Table of Contents

What Is AWS CloudHSM? ............................................................................................................................ 1
Features ................................................................................................................................................ 1
Compliance .......................................................................................................................................... 1
Pricing .................................................................................................................................................. 2
Use Cases ........................................................................................................................................... 2
  Offload the SSL/TLS Processing for Web Servers ................................................................. 2
  Protect the Private Keys for an Issuing Certificate Authority (CA) .................................. 2
  Enable Transparent Data Encryption (TDE) for Oracle Databases ..................................... 3
Concepts .................................................................................................................................................. 3
  Clusters ........................................................................................................................................... 3
  Backups .......................................................................................................................................... 7
  Client Tools and Libraries ......................................................................................................... 9
  HSM Users .................................................................................................................................... 11
Getting Started: Create A Cluster ........................................................................................................ 14
Set Up the Prerequisites ...................................................................................................................... 14
  Create an IAM User .................................................................................................................. 14
  Create a Virtual Private Cloud (VPC) ................................................................................... 16
  Create Private Subnets .......................................................................................................... 16
Create a Cluster and HSM ................................................................................................................... 17
  Create a Cluster ....................................................................................................................... 17
  Create an HSM .......................................................................................................................... 18
(Optional) Verify HSM Identity ........................................................................................................... 19
  Overview ..................................................................................................................................... 19
  Get Certificates from the HSM .............................................................................................. 21
  Verify Certificate Chains ........................................................................................................ 22
  Extract and Compare Public Keys .......................................................................................... 23
  AWS CloudHSM Root Certificate ......................................................................................... 23
Initialize the Cluster ........................................................................................................................... 25
  Get the Cluster's Certificate Signing Request (CSR) .......................................................... 25
  Sign the CSR ............................................................................................................................. 26
  Initialize the Cluster ................................................................................................................ 27
Launch a Client Instance ...................................................................................................................... 28
  Launching a Client Instance ................................................................................................. 29
  Connecting to Your Client Instance .................................................................................... 30
Install and Configure the Client ...................................................................................................... 30
  Install the AWS CloudHSM Client and Command Line Tools .......................................... 30
  Edit the Client Configuration ................................................................................................. 31
Activate the Cluster .......................................................................................................................... 31
Managing Clusters ............................................................................................................................. 33
  Adding or Removing HSMs ..................................................................................................... 33
  Adding an HSM .......................................................................................................................... 33
  Removing an HSM ..................................................................................................................... 35
Deleting a Cluster ............................................................................................................................... 36
Creating a Cluster From a Backup ................................................................................................. 37
Tagging Resources .............................................................................................................................. 38
  Adding or Updating Tags ....................................................................................................... 38
  Listing Tags .............................................................................................................................. 40
  Removing Tags ........................................................................................................................ 40
Command Line Tools .......................................................................................................................... 42
  cloudhsm_mgmt_util .............................................................................................................. 42
    Getting Started with cloudhsm_mgmt_util .................................................................... 43
    cloudhsm_mgmt_util Reference ....................................................................................... 45
  key_mgmt_util ..................................................................................................................... 74
    Getting Started with key_mgmt_util ................................................................................. 74

AWS CloudHSM User Guide
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.

Features

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.

To learn more about what you can do with AWS CloudHSM, see Use Cases (p. 2).

To learn more about the fundamental concepts of AWS CloudHSM, see Concepts (p. 3).

When you are ready to get started with AWS CloudHSM, see Getting Started: Create A Cluster (p. 14).

With AWS CloudHSM, you get your own HSM hosted in the AWS Cloud, giving you the control that comes with operating an HSM. If you want a managed service for creating and controlling your data's encryption keys, but you don't want or need to operate your own HSM, consider AWS Key Management Service.

Compliance

AWS and AWS Marketplace partners offer many solutions for protecting data in AWS. But some applications and data are subject to strict contractual or regulatory requirements for managing and using cryptographic keys. Using an HSM from AWS CloudHSM can help you meet corporate, contractual, and regulatory compliance requirements for data security in the AWS Cloud.
FIPS 140-2

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. For more information, see FIPS Compliance on the AWS website.

PCI DSS

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. For more information, see PCI DSS Compliance on the AWS website.

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.

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. 2)
- Protect the Private Keys for an Issuing Certificate Authority (CA) (p. 2)
- Enable Transparent Data Encryption (TDE) for Oracle Databases (p. 3)

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 a 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 computation to the HSMs in your AWS CloudHSM cluster. This is sometimes known as SSL acceleration. This 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 Improve Web Server Security with SSL/TLS Offload (p. 187).

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 keys that sign the certificates issued by your CA. You can store these private keys in an HSM 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. 201).

AWS CloudHSM Concepts

The following concepts will help you get started with AWS CloudHSM.

Topics
- AWS CloudHSM Clusters (p. 3)
- AWS CloudHSM Cluster Backups (p. 7)
- AWS CloudHSM Client Tools and Software Libraries (p. 9)
- HSM Users (p. 11)

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 32 HSMs (the default limit (p. 211) 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: Create A Cluster (p. 14).

For more information about clusters, see the following topics.

Topics
- Cluster Architecture (p. 3)
- Cluster Synchronization (p. 5)
- Cluster High Availability and Load Balancing (p. 6)

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 Private Subnets (p. 16) in the prerequisites section.

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. 134). For more information, see Keep HSM Users In Sync (p. 209).
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. 11). 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. 9) 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 your cluster. You can't instruct AWS CloudHSM to make backups anytime that you want, but you can take certain actions that result in AWS CloudHSM making a backup. For more information, see the following topics.

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

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

Topics

- Overview of Backups (p. 7)
- Security of Backups (p. 8)
- Durability of Backups (p. 9)
- Frequency of Backups (p. 9)

Overview of Backups

Each backup contains encrypted copies of the following data:

- All users (COs, CUs, and AUs) (p. 11) on the HSM.
- All key material and certificates on the HSM.
- The HSM's configuration and policies.

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

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

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

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

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

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

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

**Durability of Backups**

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

**Frequency of Backups**

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

- Initialize the cluster (p. 25).
- Add an HSM to an initialized cluster (p. 33).
- Remove an HSM from a cluster (p. 35).

**AWS CloudHSM Client Tools and Software Libraries**

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

**Topics**

- AWS CloudHSM Client (p. 10)
- AWS CloudHSM Command Line Tools (p. 11)
- AWS CloudHSM Software Libraries (p. 11)
AWS CloudHSM Client

The AWS CloudHSM client is a daemon that you install and run on your application hosts. The client establishes and maintains a secure, end-to-end encrypted connection with the HSMs in your AWS CloudHSM cluster. The client provides the fundamental connection between your application hosts and your HSMs. Most of the other AWS CloudHSM client software components rely on the client to communicate with your HSMs. To get started with the AWS CloudHSM client, see Install and Configure the Client (p. 30).

AWS CloudHSM Client End-to-End Encryption

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

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

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

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

The AWS CloudHSM client software includes two command line tools. You use the command line tools to manage the users and keys on the HSMs. For example, you can create HSM users, change user passwords, create keys, and more. For information about these tools, see Command Line Tools (p. 42).

