#)

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.

```
{
```

```

"Version": "2012-10-17",
"Statement": [
    "Effect": "Allow",
    "Action": [
        "cloudhsm:DescribeClusters",
        "cloudhsm:DescribeBackups",
        "cloudhsm>ListTags"
    ],
    "Resource": "*"
}

```

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 \(p. 507\)](#).

```

{
    "Version": "2012-10-17",
    "Statement": [
        "Effect": "Allow",
        "Action": [
            "cloudhsm:DescribeClusters",
            "cloudhsm:DescribeBackups",
            "cloudhsm>CreateCluster",
            "cloudhsm>CreateHsm",
            "cloudhsm:RestoreBackup",
            "cloudhsm:CopyBackupToRegion",
            "cloudhsm:InitializeCluster",
            "cloudhsm>ListTags",
            "cloudhsm:TagResource",
            "cloudhsm:UntagResource",
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeNetworkInterfaceAttribute",
            "ec2:DetachNetworkInterface",
            "ec2:DeleteNetworkInterface",
            "ec2>CreateSecurityGroup",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:DescribeSecurityGroups",
            "ec2:DeleteSecurityGroup",
            "ec2:CreateTags",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "iam:CreateServiceLinkedRole"
        ],
        "Resource": "*"
    }
}

```

Example 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 \(p. 507\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudhsm:*",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeNetworkInterfaceAttribute",
                "ec2:DetachNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:CreateSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:DescribeSecurityGroups",
                "ec2:DeleteSecurityGroup",
                "ec2:CreateTags",
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*"
        }
    ]
}
```

Service-linked roles for AWS CloudHSM

The IAM policy that you created previously to [Customer managed policies for AWS CloudHSM \(p. 505\)](#) 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.

```
{
    "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.

```
{  
    "Version": "2018-06-12",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "clouhdsm.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

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 \(p. 505\)](#).

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 \(p. 10\)](#) 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 \(p. 45\)](#). 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

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 \(p. 12\)](#).

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 \(p. 16\)](#).

Authorization of users

With AWS CloudHSM, operations performed on the HSM require the credentials of an authenticated HSM user. For more information, see the section called ["Understanding HSM users" \(p. 59\)](#).

AWS CloudHSM and VPC endpoints

You can establish a private connection between your VPC and AWS CloudHSM by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#), a technology that enables you to privately access AWS CloudHSM APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS CloudHSM APIs. Traffic between your VPC and AWS CloudHSM does not leave the Amazon network.

Each interface endpoint is represented by one or more [Elastic Network Interfaces](#) in your subnets.

For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

Considerations for AWS CloudHSM VPC endpoints

Before you set up an interface VPC endpoint for AWS CloudHSM, ensure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

- AWS CloudHSM supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for AWS CloudHSM

You can create a VPC endpoint for the AWS CloudHSM service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

To create a VPC endpoint for AWS CloudHSM, use the following service name:

```
com.amazonaws.<region>.cloudhsmv2
```

For example, in the US West (Oregon) Region (`us-west-2`), the service name would be:

```
com.amazonaws.us-west-2.cloudhsmv2
```

To make it easier to use the VPC endpoint, you can enable a [private DNS hostname](#) for your VPC endpoint. If you select the **Enable Private DNS Name** option, the standard AWS CloudHSM DNS hostname (`https://cloudhsmv2.<region>.amazonaws.com`) resolves to your VPC endpoint.

This option makes it easier to use the VPC endpoint. The AWS SDKs and AWS CLI use the standard AWS CloudHSM DNS hostname by default, so you do not need to specify the VPC endpoint URL in applications and commands.

For more information, see [Accessing a service through an interface endpoint](#) in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for AWS CloudHSM

You can attach an endpoint policy to your VPC endpoint that controls access to AWS CloudHSM. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for AWS CloudHSM actions

The following is an example of an endpoint policy for AWS CloudHSM. When attached to an endpoint, this policy grants access to the listed AWS CloudHSM actions for all principals on all resources.

```
{  
    "Statement": [  
        {  
            "Principal": "*",  
            "Effect": "Allow",  
            "Action": [  
                "cloudhsm:CopyBackupToRegion",  
                "cloudhsm>CreateCluster",  
                "cloudhsm>CreateHsm",  
                "cloudhsm>DeleteBackup",  
                "cloudhsm>DeleteCluster",  
                "cloudhsm>DeleteHsm",  
                "cloudhsm:DescribeBackups",  
                "cloudhsm:DescribeClusters",  
                "cloudhsm:InitializeCluster",  
                "cloudhsm>ListTags",  
                "cloudhsm:ModifyBackupAttributes",  
                "cloudhsm:ModifyCluster",  
                "cloudhsm:RestoreBackup",  
                "cloudhsm:TagResource",  
                "cloudhsm:UntagResource"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

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 \(p. 512\)](#)
- [Lost connection to the cluster \(p. 523\)](#)
- [Keep HSM users in sync across HSMs in the cluster \(p. 525\)](#)
- [Verify the performance of the HSM \(p. 526\)](#)
- [Resolving cluster creation failures \(p. 529\)](#)
- [Missing AWS CloudHSM audit logs in CloudWatch \(p. 531\)](#)
- [Retrieving client configuration logs \(p. 531\)](#)
- [Custom IVs with non-compliant length for AES key wrap \(p. 533\)](#)
- [Client SDK 3 key synchronization failures \(p. 534\)](#)

Known issues

AWS CloudHSM has the following known issues. Choose a topic to learn more.