AWS CloudHSM Software Libraries

You can use the AWS CloudHSM software libraries to integrate your applications with the HSMs in your cluster and use them for cryptoprocessing. For more information about installing and using the different libraries, see Using the Software Libraries (p. 155).

HSM Users

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

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

Topics
- Precrypto Officer (PRECO) (p. 11)
- Crypto Officer (CO, PCO) (p. 12)
- Crypto User (CU) (p. 12)
- Appliance User (AU) (p. 12)
- HSM User Permissions Table (p. 12)

Precrypto Officer (PRECO)

The precrypto officer (PRECO) is a temporary user that exists only on the first HSM in an AWS CloudHSM cluster. The first HSM in a new cluster contains a PRECO user with a default user name and password.
To activate a cluster (p. 31), you log in to the HSM and change the PRECO user's password. When you change the password, the PRECO user becomes a crypto officer (PCO). The PRECO user can only change its own password and perform read-only operations on the HSM.

**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. For more information, see the HSM User Permissions Table (p. 12).

When you activate a new cluster (p. 31), the first user on an HSM changes from a Precrypto Officer (p. 11) (PRECO) to a primary Crypto Officer (PCO). The PCO is the first CO created on the HSM. However, the PCO has the same permissions on the HSM as any other 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. 12).

**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. 12).

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

**HSM User Permissions Table**

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Crypto Officer (CO)</th>
<th>Crypto User (CU)</th>
<th>Appliance User (AU)</th>
<th>Unauthenticated user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get basic cluster info¹</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Zeroize an HSM²</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Change own password</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Change any user's password</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Add, remove users</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Get sync status³</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Crypto Officer (CO)</td>
<td>Crypto User (CU)</td>
<td>Appliance User (AU)</td>
<td>Unauthenticated user</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>--------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Extract, insert masked objects⁴</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Create, share, delete keys</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Encrypt, decrypt</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sign, verify</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Generate digests and HMACs</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

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

²When an HSM is zeroized, all keys, certificates, and other data on the HSM is destroyed. You can use your cluster's security group to prevent an unauthenticated user from zeroizing your HSM. For more information, see Create a Cluster (p. 17).

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

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

This section gives you a hands-on introduction to AWS CloudHSM. Complete the following steps to set up and configure an AWS CloudHSM cluster with one HSM.

To get started with AWS CloudHSM

1. Follow the steps in Set Up the Prerequisites (p. 14) to prepare your environment.
2. Follow the steps in Create a Cluster and HSM (p. 16).
3. (Optional) Follow the steps in (Optional) Verify HSM Identity (p. 19) to verify the identity and authenticity of the cluster’s HSM.
4. Follow the steps in Initialize the Cluster (p. 25).
5. (First time only) Follow the steps in Launch a Client Instance (p. 28).
6. (First time only) Follow the steps in Install and Configure the Client (p. 30).
7. Follow the steps in Activate the Cluster (p. 31).

After you complete these steps, you're ready to manage the HSM's users and use the cluster's HSM. For more information, see Managing HSM Users (p. 134) and Using the Software Libraries (p. 155).

For information about managing your cluster, see Managing Clusters (p. 33).

Getting Started with AWS CloudHSM: Set up the Necessary Prerequisites

Complete the steps in the following topics to set up your AWS environment for use with AWS CloudHSM.

Topics
- Create an IAM User (p. 14)
- Create a Virtual Private Cloud (VPC) (p. 16)
- Create Private Subnets (p. 16)

Create an IAM User

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. If you don't have one already, complete the following steps to create an IAM user for yourself and give that user administrative permissions.

To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in to the AWS Management Console as the AWS account root user.
2. In the navigation pane of the console, choose Users, and then choose Add user.
3. For User name, type Administrator.
4. Select the check box next to AWS Management Console access, select Custom password, and then type the new user's password in the text box. You can optionally select Require password reset to force the user to select a new password the next time the user signs in.

5. Choose Next: Permissions.

6. On the Set permissions for user page, choose Add user to group.

7. Choose Create group.

8. In the Create group dialog box, type Administrators.

9. For Filter, choose Job function.

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

11. 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.

12. 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 to restrict users’ permissions to specific AWS resources, go to Access Management and Example Policies.

To sign in to the AWS Management Console with your IAM user, you need your AWS account ID or alias. To get these items, see Your AWS Account ID and Its Alias in the IAM User Guide.

Restrict User Permissions to What's Necessary for AWS CloudHSM

The steps in the preceding section explain how to create an IAM user with administrative permissions in your AWS account. To restrict the user’s permissions to only those necessary for using AWS CloudHSM, remove the user from the AdministratorAccess group. Then add the user to a group that you created for AWS CloudHSM administrators. The following example policy contains the minimum set of permissions that AWS CloudHSM administrators need to manage AWS CloudHSM resources.

These permissions include full access to the AWS CloudHSM API and also some additional permissions in Amazon Elastic Compute Cloud (Amazon EC2). When you perform certain actions with the AWS CloudHSM console or API, AWS CloudHSM takes additional actions on your behalf to manage certain Amazon EC2 resources. For example, this can occur when you create and delete clusters and HSMs.

```json
{
  "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"
    ]
  }
}
```
Create a Virtual Private Cloud (VPC)

If you don’t already have an Amazon Virtual Private Cloud (VPC), create one now.

To create a VPC

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

Create Private Subnets

Create a private subnet (a subnet with no internet gateway attached) for each Availability Zone where you want to create an HSM. Creating a private subnet in each Availability Zone provides the most robust configuration for high availability.

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.

After you create a VPC and private subnets, proceed to Create a Cluster and HSM (p. 16).

Getting Started with AWS CloudHSM: Create a Cluster and HSM

Complete the steps in the following topics to create an AWS CloudHSM cluster and the cluster’s first HSM.
Create a Cluster

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. 28) and add it to this security group.

Warning
The cluster's 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, but 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 this, ensure that only trusted administrators can access the instances in the cluster's security group or modify the security group.

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)

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 when you set up the prerequisites (p. 14).
   b. For AZ(s), next to each Availability Zone, choose the private subnet that you created when you set up the prerequisites (p. 14).
5. Choose Next: Review.
6. Review your cluster configuration, then choose Create cluster.

To create a cluster (AWS CLI)

- At a command prompt, issue the create-cluster command. Specify the HSM instance type and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you created when you set up the prerequisites (p. 14). Specify only one subnet per Availability Zone.

```bash
$ aws cloudhsmv2 create-cluster --hsm-type hsm1.medium --subnet-ids subnet-ID1 subnet-ID2 subnet-IDN
```

```json
{
   "Cluster": {
      "BackupPolicy": "DEFAULT",
      "VpcId": "vpc-50ae0636",
      "SubnetMapping": {
         "us-west-2b": "subnet-49albc00",
         "us-west-2c": "subnet-6f950334",
         "us-west-2a": "subnet-fd54af9b"
      }
   }
}
```
To create a cluster (AWS CloudHSM API)

- Send a CreateCluster request. Specify the HSM instance type and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you set up the prerequisites (p. 14). Specify only one subnet per Availability Zone.

Create an HSM

Before you can create an HSM in the 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 issue the describe-clusters command, or send a DescribeClusters request in the AWS CloudHSM API.

You can create an HSM from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To create an HSM (console)

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

To create an HSM (AWS CLI)

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

```bash
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-zone <Availability Zone>
{
  "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. 19), or proceed directly to Initialize the Cluster (p. 25).

(Optional) Verify the Identity and Authenticity of Your Cluster's HSM

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. 19)
- Get Certificates from the HSM (p. 21)
- Verify Certificate Chains (p. 22)
- Extract and Compare Public Keys (p. 23)
- AWS CloudHSM Root Certificate (p. 23)

**Overview**

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

1. **Get the certificates and CSR (p. 21)** – In this step, you get three certificates and a CSR from the HSM. You also get two root certificates, one from AWS CloudHSM and one from the HSM hardware manufacturer.

2. **Verify the certificate chains (p. 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.

3. **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.
Overview

AWS Root Certificate

This is AWS CloudHSM's root certificate. You can view and download this certificate at https://docs.aws.amazon.com/cloudhsm/latest/userguide/root-certificate.html (p. 23).

Manufacturer Root Certificate

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

AWS Hardware Certificate

AWS CloudHSM created this certificate when it claimed the HSM hardware. Your cluster's HSM is a virtual device that runs on specialized, FIPS-validated hardware. 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 hardware created this certificate was it created the cluster's virtual HSM device. This certificate asserts that the HSM hardware created the virtual HSM.

Cluster CSR

The virtual HSM device created this CSR. To initialize and claim your cluster, you sign this CSR.

Get Certificates from the HSM

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

To get the CSR and HSM certificates (console)

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

To get the CSR and HSM certificates (AWS CLI)

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

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

b. Issue the following command to extract the HSM certificate. Replace <cluster ID> with the ID of the cluster that you created previously.
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.

To get the AWS CloudHSM and manufacturer root certificates

1. Go to https://docs.aws.amazon.com/cloudhsm/latest/userguide/root-certificate.html (p. 23), and then choose AWS_CloudHSM_Root-G1.zip. After you download the file, extract (unzip) its contents.
2. Go to https://www.cavium.com/LS/TAmanuCert/, and then choose Download Certificate. You might need to right-click the Download Certificate link and then choose Save Link As... to save the certificate file.

Verify Certificate Chains

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 verify the HSM certificate with the AWS CloudHSM root certificate

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

```bash
$ cat <cluster ID>_AwsHardwareCertificate.crt 
AWS_CloudHSM_Root-G1.crt 
```
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 extract and compare the public keys

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

```bash
$ 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.

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

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

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

After you verify the identity and authenticity of the HSM, proceed to Initialize the Cluster (p. 25).

AWS CloudHSM Root Certificate

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

Data:

Version: 3 (0x2)
Serial Number: 17952736724058457791 (0xf924e6ecf9ea64f)
Signature Algorithm: sha256WithRSAEncryption

Issuer: C=US,
ST=Virginia,
L=Herndon,
O=Amazon Web Services INC.,
OU=CloudHSM,
CN=AWS CloudHSM Root G1

Validity
Not Before: Apr 28 08:37:46 2017 GMT
Not After : Apr 26 08:37:46 2027 GMT

Subject: C=US,
ST=Virginia,
L=Herndon,
O=Amazon Web Services INC.,
OU=CloudHSM,
CN=AWS CloudHSM Root G1

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)
Modulus:
00:c8:e3:f6:2a:ea:01:ff:1e:66:73:00:1e:57:dc:3e:
52:61
Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Subject Key Identifier:

X509v3 Authority Key Identifier:

X509v3 Basic Constraints:
CA:TRUE

Signature Algorithm: sha256WithRSAEncryption
6c:1a:39:00:4d:98:1e:21:26:4a:02:ff:55:3e:6c:47:9d:1c:
Getting Started with AWS CloudHSM: 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. 19). The process is optional, but 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**
- Get the Cluster’s Certificate Signing Request (CSR) (p. 25)
- Sign the CSR (p. 26)
- Initialize the Cluster (p. 27)

**Get the Cluster's Certificate Signing Request (CSR)**

Before you can initialize the cluster, you get (and then later 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. 19), you already have the CSR and you can proceed to sign the CSR (p. 26). Otherwise, get the CSR now by using the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

**To get the CSR (console)**

2. Choose **Initialize** next to the cluster that you created previously (p. 16).
3. When the CSR is ready, you see a link to download it.
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. 16).

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

**To get the CSR (AWS CloudHSM API)**

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

**Sign the CSR**

To sign the cluster’s CSR, you typically use a private certificate authority (CA). Your CA signs the CSR, which creates a signed certificate. Then you provide the signed certificate and your CA’s issuing certificate to initialize the cluster.

If you don’t have a CA, you can use the following OpenSSL commands to create a self-signed certificate and use it as your issuing certificate.

**Important**

The following example is a proof-of-concept demonstration only. For production systems, use a more secure method (such as a CA) to sign the CSR. The certificate that you use is not designed to be rotated. It demonstrates that you extended trust to this cluster by signing the cluster certificate. We recommend that you create and use a certificate that is valid for ten years.

**To sign the cluster’s CSR (OpenSSL)**

1. Use the following command to create a private key.

```bash
$ 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:
```

2. Use the following command to create a self-signed issuing certificate with the private key that you created in the previous step. The certificate is valid for 10 years (3652 days). Read the on-screen instructions and follow the prompts to provide identifying information for the issuing certificate.

```bash
$ 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 [:]

When completed, this command creates a file named `customerCA.crt`. Use this file as your issuing certificate (trust anchor) when you initialize the cluster.

3. Sign the cluster's CSR with your issuer. Replace `<cluster ID>` with the ID of the cluster that you created previously. This signature is also valid for 10 years.

```
$ openssl x509 -req -days 3652 -in <cluster ID>_ClusterCsr.csr -CA customerCA.crt -CAkey customerCA.key -CAcreateserial -out <cluster ID>_CustomerHsmCertificate.crt
```

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

To initialize your cluster, you provide the signed HSM certificate and your issuing certificate (trust anchor). You can initialize your cluster with the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

### To initialize a cluster (console)

2. Choose Initialize next to the cluster that you created previously.
3. On the Download certificate signing request page, choose Next. If Next is not available, first choose one of the CSR or certificate links. Then choose Next.
4. On the Sign certificate signing request (CSR) page, choose Next.
5. On the Upload the certificates page, do the following:
   a. Next to Cluster certificate, choose Upload file. Then 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 CA's issuing certificate. If you completed the steps in the previous section, select the file named `customerCA.crt`.

   If you used a CA to issue the cluster certificate, provide a certificate chain that begins with the certificate that issued the cluster certificate and ends with the CA's root certificate. The certificate chain must be in PEM format and can contain a maximum of 5000 characters.
   c. Choose Upload and initialize.
To initialize a cluster (AWS CLI)

- At a command prompt, issue 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 CA's issuing certificate. If you completed the steps in the previous section, it's saved in a file named `customerCA.crt`.

If you used a CA to issue the cluster certificate, provide a certificate chain that begins with the certificate that issued the cluster certificate and ends with the CA's root certificate. The certificate chain must be in PEM format and can contain a maximum of 5000 characters.