Topics

- [Known issues for all HSM instances \(p. 512\)](#)
- [Known issues for the PKCS #11 library \(p. 515\)](#)
- [Known issues for the JCE SDK \(p. 519\)](#)
- [Known issues for the OpenSSL Dynamic Engine \(p. 521\)](#)
- [Known issues for Amazon EC2 instances running Amazon Linux 2 \(p. 522\)](#)
- [Known issues for integrating third-party applications \(p. 523\)](#)

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.

Topics

- [Issue: AES key wrapping uses PKCS #5 padding instead of providing a standards-compliant implementation of key wrap with zero padding \(p. 513\)](#)
- [Issue: The client daemon requires at least one valid IP address in its configuration file to successfully connect to the cluster \(p. 513\)](#)
- [Issue: There was an upper limit of 16 KB on data that can be hashed and signed by AWS CloudHSM \(p. 513\)](#)
- [Issue: Imported keys could not be specified as nonexportable \(p. 514\)](#)

- Issue: The default mechanism for the wrapKey and unWrapKey commands in the key_mgmt_util has been removed (p. 514)
- Issue: If you have a single HSM in your cluster, HSM failover does not work correctly (p. 514)
- 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 (p. 514)
- Issue: Digest operations with HMAC keys of size greater than 800 bytes are not supported (p. 514)
- Issue: The client_info tool, distributed with Client SDK 3, deletes the contents of the path specified by the optional output argument (p. 515)
- Issue: You receive an error when running the SDK 5 configure tool using the --cluster-id argument in containerized environments (p. 515)

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 and zero padding is not supported.

- **Impact:** There is no impact if you wrap and unwrap using this algorithm within AWS CloudHSM. However, keys wrapped with AWS CloudHSM cannot be unwrapped within other HSMs or software that expects compliance to the no-padding specification. This is because eight bytes of padding data might be added to the end of your key data during 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 Wrap with PKCS #5 Padding on an 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:** With the 3.1.0 client and software release, AWS CloudHSM provides standards-compliant options for AES key wrapping. For more information, see [AES Key Wrapping \(p. 95\)](#).

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 \(Linux\) \(p. 28\)](#) or [Install and Configure the AWS CloudHSM Client \(Windows\) \(p. 30\)](#).

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 \(p. 232\)](#) or [unWrapKey \(p. 226\)](#) 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 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 release. You must upgrade to this client to benefit from the fix.

Issue: Digest operations with HMAC keys of size greater than 800 bytes are not supported

- **Impact:** HMAC keys larger than 800 bytes can be generated on or imported into the HSM. However, if you use this larger key in a digest operation via the JCE or key_mgmt_util, the operation will fail. Note that if you are using PKCS11, HMAC keys are limited to a size of 64 bytes.
- **Workaround:** If you will be using HMAC keys for digest operations on the HSM, ensure the size is smaller than 800 bytes.
- **Resolution status:** None at this time.

Issue: The client_info tool, distributed with Client SDK 3, deletes the contents of the path specified by the optional output argument

- **Impact:** All existing files and sub-directories under the specified output path may be permanently lost.
- **Workaround:** Do not use the optional argument `--output path` when using the `client_info` tool.
- **Resolution status:** This issue has been resolved in the [Client SDK 3.3.2 release](#). You must upgrade to this client to benefit from the fix.

Issue: You receive an error when running the SDK 5 configure tool using the --cluster-id argument in containerized environments

You receive the following error when using the `--cluster-id` argument with the Configure Tool:

```
No credentials in the property bag
```

This error is caused by an update to Instance Metadata Service Version 2 (IMDSv2). For more information, see the [IMDSv2](#) documentation.

- **Impact:** This issue will impact users running the configure tool on SDK versions 5.5.0 and later in containerized environments and utilizing EC2 instance metadata to provide credentials.
- **Workaround:** Set the PUT response hop limit to at least two. for guidance on how to do this, see [Configure the instance metadata options](#).

Known issues for the PKCS #11 library

Topics

- Issue: AES key wrap in version 3.0.0 of the PKCS #11 library does not validate IVs before use (p. 516)
- Issue: PKCS#11 SDK 2.0.4 and earlier versions always used the default IV of 0xA6A6A6A6A6A6A6A6 for AES key wrap and unwrap (p. 516)
- Issue: The CKA_DERIVE attribute was not supported and was not handled (p. 516)
- Issue: The CKA_SENSITIVE attribute was not supported and was not handled (p. 517)
- Issue: Multipart hashing and signing are not supported (p. 517)
- 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 (p. 517)
- Issue: You could not hash more than 16KB of data (p. 517)
- Issue: Buffers for the C_Encrypt and C_Decrypt API operations cannot exceed 16 KB when using the CKM_AES_GCM mechanism (p. 517)
- Issue: Elliptic-curve Diffie-Hellman (ECDH) key derivation is executed partially within the HSM (p. 518)
- Issue: Verification of secp256k1 signatures fails on EL6 platforms such as CentOS6 and RHEL6 (p. 518)
- Issue: Incorrect sequence of function calls gives undefined results instead of failing (p. 518)
- Issue: Read Only Session is not supported in SDK 5 (p. 519)

- Issue: [cryptoki.h header file is Windows-only \(p. 519\)](#)

Issue: AES key wrap in version 3.0.0 of the PKCS #11 library does not validate IVs before use

If you specify an IV shorter than 8 bytes in length, it is padded with unpredictable bytes before use.