```bash
$ 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 CA's issuing certificate.

After you initialize the cluster, proceed to Launch a Client Instance (p. 28).

Getting Started with AWS CloudHSM: Launch a Client Instance and Add the Cluster Security Group

Complete the following steps to create an Amazon EC2 instance, known as a client instance. Later you install and configure the AWS CloudHSM client software, which you use to access your cluster from the client instance.

**Important**

Make sure that you assign the cluster's security group, named `cloudhsm-<cluster ID>-sg`, to your client instance as described in the following procedure. If you don't, you won't be able to connect to your cluster from the client instance.

Ensure that only trusted administrators can access this client instance and all instances in the cluster's security group. For more information, see the warning in Create a Cluster (p. 17).

**Topics**

- Launching a Client Instance (p. 29)
- Connecting to Your Client Instance (p. 30)
Launching a Client Instance

Complete the following steps to launch a client instance.

**To launch a client instance**

1. Open the Amazon EC2 console at [https://console.aws.amazon.com/ec2/](https://console.aws.amazon.com/ec2/).
2. On the navigation bar, choose the AWS Region where you created your cluster (p. 16).
3. Choose **Launch Instance**.
4. In the row for the newest Amazon Linux AMI, choose **Select**.
5. Select an instance type, and then choose **Next: Configure Instance Details**.
6. For **Step 3: Configure Instance Details**, do the following:
   a. For **Network**, choose the VPC that you created previously (p. 16).
   b. For **Subnet**, choose the public subnet that you created previously (p. 16).
   c. For **Auto-assign Public IP**, choose **Enable**.
   d. Change the remaining instance details as preferred. Then choose **Next: Add Storage**.
7. Change the storage settings as preferred. Then choose **Next: Add Tags**.
8. Add tags as preferred. Then choose **Next: Configure Security Group**.
9. For **Step 6: Configure Security Group**, do the following:
   a. For **Assign a security group**, choose **Select an existing security group**.
   b. Select the check box next to the security group named `cloudhsm-<cluster ID>-sg`. AWS CloudHSM created this security group on your behalf when you created the cluster (p. 17). You must choose this security group to allow the client instance to connect to the HSMs in the cluster.
   c. Select the check box next to a security group that allows inbound SSH traffic from your network. That is, the security group must allow inbound TCP traffic on port 22. Otherwise, you cannot connect to your client instance with SSH. If you don't have a security group like this, you must create one and then assign it to your client instance later. For more information about creating a security group, see Create a Security Group in the Amazon VPC Getting Started Guide.

Then choose **Review and Launch**.
10. Review your instance details, and then choose **Launch**.
11. Choose whether to launch your instance with an existing key pair or create a new 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 an identifiable key pair name such as CloudHSM client instance key pair.
  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.
Then choose Launch Instances.

Connecting to Your Client Instance

When the client instance is running, you can connect to it using SSH. For more information, see one of the following topics.

- If your computer’s operating system is Linux or Apple macOS, see Connecting to Your Linux Instance Using SSH in the Amazon EC2 User Guide for Linux Instances.
- If your computer’s operating system is Microsoft Windows, see Connecting to Your Linux Instance from Windows Using PuTTY in the Amazon EC2 User Guide for Linux Instances.

After you connect to your client instance, proceed to Install and Configure the Client (p. 30).

Getting Started with AWS CloudHSM: Install and Configure the Client

To interact with the HSM in your AWS CloudHSM cluster, you need the AWS CloudHSM client software. We recommend that you install it on the client instance that you created previously (p. 28).

Topics

- Install the AWS CloudHSM Client and Command Line Tools (p. 30)
- Edit the Client Configuration (p. 31)

Install the AWS CloudHSM Client and Command Line Tools

Complete the steps in the following procedure to install the AWS CloudHSM client and command line tools.

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

1. If you haven’t already done so, connect to the client instance that you created previously (p. 28). You can do this using SSH (Linux or macOS) or from Windows using PuTTY.
2. Use the following commands to download and then install the client and command line tools.

   Amazon Linux

```bash
$ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-latest.x86_64.rpm
$ sudo yum install -y ./cloudhsm-client-latest.x86_64.rpm
```
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/cloudhsm/etc/customerCA.crt. You need root permissions on the client instance to copy your certificate to this location.

2. Use the following 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. If you don't know the HSM's IP address, view your cluster in the AWS CloudHSM console, or use the AWS CLI to issue the describe-clusters command. In the command's output, the HSM's IP address is the value of the EniIp field. If you have more than one HSM, choose the IP address for any of the HSMs; it doesn't matter which one.

```
$ sudo /opt/cloudhsm/bin/configure -a <IP address>
Inserting '<IP address>' into '/opt/cloudhsm/etc/cloudhsm_client.cfg'
Inserting '<IP address>' into '/opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg'
```

Getting Started with AWS CloudHSM: Activate the Cluster

When you activate an AWS CloudHSM cluster, the cluster's state changes from initialized to active. You can then manage the HSM's users (p. 134) and use the HSM (p. 155).

To activate the cluster, you log in to the HSM with the credentials of the HSM's precrypto officer (PRECO) user (p. 11). Then you change the user's default password, which makes the user a crypto officer (CO).

To activate a cluster

1. If you haven't already done so, connect to the client instance that you created previously (p. 28). You can do this using SSH (Linux or macOS) or from Windows using PuTTY.

2. Use the following command to start the AWS CloudHSM cloudhsm_mgmt_util (p. 42) command line tool.

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

3. Use the enable_e2e command to enable end-to-end encryption.
4. (Optional) Use the `listUsers` command to display the existing users.

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

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

5. Use the `loginHSM` command to log in to the HSM as the precrypto officer (PRECO) (p. 11) user.

```bash
aws-cloudhsm> loginHSM PRECO admin password
loginHSM success on server 0(server1)
```

6. Use the `changePswd` command to change the precrypto officer (PRECO) user's password.

```bash
aws-cloudhsm> changePswd PRECO admin <NewPassword>
****************************************************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)? y
Changing password for admin(PRECO) on 1 nodes
```

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

7. (Optional) Use the `listUsers` command to verify that the user's type changed to (primary) crypto officer (PCO) (p. 12).

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

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

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

```bash
aws-cloudhsm> quit
```
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: Create A Cluster (p. 14).

Topics
- Adding or Removing HSMs in an AWS CloudHSM Cluster (p. 33)
- Deleting an AWS CloudHSM Cluster (p. 36)
- Creating an AWS CloudHSM Cluster from a Previous Backup (p. 37)
- Tagging AWS CloudHSM Resources (p. 38)

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. 33)
- Removing an HSM (p. 35)

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. 7).

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

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

5. The client establishes a connection to the new HSM.
To add an HSM (console)

2. Choose a cluster for the HSM that you are adding.
3. On the HSMs tab, choose Create HSM.
4. Choose an Availability Zone (AZ) for the HSM that you are creating. Then choose Create.

To add an HSM (AWS CLI)

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

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-zone <Availability Zone>
{
    "Hsm": {
        "State": "CREATE_IN_PROGRESS",
        "ClusterId": "cluster-5a73d5gqrdh",
        "HsmId": "hsm-lgavqitna2a",
        "SubnetId": "subnet-0e358c43",
        "AvailabilityZone": "us-east-2c",
        "EniId": "eni-bab18892",
        "EniIp": "10.0.3.10"
    }
}
```

To add an HSM (AWS CloudHSM API)

- Send a CreateHsm request, specifying the cluster ID and an Availability Zone for the HSM that you are creating.

Removing an HSM

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

To remove an HSM (console)

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

To remove an HSM (AWS CLI)

- At a command prompt, issue the delete-hsm command. Pass the ID of the cluster that contains the HSM that you are deleting and one of the following HSM identifiers:
  - The HSM ID (--hsm-id)
  - The HSM IP address (--eni-ip)
  - The HSM's elastic network interface ID (--eni-id)
If you don't know the values for these identifiers, issue the `describe-clusters` command.

```bash
$ aws cloudhsmv2 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. 35).

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

### To delete a cluster (console)

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

### To delete a cluster (AWS CLI)

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

```bash
$ aws cloudhsmv2 delete-cluster --cluster-id <cluster ID>
{
    "Cluster": {
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "SourceBackupId": "backup-rtq2dw12gq6",
        "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 an AWS CloudHSM Cluster from a Previous Backup

To restore an AWS CloudHSM cluster from a previous backup, you create a new cluster, specifying the backup to restore. After you create the cluster, you don’t need to initialize or activate it. You can just add an HSM to the cluster; this HSM contains the same users, key material, certificates, configuration, and policies that were in the backup that you restored. For more information about backups, see Backups (p. 7).

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

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

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

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

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

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

```json
{
  "Cluster": {
    "HsmType": "hsm1.medium",
    "VpcId": "vpc-641d3c0d",
    "State": "CREATE_IN_PROGRESS",
    "SourceBackupId": "backup-rtq2dwi2gq6",
    "BackupPolicy": "DEFAULT",
    "SecurityGroup": "sg-640fab0c",
    "CreateTimestamp": 1504907311.112,
    "SubnetMapping": {
      "us-east-2c": "subnet-0e358c43",
      "us-east-2a": "subnet-f1d6e798",
      "us-east-2b": "subnet-40ed9d3b"
    }
  }
}
```
To create a cluster from a previous backup (AWS CloudHSM API)

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

To create an HSM that contains the same users, key material, certificates, configuration, and policies that were in the backup that you restored, add an HSM (p. 33) to the cluster.

Tagging AWS CloudHSM Resources

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

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

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

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

Topics
- Adding or Updating Tags (p. 38)
- Listing Tags (p. 40)
- Removing Tags (p. 40)

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)

2. Choose the cluster that you are tagging.
3. Choose Tags.
4. To add a tag, do the following:
   a. Choose Add Tag.
b. For **Tag Key**, type a key for the tag.

c. (Optional) For **Tag Value**, type a value for the tag.

d. Choose the action for adding a tag, as shown in the following image.

5. To update a tag, do the following:

a. Choose the tag value to update.

   **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. Then choose the action for updating a tag, as shown in the following image.

**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.

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

2. To update tags, use the same command but specify an existing tag key. When you specify a new tag value for an existing tag, the tag is overwritten with the new value.
To add or update tags (AWS CloudHSM API)

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

Listing Tags

You can list tags for a cluster from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To list tags (console)

2. Choose the cluster whose tags you are listing.
3. Choose Tags.

To list tags (AWS CLI)

- At a command prompt, issue the list-tags command, specifying the ID of the cluster whose tags you are listing. If you don't know the cluster ID, issue the describe-clusters command.

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

To list tags (AWS CloudHSM API)

- Send a ListTags request, specifying the ID of the cluster whose tags you are listing.

Removing Tags

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

To remove tags (console)

2. Choose the cluster whose tags you are removing.
3. Choose Tags.
4. Next to the tag that you are removing, choose the action for deleting a tag, as shown in the following image.
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.
AWS CloudHSM Command Line Tools

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

Topics
- cloudhsm_mgmt_util (p. 42)
- key_mgmt_util (p. 74)

Manage Clusters and HSMs

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

Manage Users

This tool creates and deletes HSM users, including implementing quorum authentication of user management tasks:
- cloudhsm_mgmt_util (p. 42). This tool is included in the AWS CloudHSM client software.

Manage Keys

This tool creates, deletes, imports, and exports symmetric keys and asymmetric key pairs:
- key_mgmt_util (p. 74). 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. 31) updates your CloudHSM client configuration files.
- pkpspeed (p. 209) measures the performance of your HSM hardware independent of software libraries.

cloudhsm_mgmt_util

The cloudhsm_mgmt_util command line tool helps Crypto Officers (PCOs and COs) 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, get and set key attributes. These commands complement the key management commands in the primary key management tool, key_mgmt_util (p. 74).

For a quick start, see Getting Started with cloudhsm_mgmt_util (p. 43). For detailed information about the cloudhsm_mgmt_util commands and examples of using the commands, see cloudhsm_mgmt_util Command Reference (p. 45).

Topics
• Getting Started with cloudhsm_mgmt_util (p. 43)
• cloudhsm_mgmt_util Command Reference (p. 45)

Getting Started with cloudhsm_mgmt_util

AWS CloudHSM includes two command line tools with the AWS CloudHSM client software that you installed previously (p. 30). One of these tools is cloudhsm_mgmt_util. You use cloudhsm_mgmt_util primarily to manage HSM users (p. 134).

To use cloudhsm_mgmt_util, first connect to your client instance (p. 30) and then install and configure the AWS CloudHSM client software (p. 30). Then see the following topics to get started.

Topics
• Setup cloudhsm_mgmt_util (p. 43)
• Basic Usage of cloudhsm_mgmt_util (p. 44)

Setup cloudhsm_mgmt_util

Complete the following setup before you use cloudhsm_mgmt_util. You need to do these steps the first time you use cloudhsm_mgmt_util and when you add or remove HSMs in your cluster.

Topics
• Start the AWS CloudHSM Client (p. 43)
• Update the cloudhsm_mgmt_util Configuration File (p. 43)

Start the AWS CloudHSM Client

Before you use cloudhsm_mgmt_util, start the AWS CloudHSM client. The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster. When you add or remove HSMs in your cluster, the cluster informs the client of these changes. The client keeps the current list of HSMs in its configuration file, and cloudhsm_mgmt_util can use this file to get an updated list of the cluster's HSM.

To start the AWS CloudHSM client

Use the following command to start the AWS CloudHSM client.

$ sudo start cloudhsm-client

Update the cloudhsm_mgmt_util Configuration File

After you start the AWS CloudHSM client as described in the previous section, you can update the cloudhsm_mgmt_util configuration file to include all the HSMs in your cluster. If you don't do this, you might run into problems because HSM users are not in sync across your cluster's HSMs (p. 209).
Use the following command to update the cloudhsm_mgmt_util configuration file.

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

**Basic Usage of cloudhsm_mgmt_util**

See the following topics for the basic usage of the cloudhsm_mgmt_util tool.

**Note**
The cloudhsm_mgmt_util tool doesn’t support auto-completing commands with the **Tab** key. Don’t use the **Tab** key with cloudhsm_mgmt_util, because that can make the tool unresponsive.

**Topics**
- Start cloudhsm_mgmt_util (p. 44)
- Enable End-to-End Encryption (p. 44)
- Log in to the HSMs (p. 44)
- Log Out from the HSMs (p. 45)
- Stop cloudhsm_mgmt_util (p. 45)

**Start cloudhsm_mgmt_util**

Use the following command to start cloudhsm_mgmt_util.

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

Connecting to the server(s), it may take time depending on the server(s) load, please wait...

Connecting to server '10.0.2.9': hostname '10.0.2.9', port 2225...
Connected to server '10.0.2.9': hostname '10.0.2.9', port 2225.

Connecting to server '10.0.3.11': hostname '10.0.3.11', port 2225...
Connected to server '10.0.3.11': hostname '10.0.3.11', port 2225.

Connecting to server '10.0.1.12': hostname '10.0.1.12', port 2225...
Connected to server '10.0.1.12': hostname '10.0.1.12', port 2225.

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

**Enable End-to-End Encryption**

Use the `enable_e2e` command to establish end-to-end encrypted communication between cloudhsm_mgmt_util and the HSMs in your cluster. You should enable end-to-end encryption each time you start cloudhsm_mgmt_util.

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

**Log in to the HSMs**

Use the `loginHSM` command to log in to the HSMs. The following command logs in as the default crypto officer (CO) (p. 11) named admin. You set this user’s password when you activated the cluster (p. 31).
The output shows that the command logged the *admin* user in to all of the HSMs in the cluster.