Note

This impacts `C_WrapKey` with `CKM_AES_KEY_WRAP` mechanism only.

- **Impact:** If you provide an IV that is shorter than 8 bytes in version 3.0.0 of PKCS #11 library, you may be unable to unwrap the key.
- **Workarounds:**
 - We strongly recommend you upgrade to version 3.0.1 or higher of the PKCS #11 library, which properly enforces IV length during AES key wrap. Amend your wrapping code to pass a NULL IV, or specify the default IV of `0xA6A6A6A6A6A6A6A6`. For more information, see [Custom IVs with Non-Compliant Length for AES Key Wrap \(p. 533\)](#).
 - If you wrapped any keys with version 3.0.0 of the PKCS #11 library using an IV shorter than 8 bytes, reach out to us for [support](#).
- **Resolution status:** This issue has been resolved in version 3.0.1 of the PKCS #11 library. To wrap keys using AES key wrap, specify an IV that is NULL or 8 bytes long.

Issue: PKCS#11 SDK 2.0.4 and earlier versions always used the default IV of `0xA6A6A6A6A6A6A6A6` for AES key wrap and unwrap

User-provided IVs were silently ignored.

Note

This impacts `C_WrapKey` with `CKM_AES_KEY_WRAP` mechanism only.

- **Impact:**
 - If you used PKCS#11 SDK 2.0.4 or an earlier version and a user-provided IV, your keys are wrapped with the default IV of `0xA6A6A6A6A6A6A6A6`.
 - If you used PKCS#11 SDK 3.0.0 or later and a user-provided IV, your keys are wrapped with the user-provided IV.
- **Workarounds:**
 - To unwrap keys wrapped with PKCS#11 SDK 2.0.4 or earlier use the default IV of `0xA6A6A6A6A6A6A6A6`.
 - To unwrap keys wrapped with PKCS#11 SDK 3.0.0 or later, use the user-provided IV.
- **Resolution status:** We strongly recommend that you amend your wrapping and unwrapping code to pass a NULL IV, or specify the default IV of `0xA6A6A6A6A6A6A6A6`.

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

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 happens 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.
- **Workaround:** We recommend using either Amazon Linux 1 or any EL7 platform if your PKCS#11 application needs to verify secp256k1 signatures. Alternatively, upgrade to a version of the OpenSSL package that supports the secp256k1 curve.
- **Resolution status:** We are implementing fixes to fall back to the HSM if local curve validation is not available. The updated PKCS#11 library will be announced on the [version history \(p. 260\)](#) page.

Issue: Incorrect sequence of function calls gives undefined results instead of failing

- **Impact:** If you call an incorrect sequence of functions, the final result is incorrect even though the individual function calls return success. For instance, decrypted data may not match the original plaintext or signatures may fail to verify. This issue affects both single part and multi-part operations.

Examples of incorrect function sequences:

- C_EncryptInit/C_EncryptUpdate followed by C_Encrypt
- C_DecryptInit/C_DecryptUpdate followed by C_Decrypt
- C_SignInit/C_SignUpdate followed by C_Sign
- C_VerifyInit/C_VerifyUpdate followed by C_Verify
- C_FindObjectsInit followed by c_FindObjectsInit

- **Workaround:** Your application should, in compliance with the PKCS #11 specification, use the right sequence of function calls for both single and multi-part operations. Your application should not rely on the CloudHSM PKCS #11 library to return an error under this circumstance.

Issue: Read Only Session is not supported in SDK 5

- **Issue:** SDK 5 does not support opening Read-Only sessions with `C_OpenSession`.
- **Impact:** If you attempt to call `C_OpenSession` without providing `CKF_RW_SESSION`, the call will fail with the error `CKR_FUNCTION_FAILED`.
- **Workaround:** When opening a session, you must pass the `CKF_SERIAL_SESSION` | `CKF_RW_SESSION` flags to the `C_OpenSession` function call.

Issue: `cryptoki.h` header file is Windows-only

- **Issue:** With AWS CloudHSM Client SDK 5 versions 5.0.0 through 5.4.0 on Linux, the header file `/opt/cloudhsm/include/pkcs11/cryptoki.h` is only compatible with Windows operating systems.
- **Impact:** You may encounter issues when trying to include this header file in your application on Linux-based operating systems.
- **Resolution status:** Upgrade to AWS CloudHSM Client SDK 5 version 5.4.1 or above, which includes a Linux-compatible version of this header file.