```
aws-cloudhsm> loginHSM CO admin <password>
loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
loginHSM success on server 2(10.0.1.12)
```

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

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

### Log Out from the HSMs

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

```
aws-cloudhsm> logoutHSM
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)
```

### Stop cloudhsm_mgmt_util

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 (PCOs and COs) 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. 74) command line tool.

For a quick start, see Getting Started with cloudhsm_mgmt_util (p. 43).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

To list all cloudhsm_mgmt_util commands, type:

```
aws-cloudhsm> help
```

To get the syntax for a cloudhsm_mgmt_util command, type:

```
aws-cloudhsm> help <command-name>
```

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

For example, to get a list of users on the HSMs, type `listUsers` or `listU`.
To end your cloudhsm_mgmt_util session, type:

```
aws-cloudhsm> quit
```

For help interpreting the key attributes, see the [Key Attribute Reference](#) (p. 130).

The following topics describe commands in cloudhsm_mgmt_util.

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>User Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>changePswd</code></td>
<td>Changes the passwords of users on the HSMs. Any user can change their own password. COs can change anyone's password.</td>
<td>CO</td>
</tr>
<tr>
<td><code>createUser</code></td>
<td>Creates users of all types on the HSMs.</td>
<td>CO</td>
</tr>
<tr>
<td><code>deleteUser</code></td>
<td>Deletes users of all types from the HSMs.</td>
<td>CO</td>
</tr>
<tr>
<td><code>findAllKeys</code></td>
<td>Gets the keys that a user owns or shares. Also gets a hash of the key ownership and sharing data for all keys on each HSM.</td>
<td>CO, AU</td>
</tr>
<tr>
<td><code>getAttribute</code></td>
<td>Gets an attribute value for an AWS CloudHSM key and writes it to a file or stdout.</td>
<td>CU</td>
</tr>
<tr>
<td><code>getHSMInfo</code></td>
<td>Gets information about the hardware on which an HSM is running.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td><code>getKeyInfo</code></td>
<td>Gets owners, shared users, and the quorum authentication status of a key.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td><code>info</code></td>
<td>Gets information about an HSM, including the IP address, host name, port, and current user.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td><code>listUsers</code></td>
<td>Gets the users in each of the HSMs, their user type and ID, and other attributes.</td>
<td>All. Login is not required.</td>
</tr>
<tr>
<td><code>setAttribute</code></td>
<td>Changes the values of the label, encrypt, decrypt, wrap, and unwrap attributes of an existing key.</td>
<td>CU</td>
</tr>
</tbody>
</table>
### Command Reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>User Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>shareKey</td>
<td>Shares an existing key with other users.</td>
<td>CU</td>
</tr>
</tbody>
</table>

#### changePswd

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

Any user can change their own password. Crypto officers (COs and PCOs) can also change the password of any other user. You do not need to enter the current password to make the change. However, you cannot change the password of a user who is logged into the AWS CloudHSM client or `key_mgmt_util`.

Before you run any `cloudhsm_mgmt_util` command, you must start `cloudhsm_mgmt_util` (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

#### User Type

The following types of users can run this command.

- All users.

#### Syntax

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

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

#### Examples

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

**Example: Change your password**

Any user on the HSMs can change use `changePswd` to change their own password.

The first command uses `info` (p. 65) to get the current user. The output shows that the current user, `bob`, is a crypto user (CU).

```
aws-cloudhsm> info server 0
Id   Name          Hostname        Port  State      Partition
LoginState
0    bob(CU)      10.0.3.10       2225  Connected  hsm-aaaabbbbcccc Logged
in as 'bob(CU)'

aws-cloudhsm> info server 1
Id   Name          Hostname        Port  State      Partition
LoginState
0    bob(CU)      10.0.3.10       2225  Connected  hsm-cccaaaaabbb Logged
in as 'bob(CU)'
```

To change his password, `bob` runs `changePswd` with a new password, `newPassword`.

When the command completes, the password change is effective.
aws-cloudhsm> createUser CU bob newPassword

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

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

Example: Change the password of another user

This example shows how to change password of a different user. Any crypto officer (CO, PCO) can change the password of any user on the HSMs without specifying the existing password.

The first command uses info (p. 65) to confirm that alice, a CO, is logged into the HSMs in the cluster.

aws-cloudhsm> info server 0
Id      Name             Hostname         Port   State           Partition
LoginState
0       10.0.3.10        10.0.3.10        2225   Connected       hsm-aaaabbbccc  Logged in as 'alice(CO)'
aws-cloudhsm> info server 1
Id      Name             Hostname         Port   State           Partition
LoginState
0       10.0.3.10        10.0.3.10        2225   Connected       hsm-ccccaaaabbb  Logged in as 'alice(CO)'

This command uses changePswd to change the password of officer1, another CO on the HSMs. In this case, the command resets the password to defaultPassword, the password that this fictitious enterprise uses as its default. Later, officer1 can reset their password to a more secure value.

aws-cloudhsm> changePswd CO officer1 defaultPassword

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

Do you want to continue(y/n)? y
Changing password for officer1(CO) on 2 nodes

Arguments

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

changePswd <user-type> <user-name> <password> [1FA | 2FA]

[user-type]

Specifies the current type of the user whose password you are changing. You cannot use changePswd to change the user type.
Valid values are CO, CU, AU, PCO, and PRECO.

To get the user type, use listUsers (p. 67). For detailed information about the user types on an HSM, see HSM Users (p. 11).

Required: Yes

<user-name>

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

Required: Yes

<password>

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

Required: Yes

1FA | 2FA

Enables or disables dual-factor authentication for the new user. Enter 1FA or 2FA.

This parameter is valid only when the cluster has been configured for dual-factor authentication.

Required: No

Default: 1FA. Dual factor authentication is not enabled.

Related Topics

- listUsers (p. 67)
- createUser (p. 49)
- deleteUser (p. 52)
- syncUser

createUser

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

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

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

User Type

The following types of users can run this command.

- Crypto officers (CO, PCO)
Syntax

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

User Type: Crypto officer (CO, PCO)

createUser <user-type> <user-name> <password> [1FA | 2FA]

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 to log in to the HSM as a crypto officer.

aws-cloudhsm> loginHSM CO admin 735782961
loginHSM success on server 0(10.0.0.1)
loginHSM success on server 1(10.0.0.2)
loginHSM success on server 1(10.0.0.3)

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

aws-cloudhsm> createUser CO alice 391019314
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'
Do you want to continue(y/n)?y
Creating User alice(CO) on 3 nodes

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

The final command uses the listUsers (p. 67) 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. 55).

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

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2FA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 \texttt{y} to respond to the caution message, the output shows that \texttt{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 \texttt{defaultPassword}. Later, bob or any CO can use the \texttt{changePswd (p. 47)} command to change his password.

```
aws-cloudhsm> createUser CU bob defaultPassword
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'
Do you want to continue(y/n)?y
Creating User bob(CU) on 3 nodes
```

Arguments

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

```
createUser <user-type> <user-name> <password> [ 1FA | 2FA ]
```
<user-type>
  Specifies the type of user. This parameter is required.
  For detailed information about the user types on an HSM, see HSM Users (p. 11).
  Valid values:
  • **CO**: Crypto officers can manage users, but they cannot manage keys.
  • **CU**: Crypto users can create and manage keys and use keys in cryptographic operations.
  • **AU**: Appliance users can clone and synchronize operations. One AU is created for you on each HSM that you install.
  PCO, PRECO, and preCO are also valid values, but they are rarely used. A PCO is functionally identical to a CO user. A PRECO user is a temporary type that is created automatically on each HSM. The PRECO is converted to a PCO when you assign a password during HSM activation (p. 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 cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.
  Required: Yes

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

1FA | 2FA
  Enables or disables dual-factor authentication for the new user. Enter 1FA or 2FA.
  This parameter is valid only when the cluster has been configured for dual-factor authentication.
  Required: No
  Default: 1FA: Dual factor authentication is not enabled.

Related Topics
• listUsers (p. 67)
• deleteUser (p. 52)
• syncUser
• changePswd (p. 47)

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

**Warning**
When you delete a crypto user (CU), all keys that the user owned are deleted, even if the keys were shared with other users. To make accidental or malicious deletion of users less likely, use quorum authentication (p. 147).

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

**User Type**

The following types of users can run this command.
- Crypto officers (CO, PCO)

**Syntax**

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

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

**Example**

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

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

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

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.2):
Number of users found:3

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Users on server 1(10.0.0.3):
Number of users found:3
```
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` (p. 67) command to verify that `alice` is deleted from all three of the HSMs on the cluster.

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

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2FA</td>
<td>admin</td>
<td>YES</td>
</tr>
<tr>
<td>1</td>
<td>PCO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CO</td>
<td>alice</td>
<td>NO</td>
</tr>
<tr>
<td>0</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

Arguments

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

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

`<user-type>`

Specifies the type of user. This parameter is required.
Warning
When you delete a crypto user (CU), all keys that the user owned are deleted, even if the keys were shared with other users. To make accidental or malicious deletion of users less likely, use quorum authentication (p. 147).

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

To get the user type, use listUsers (p. 67). For detailed information about the user types on an HSM, see HSM Users (p. 11).

Required: Yes

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

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

Required: Yes

Related Topics
- listUsers (p. 67)
- createUser (p. 49)
- syncUser
- changePswd (p. 47)

findAllKeys

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

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. 87) 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 listUsers (p. 67) to find all users, findKey (p. 87) in key_mgmt_util to find the keys that they can use and getKeyInfo (p. 113) in key_mgmt_util to find the owner and shared users of a particular key they own or share.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

User Type

The following types of 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.

```plaintext
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.

The output shows that user 4 can use 6 keys: 8, 9, 17, 262162, 19, and 31. The output uses an (s) to indicate that keys 8, 9, and 262162 are explicitly shared, although it does not indicate whether user 4 owns or shares them. The keys that are not marked with (s) include symmetric and private keys that the user 4 owns and 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),19,31
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),19,31
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),19,31
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 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),11,17(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
6,7,8(s),11,12,14,262159,262160,17(s),262162(s),19(s),20,21,262177,262179,262180,262181
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
6,7,8(s),11,12,14,262159,262160,17(s),262162(s),19(s),20,21,262177,262179,262180,262181
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. 67).

All users ID 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 on 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.
getAttribute

The `getAttribute` command in cloudhsm_mgmt_util gets one attribute value for a key from all HSMs in the cluster and writes it to stdout 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` 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` command. To change the attribute values of existing keys, use `setAttribute` in key_mgmt_util and `setAttribute` in cloudhsm_mgmt_util. For help interpreting the key attributes, see the **Key Attribute Reference**.

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

**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)

```
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` 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. 130).

```
aws-cloudhsm> 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
```

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.

```
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. 87) 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. 113) 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, type 256, which is the constant for the OBJ_ATTR_KEY_TYPE attribute.

To list the attributes and their constants, use listAttributes (p. 122). For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

Required: Yes

<filename>
Writes the output to the specified file. Type a file path.

If the specified file exists, getAttribute overwrites the file without warning.

Required: No

Default: Stdout

Related Topics
• AttributeSet (p. 110) in key_mgmt_util
• listAttributes (p. 66)
• setAttribute (p. 69) in cloudhsm_mgmt_util
• setAttribute (p. 124) in key_mgmt_util
• Key Attribute Reference (p. 130)

getHSMInfo

The getHSMInfo command in cloudhsm_mgmt_util gets information about the hardware on which each HSM runs, including the model, serial number, FIPS state, memory, temperature, and the version numbers of the hardware and firmware. The information also includes the server ID that cloudhsm_mgmt_util uses to refer to the HSM.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

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          :xxxxxxxxxxxxxxxxx
...
```

**Related Topics**

- `info` (p. 65)
- `loginHSM`

**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. 123). To find the keys for a particular user, use `findKey` (p. 87) -`u` in `key_mgmt_util`. Crypto officers can use `findAllKeys` (p. 55) 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. 71) in `cloudhsm_mgmt_util`.