Known issues for the JCE SDK

Topics

- [Issue: When working with asymmetric key pairs, you see occupied key capacity even when you are not explicitly creating or importing keys \(p. 519\)](#)
- [Issue: You cannot specify attributes when unwrapping keys \(p. 520\)](#)
- [Issue: The JCE KeyStore is read only \(p. 520\)](#)
- [Issue: Buffers for AES-GCM encryption cannot exceed 16,000 bytes \(p. 520\)](#)
- [Issue: Elliptic-curve Diffie-Hellman \(ECDH\) key derivation is executed partially within the HSM \(p. 520\)](#)
- [Issue: KeyGenerator and KeyAttribute incorrectly interprets key size parameter as number of bytes instead of bits \(p. 521\)](#)

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 and 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

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 \(p. 366\)](#).

- **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.

Issue: KeyGenerator and KeyAttribute incorrectly interprets key size parameter as number of bytes instead of bits

When generating a key using the `init` function of the [KeyGenerator class](#) or the `SIZE` attribute of the [AWS CloudHSM KeyAttribute enum \(p. 376\)](#), the API incorrectly expects the argument to be the number of key bytes, when it should instead be the number of key bits.

- **Impact:** Client SDK versions 5.4.0 through 5.4.2 incorrectly expects the key size to be provided to the specified APIs as bytes.
- **Workaround:** Convert the key size from bits to bytes before using the KeyGenerator class or KeyAttribute enum to generate keys using the AWS CloudHSM JCE provider if using Client SDK versions 5.4.0 through 5.4.2.
- **Resolution status:** Upgrade your client SDK version to 5.5.0 or later, which includes a fix to correctly expect key sizes in bits when using the KeyGenerator class or KeyAttribute enum to generate keys.

Known issues for the OpenSSL Dynamic Engine

These are the known issues for OpenSSL Dynamic Engine

Topics

- [Issue: You cannot install AWS CloudHSM OpenSSL Dynamic Engine on RHEL6 and CentOS6 \(p. 521\)](#)
- [Issue: Only RSA offload to the HSM is supported by default \(p. 521\)](#)
- [Issue: RSA encryption and decryption with OAEP padding using a key on the HSM is not supported \(p. 522\)](#)
- [Issue: Only private key generation of RSA and ECC keys is offloaded to the HSM \(p. 522\)](#)
- [Issue: You cannot install OpenSSL Dynamic Engine for Client SDK 3 on RHEL 8, CentOS 8, or Ubuntu 18.04 LTS \(p. 522\)](#)

Issue: You cannot install AWS CloudHSM OpenSSL Dynamic Engine on RHEL6 and CentOS6

- **Impact:** The OpenSSL Dynamic Engine only [supports OpenSSL 1.0.2\[f+\] \(p. 314\)](#). By default, RHEL 6 and CentOS 6 ship with OpenSSL 1.0.1.
- **Workaround:** Upgrade the OpenSSL library on RHEL 6 and CentOS 6 to version 1.0.2[f+].

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 [PKCS #11 library \(p. 324\)](#) or the [JCE provider \(p. 352\)](#).
- **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.

Issue: You cannot install OpenSSL Dynamic Engine for Client SDK 3 on RHEL 8, CentOS 8, or Ubuntu 18.04 LTS

- **Impact:** By default, RHEL 8, CentOS 8, and Ubuntu 18.04 LTS ship a version of OpenSSL that is not compatible with OpenSSL Dynamic Engine for Client SDK 3.
- **Workaround:** Use a Linux platform that provides support for OpenSSL Dynamic Engine. For more information about supported platforms, see [Supported Platforms \(p. 314\)](#).
- **Resolution status:** AWS CloudHSM supports these platforms with OpenSSL Dynamic Engine for Client SDK 5. For more information, see [Supported Platforms \(p. 314\)](#) and [OpenSSL Dynamic Engine \(p. 347\)](#).

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

You see the following error returned upon running the AWS CloudHSM [cloudhsm_mgmt_util \(p. 106\)](#) or [key_mgmt_util \(p. 152\)](#):

```
/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 release. You must upgrade to this client to benefit from the fix.

Known issues for integrating third-party applications

Issue: Oracle sets the PKCS #11 attribute CKA_MODIFIABLE during master key generation, but the HSM does not support it

This limit is defined in the PKCS #11 library. For more information, see annotation 1 on [Supported PKCS #11 Attributes \(p. 336\)](#).

- **Impact:** Oracle master key creation fails.
- **Workaround:** Set the special environment variable CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE to TRUE when creating a new master key. This environment variable is only needed for master key generation and you do not need to use this environment variable for anything else. For example, you would use this variable for the first master key you create and then you would only use this environment variable again if you wanted to rotate your master key edition. For more information, see [Generate the Oracle TDE Master Encryption Key \(p. 454\)](#).
- **Resolution status:** We are improving the HSM firmware to fully support the CKA_MODIFIABLE attribute. Updates will be announced in the AWS CloudHSM forum and on the version history page

Lost connection to the cluster

When you [configured the AWS CloudHSM client \(p. 30\)](#), 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 cloudfsm-client
```

Amazon Linux 2

```
$ sudo service cloudfsm-client stop
```

CentOS 7

```
$ sudo service cloudfsm-client stop
```

CentOS 8

```
$ sudo service cloudfsm-client stop
```

RHEL 7

```
$ sudo service cloudfsm-client stop
```

RHEL 8

```
$ sudo service cloudfsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service cloudfsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service cloudfsm-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/cloudfsm/bin/configure -a <IP address>
```

4. Use the following command to start the client.

Amazon Linux

```
$ sudo start cloudfsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- 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
```

Keep HSM users in sync across HSMs in the cluster

To [manage your HSM's users \(p. 59\)](#), 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 \(p. 59\)](#), 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 [???](#) (p. 61).

Verify the performance of the HSM

This topic describes how to verify HSM performance with Client SDK 3.

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 Client SDK 3. For more information about installing the client on a Linux EC2 instance, see [Install and configure the AWS CloudHSM client \(Linux\)](#) (p. 28). For more information about installing the client on a Windows instance, see [Install and configure the AWS CloudHSM client \(Windows\)](#) (p. 30).

After you install and configure the AWS CloudHSM client, run the following command to start it.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- 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
```

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.

```
/opt/cloudhsm/bin/pkpspeed -s CU user name -p password

SDK Version: 2.03

Available Ciphers:
    AES_128
    AES_256
    3DES
    RSA (non-CRT. modulus size can be 2048/3072)
    RSA_CRT (same as RSA)

For RSA, Exponent will be 65537

Current FIPS mode is: 00002
Enter the number of thread [1-10]: 3
Enter the cipher: RSA_CRT
Enter modulus length: 2048
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 245 bytes...
[Test duration is 60 seconds]

Do you want to use static key[y/n] (Make sure that KEK is available)?n
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password

Please select the test you want to run
```

```
RSA non-CRT----->A
RSA CRT----->B
Basic 3DES CBC----->C
Basic AES----->D
FIPS Random----->H
Random----->I
AES GCM ----->K

exit----->X
B

Running 4 threads for 25 sec

Enter mod size(2048/3072):2048
Do you want to use Token key[y/n]n
Do you want to use static key[y/n] (Make sure that KEK is available)? n
OPERATIONS/second      821/1
OPERATIONS/second      833/1
OPERATIONS/second      845/1
OPERATIONS/second      835/1
OPERATIONS/second      837/1
OPERATIONS/second      836/1
OPERATIONS/second      837/1
OPERATIONS/second      849/1
OPERATIONS/second      841/1
OPERATIONS/second      856/1
OPERATIONS/second      841/1
OPERATIONS/second      847/1
OPERATIONS/second      838/1
OPERATIONS/second      843/1
OPERATIONS/second      852/1
OPERATIONS/second      837/
```

Example – Using pkpspeed to test AES performance

Linux

Use these instructions for Linux and compatible operating systems.

```
/opt/cloudhsm/bin/pkpspeed -s <CU user name> -p <password>

SDK Version: 2.03

Available Ciphers:
    AES_128
    AES_256
    3DES
    RSA (non-CRT. modulus size can be 2048/3072)
    RSA_CRT (same as RSA)
For RSA, Exponent will be 65537

Current FIPS mode is: 00000002
Enter the number of thread [1-10]: 1
Enter the cipher: AES_256
Enter the data size [1-16200]: 8192
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 8192 bytes...
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password
```

```
login as USER
Initializing Cfm2 library
SDK Version: 2.03

Current FIPS mode is: 00000002
Please enter the number of threads [MAX=400] : 1
Please enter the time in seconds to run the test [MAX=600]: 20

Please select the test you want to run

RSA non-CRT----->A
RSA CRT----->B
Basic 3DES CBC----->C
Basic AES----->D
FIPS Random----->H
Random----->I
AES GCM ----->K

exit----->X
D

Running 1 threads for 20 sec

Enter the key size(128/192/256):256
Enter the size of the packet in bytes[1-16200]:8192
OPERATIONS/second      9/1
OPERATIONS/second      10/1
OPERATIONS/second      11/1
OPERATIONS/second      10/1
OPERATIONS/second      10/1
OPERATIONS/second      10/...
```

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 \(p. 507\)](#).

Topics

- [Add the missing permission \(p. 529\)](#)
- [Create the service-linked role manually \(p. 530\)](#)
- [Use a nonfederated user \(p. 530\)](#)

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 \(p. 12\)](#) 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 \(p. 10\)](#) with the `iam:CreateServiceLinkedRole` permission, or give an existing non-federated user the `iam:CreateServiceLinkedRole` permission. Then have that user [create a cluster \(p. 12\)](#) 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": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "clouhdsm.amazonaws.com"
                    }
                }
            ]
        },
        "RoleId": "AROAJ4I6WN5QVGG5G7CBY",
        "CreateDate": "2017-12-19T20:53:12Z",
        "RoleName": "AWSServiceRoleForCloudHSM",
        "Path": "/aws-service-role/clouhdsm.amazonaws.com/",
        "Arn": "arn:aws:iam::111122223333:role/aws-service-role/clouhdsm.amazonaws.com/AWSServiceRoleForCloudHSM"
    }
}
```

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 \(p. 507\)](#) 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 \(p. 507\)](#), as well as [Creating a Service-Linked Role](#) in the IAM User Guide.

Retrieving client configuration logs

AWS CloudHSM offers tools for Client SDK 3 and Client SDK 5 to gather information about your environment for AWS Support to troubleshoot problems.

Topics

- [Client SDK 3 support tool \(p. 531\)](#)
- [Client SDK 5 support tool \(p. 531\)](#)

Client SDK 3 support tool

The script extracts the following information:

- Operating system and its current version
- Client configuration information from `cloudfsm_client.cfg`, `cloudfsm_mgmt_util.cfg`, and `application.cfg` files
- Client logs from the location specific to the platform
- Cluster and HSM information by using `cloudfsm_mgmt_util`
- OpenSSL information
- Current client and build version
- Installer version

Running the client tool for Client SDK 3

The script creates an output file with all the gathered information. The script creates the output file inside the `/tmp` directory.

Linux: `/opt/cloudfsm/bin/client_info`

Windows: `C:\Program Files\Amazon\CloudHSM\client_info`

Warning

This script has a known issue for Client SDK 3 versions 3.1.0 through 3.3.1. We strongly recommend you upgrade to version 3.3.2 which includes a fix for this issue. Please refer to the [Known Issues](#) page for more information before using this tool.

Client SDK 5 support tool

The script extracts the following information:

- The configuration file for the Client SDK 5 component

- Available log files
- Current version of the operating system
- Package information

Running the client tool for Client SDK 5

Client SDK 5 includes a client support tool for each component, but all tools function the same. Run the tool to create an output file with all the gathered information.

The tools use a syntax like this:

```
[ pkcs11 | dyn | jce ]_info
```

For example, to gather information for support from a Linux host running PKCS #11 library and have the system write to the default directory, you would run this command:

```
/opt/cloudhsm/bin/pkcs11_info
```

The tool creates the output file inside the /tmp directory.

PKCS #11 library

To gather support data for PKCS #11 library on Linux

- Use the support tool to gather data.

```
/opt/cloudhsm/bin/pkcs11_info
```

To gather support data for PKCS #11 library on Windows

- Use the support tool to gather data.

```
C:\Program Files\Amazon\CloudHSM\bin\pkcs11_info.exe
```

OpenSSL Dynamic Engine

To gather support data for OpenSSL Dynamic Engine on Linux

- Use the support tool to gather data.

```
/opt/cloudhsm/bin/dyn_info
```

JCE provider

To gather support data for JCE provider on Linux

- Use the support tool to gather data.

```
/opt/cloudhsm/bin/jce_info
```

To gather support data for JCE provider on Windows

- Use the support tool to gather data.

```
c:\Program Files\Amazon\CloudHSM\bin\jce_info.exe
```

Custom IVs with non-compliant length for AES key wrap

This troubleshooting topic helps you determine if your application generates irrecoverable wrapped keys. If you are impacted by this issue, use this topic to address the problem.

Topics

- [Determine whether your code generates irrecoverable wrapped keys \(p. 533\)](#)
- [Actions you must take if your code generates irrecoverable wrapped keys \(p. 534\)](#)

Determine whether your code generates irrecoverable wrapped keys

You are impacted only if you meet *all* the conditions below:

Condition	How do I know?
Your application uses PKCS #11 library	The PKCS #11 library is installed as the <code>libpkcs11.so</code> file in your <code>/opt/cloudhsm/lib</code> folder. Applications written in the C language generally use the PKCS #11 library directly, while application written in Java may be using the library indirectly via a Java abstraction layer. If you're using Windows, you are NOT affected, as PKCS #11 library is not presently available for Windows.
Your application specifically uses version 3.0.0 of the PKCS #11 library	If you received an email from the AWS CloudHSM team, you are likely using version 3.0.0 of the PKCS #11 library. To check the software version on your application instances, use this command: <pre>rpm -qa grep ^cloudhsm</pre>
You wrap keys using AES key wrapping	AES key wrapping means you use an AES key to wrap out some other key. The corresponding mechanism name is <code>CKM_AES_KEY_WRAP</code> . It is used with the function <code>C_WrapKey</code> . Other AES based wrapping mechanisms that use initialization vectors (IVs), such as <code>CKM_AES_GCM</code> and <code>CKM_CLOUDHSM_AES_GCM</code> , are not affected

Condition	How do I know?
	by this issue. Learn more about functions and mechanisms (p. 329) .
You specify a custom IV when calling AES key wrapping, and the length of this IV is shorter than 8	AES key wrap is generally initialized using a CK_MECHANISM structure as follows: <pre>CK_MECHANISM mech = {CKM_AES_KEY_WRAP, IV_POINTER, IV_LENGTH};</pre> This issue applies to you only if: <ul style="list-style-type: none">IV_POINTER is not NULLIV_LENGTH is less than 8 bytes

If you do not meet all the conditions above, you may stop reading now. Your wrapped keys can be unwrapped properly, and this issue does not impact you. Otherwise, see the section called “Actions you must take if your code generates irrecoverable wrapped keys” (p. 534).

Actions you must take if your code generates irrecoverable wrapped keys

You should take the following three steps:

1. Immediately upgrade your PKCS #11 library to a newer version

- [Latest PKCS #11 library for Amazon Linux, CentOS 6 and RHEL 6 \(p. 317\)](#)
- [Latest PKCS #11 library for Amazon Linux 2, CentOS 7 and RHEL 7 \(p. 317\)](#)
- [Latest PKCS #11 library for Ubuntu 16.04 LTS \(p. 317\)](#)

2. Update your software to use a standards-compliant IV

We strongly recommend you follow our sample code and simply specify a NULL IV, which causes the HSM to utilize the standards-compliant default IV. Alternatively, you may explicitly specify the IV as 0xA6A6A6A6A6A6A6 with a corresponding IV length of 8. We do not recommend using any other IV for AES key wrapping, and will explicitly disable custom IVs for AES key wrapping in a future version of the PKCS #11 library.

Sample code for properly specifying the IV appears in [aes_wrapping.c](#) on GitHub.

3. Identify and recover existing wrapped keys

You should identify any keys you wrapped using version 3.0.0 of the PKCS #11 library, and then contact support for assistance (<https://aws.amazon.com/support>) in recovering these keys.

Important

This issue only impacts keys wrapped with version 3.0.0 of the PKCS #11 library. You can wrap keys using earlier versions (2.0.4 and lower-numbered packages) or later versions (3.0.1 and higher-numbered packages) of the PKCS #11 library.

Client SDK 3 key synchronization failures

In Client SDK 3, if client-side synchronization fails, AWS CloudHSM makes a best-effort response to clean up any unwanted keys that may have been created (and are now unwanted). This process involves

removing unwanted key material immediately or marking unwanted material for later removal. In both these cases, the resolution does not require any action from you. In the rare case that AWS CloudHSM cannot remove *and* cannot mark unwanted key material, you must delete the key material.

Problem: You attempt a token key generation, import, or unwrap operation and see errors that specify a failure to *tombstone*.

```
2018-12-24T18:28:54Z liquidSecurity ERR: print_node_ts_status:  
[create_object_min_nodes]Key: 264617 failed to tombstone on node:1
```

Cause: AWS CloudHSM was unsuccessful removing *and* marking unwanted key material.

Resolution: An HSM in your cluster contains unwanted key material that is not marked as unwanted. You must manually remove the key material. To manually delete unwanted key material, use `key_mgmt_util` (KMU) or an API from the PKCS #11 library or the JCE provider. For more information, see [deleteKey \(p. 160\)](#) or [Client SDKs \(p. 258\)](#).

To make token keys more durable, AWS CloudHSM fails key creation operations that don't succeed on the minimum number of HSMs specified in client-side synchronization settings. For more information, see [Key Synchronization in AWS CloudHSM \(p. 88\)](#).

Document history

This topic describes significant updates to the *AWS CloudHSM User Guide*.

Topics

- [Recent updates \(p. 536\)](#)
- [Earlier updates \(p. 540\)](#)

Recent updates

The following table describes significant changes to this documentation since April 2018. In addition to major changes listed here, we also update the documentation frequently to improve the descriptions and examples, and to address the feedback you send us. To be notified about significant changes, use the link in the upper right corner to subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Added new release	Released AWS CloudHSM client version 5.5.0, which adds support for OpenJDK 11, Keytool, and Jarsigner integration and additional mechanisms to the JCE provider. Resolves a known issue regarding a KeyGenerator class incorrectly interpreting key size parameter as number of bytes instead of bits.	May 13, 2022
Added new release	Released AWS CloudHSM client version 5.4.2, which includes improved stability and bug fixes.	March 18, 2022
Added new release	Released AWS CloudHSM client version 5.4.1, which includes improved stability and bug fixes.	February 10, 2022
Added new release	Released the AWS CloudHSM JCE provider version 5.4.0 for Windows platforms.	February 1, 2022
Added new release	Released the AWS CloudHSM client version 5.4.0, which adds initial support for the JCE provider for all Linux platforms.	January 28, 2022
Added new release	Released AWS CloudHSM client version 5.3.0, which adds additional support for the OpenSSL Dynamic Engine including ECDSA, OpenSSL 1.0.2, and platforms: Amazon Linux, Amazon Linux 2, Centos 7.8+, and RHEL 7.8+.	January 3, 2022

Added new release	Released AWS CloudHSM client version 3.4.4 for all platforms.	January 3, 2022
Added new release	Released AWS CloudHSM client version 3.4.3 for all platforms.	December 20, 2021
Added new release	Released AWS CloudHSM client version 3.4.2 for all platforms.	December 15, 2021
Added new release	Released AWS CloudHSM client version 3.4.1 for all platforms.	December 10, 2021
Added new release	Released AWS CloudHSM client version 5.2.1, which adds improved stability and bug fixes.	October 4, 2021
Added new release	Released AWS CloudHSM client version 3.4.0 for all platforms.	August 25, 2021
Added new release	Released AWS CloudHSM client version 5.2.0, which adds new supported key types and mechanisms for the PKCS #11 library.	August 3, 2021
Added new release	Released AWS CloudHSM client version 3.3.2, which resolves an issue with the <code>client_info</code> script.	July 2, 2021
Added new release	Released AWS CloudHSM client version 5.1.0, which adds new supported mechanisms for the PKCS #11 library.	June 1, 2021
Added new release	Released AWS CloudHSM client version 3.3.1 for Amazon Linux, Amazon Linux 2, CentOS 7.3+, CentOS 8, Red Hat Enterprise Linux (RHEL) 7.3+, Red Hat Enterprise Linux (RHEL) 8, Ubuntu 16.04 LTS, Ubuntu 18.04 LTS, Windows Server 2012, Windows Server 2012 R2 and Windows Server 2016.	April 26, 2021
Added new release	Released AWS CloudHSM client version 5.0.1, which introduces initial support for OpenSSL Dynamic Engine to Client SDK 5, offering SSL/TLS offload for Red Hat Enterprise Linux (RHEL) 8.3+, CentOS 8.3+, and Ubuntu 18.04 LTS.	April 8, 2021

Added new release	Released AWS CloudHSM client version 5.0.0, which introduces an all-new Client SDK with different requirements, capabilities, and platform support. AWS CloudHSM now offers two versions of the Client SDK, Client SDK 5 and Client SDK 3.	March 12, 2021
Added new content	Added interface VPC endpoint, an AWS feature that allows you to create a private connection between your VPC and AWS CloudHSM without requiring access over the internet or through a NAT device, a VPN connection, or an AWS Direct Connect connection.	February 10, 2021
Added new release	Released AWS CloudHSM client version 3.3.0 for all platforms.	February 3, 2021
Add new content	Added managed backup retention, a feature that automatically deletes old backups.	November 18, 2020
Add new content	Added a compliance report that analyzes the AWS CloudHSM Client SDK 3.2.1 implementation of the PKCS #11 library with the PKCS #11 standard.	October 29, 2020
Added new release	Released AWS CloudHSM client version 3.2.1 for all platforms. This release adds support for additional platforms, including Red Hat Enterprise Linux 8, CentOS 8, and Ubuntu 18.04 LTS.	October 8, 2020
Added new content	Added documentation that describes key synchronization settings in AWS CloudHSM.	September 1, 2020
Added new release	Released AWS CloudHSM client version 3.2.0 for all platforms.	August 31, 2020
Added new release	Released AWS CloudHSM client version 3.1.2 for all platforms.	July 30, 2020
Added new release	Released AWS CloudHSM client version 3.1.1 for all platforms.	June 3, 2020
Added new release	Released AWS CloudHSM client version 3.1.0 for all platforms.	May 21, 2020

Added new release (p. 536)	Released AWS CloudHSM client version 3.0.1 for all platforms.	April 20, 2020
Added new release (p. 536)	Released AWS CloudHSM client version 3.0.0 for Windows Server platform.	October 30, 2019
Added new release (p. 536)	Released AWS CloudHSM client version 3.0.0 for all platforms, except Windows.	October 22, 2019
Added new release (p. 536)	Released AWS CloudHSM client version 2.0.4 for all platforms.	August 26, 2019
Added new release (p. 536)	Released AWS CloudHSM client version 2.0.3 for all platforms.	May 13, 2019
Added new release (p. 536)	Released AWS CloudHSM client version 2.0.1 for all platforms.	March 21, 2019
Added new release (p. 536)	Released AWS CloudHSM client version 2.0.0 for all platforms.	February 6, 2019
Added region support (p. 536)	Added AWS CloudHSM support for the EU (Stockholm) and AWS GovCloud (US-East) regions.	December 19, 2018
Added new release (p. 536)	Released AWS CloudHSM client version 1.1.2 for Windows.	November 20, 2018
Updated known issues	New content was added to the Troubleshooting guide.	November 8, 2018
Added new release (p. 536)	Released AWS CloudHSM client version 1.1.2 for Linux platforms.	November 8, 2018
Added region support (p. 536)	Added AWS CloudHSM support for the EU (Paris) and Asia Pacific (Seoul) regions.	October 24, 2018
Added new content	Added the ability to delete and restore AWS CloudHSM backups.	September 10, 2018
Added new content	Added automatic audit log delivery to Amazon CloudWatch Logs.	August 13, 2018
Added new content	Added the ability to copy an AWS CloudHSM cluster backup across regions.	July 30, 2018
Added region support (p. 536)	Added AWS CloudHSM support for the EU (London) region.	June 13, 2018

Added new content	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.	May 10, 2018
Added new release	Added a Windows AWS CloudHSM client.	April 30, 2018

Earlier updates

The following table describes the important changes to the AWS CloudHSM prior to 2018.

Change	Description	Date
New content	Added quorum authentication (M of N access control) for crypto officers (COs). For more information, see Managing quorum authentication (M of N access control) (p. 74) .	November 9, 2017
Update	Added documentation about using the <code>key_mgmt_util</code> command line tool. For more information, see key_mgmt_util command reference (p. 155) .	November 9, 2017
New content	Added Oracle Transparent Data Encryption. For more information, see Oracle database encryption (p. 451) .	October 25, 2017
New content	Added SSL Offload. For more information, see SSL/TLS offload (p. 392) .	October 12, 2017
New guide	This release introduces AWS CloudHSM	August 14, 2017