Before you run any `cloudhsm_mgmt_util` command, you must `start cloudhsm_mgmt_util` (p. 44), `enable end-to-end encryption` (p. 44), and `log in` (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

**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. 104). 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. 87) 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. 104) to create a 256-bit AES key that is shared with user 4. It uses the m_value parameter to enable quorum authentication and set the quorum size to two users. The number of users must be large enough to provide the required approvals.

The output shows that the command created key 10.

```
Command: genSymKey -t 31 -s 32 -l aes256m2 -u 4 -m_value 2
         Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
         Symmetric Key Created. Key Handle: 10
         Cluster Error Status
            Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
            Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses getKeyInfo in cloudhsm_mgmt_util to get information about the users of key 10. The output shows that the key is owned by user 3 and shared with user 4. It also shows that a quorum of two users must approve every cryptographic operation that uses the key.

```
aws-cloudhsm> getKeyInfo 10
Key Info on server 0(10.0.0.1):
         Token/Flash Key,
         Owned by user 3
         also, shared to following 1 user(s):
               4
            2 Users need to approve to use/manage this key
Key Info on server 1(10.0.0.2):
         Token/Flash Key,
         Owned by user 3
         also, shared to following 1 user(s):
               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. 113) in key_mgmt_util
• findKey (p. 87) in key_mgmt_util
• findAllKeys (p. 55) in cloudhsm_mgmt_util
• listUsers (p. 67)
• shareKey (p. 71)

info
The info command in cloudhsm_mgmt_util gets information about each of the HSMs in the cluster, including the host name, port, IP address and the name and type of the user who is logged in to cloudhsm_mgmt_util on the HSM.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

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 names of the current user.

```
aws-cloudhsm> info server 0
Id   Name                   Hostname      Port  State  Partition
LoginState
```
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. 60)
- loginHSM

**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. 58) and `setAttribute` (p. 69) commands.

For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) to the HSM as a crypto user (CU).

**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**

```
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. 130). To represent all attributes, use 512.

Command: `listAttributes`

Description

```
The following are all of the possible attribute values for getAttributes.
```
Parameters

-h

Displays help for the command.

Required: Yes

Related Topics

- `getAttribute (p. 58)`
- `setAttribute (p. 69)`
- `Key Attribute Reference (p. 130)`

`listUsers`

The `listUsers` command in the `cloudhsm_mgmt_util` gets the users in each of the HSMs, along with their user type and other attributes. All types of users can run this command. You do not even need to be logged in to `cloudhsm_mgmt_util` to run this command.

Before you run any `cloudhsm_mgmt_util` command, you must `start cloudhsm_mgmt_util (p. 44)`, `enable end-to-end encryption (p. 44)`, and `log in (p. 44)` to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

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.
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`.

The output shows that one of the users, officer2, exists only on the first HSM. To add that user to other HSMs, use `syncUser`.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:6

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2FA</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CU</td>
<td>crypto_user1</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CU</td>
<td>crypto_user2</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CO</td>
<td>officer1</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CO</td>
<td>officer2</td>
<td>NO</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>User Id</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
<th>LoginFailureCnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2FA</td>
<td>PCO</td>
<td>admin</td>
<td>YES</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CU</td>
<td>crypto_user1</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CU</td>
<td>crypto_user2</td>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NO</td>
<td>CO</td>
<td>officer1</td>
<td>YES</td>
<td>0</td>
</tr>
</tbody>
</table>

The output includes the following user attributes:

- **User ID**: Identifies the user in `key_mgmt_util` and `cloudhsm_mgmt_util` (p. 42) commands.
- **User type (p. 11)**: 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. 142).
- **LoginFailureCnt**: Indicates whether the user has enabled multi-factor authentication.

Related Topics

- `listUsers` (p. 123) in `key_mgmt_util`
- `createUser` (p. 49)
- `deleteUser` (p. 52)
**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` 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 cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util, enable end-to-end encryption, and log in to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

**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)

```
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 to encrypt or decrypt data.

The first step is to create the wrapping key. This command uses `genSymKey` 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
```

Next, we want to confirm the current value of the decrypt attribute. To get the attribute ID of the decrypt attribute, use `listAttributes` in the key_mgmt_util to generate a 256-bit AES symmetric key. The output shows that the new key has key handle 14.

```
aws-cloudhsm> listAttributes
Following are the possible attribute values for getAttributes:

| OBJ_ATTR_CLASS | = 0 |
```
To get the current value of the decrypt attribute for key 14, the next command uses `getAttribute (p. 58)` in cloudhsm_mgmt_util.

The output shows that the value of the decrypt attribute is true (1) on both HSMs in the cluster.

```sql
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 will disable the decrypt functionality on the key.

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

```sql
aws-cloudhsm> setAttribute 14 261 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)? y
setAttribute success on server 0(10.0.0.1)
setAttribute success on server 1(10.0.0.2)
```

The final command repeats the `getAttribute` command. Again, it gets the decrypt attribute (attribute 261) of key 14.

This time, the output shows that the value of the decrypt attribute is false (0) on both HSMs in the cluster.

```sql
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):
```
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. 87) in key_mgmt_util. To find the users of a key, use getKeyInfo (p. 61).

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. 122). For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

Valid values:
- 3: OBJ_ATTR_LABEL.
- 260: OBJ_ATTR_ENCRYPT.
- 261: OBJ_ATTR_DECRYPT.
- 262: OBJ_ATTR_WRAP.
- 263: OBJ_ATTR_UNWRAP.

Required: Yes

Related Topics

- setAttribute (p. 124) in key_mgmt_util
- getAttribute (p. 58)
- listAttributes (p. 66)
- Key Attribute Reference (p. 130)

shareKey

The shareKey command in cloudhsm_mgmt_util shares and unshares keys that you own with other crypto users. Only the key owner can share and unshare a key. You can also share a key when you create it.

Users who share the key can use the key in cryptographic operations, but they cannot delete, export, share, or unshare the key, or change its attributes. When quorum authentication is enabled on a key, the quorum must approve any operations that share or unshare the key.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util (p. 44), enable end-to-end encryption (p. 44), and log in (p. 44) to the HSM. Be sure that the user type of the account that you use to log in can run the commands you plan to use.

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. 104) 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. 61) 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 example 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. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

To verify that the operation succeeded, the example repeats the first getKeyInfo command.
The output shows that key 262177 is now shared with user 4.

```
aws-cloudhsm> getKeyInfo 262177
Key Info on server 0(10.0.3.10):
   Token/Flash Key,
   Owned by user 3
   also, shared to following 1 user(s):
       4
Key Info on server 1(10.0.3.6):
   Token/Flash Key,
   Owned by user 3
   also, shared to following 1 user(s):
       4
```

**Example : Unshare a Key**

This example unshares a symmetric key, that is, it removes a crypto user from the list of shared users for the key.

This command uses `shareKey` to remove user 4 from the list of shared users for key 6. The final argument uses a value of 0 to indicate an unshare operation.

The output shows that the command succeeded on both HSMs. As a result, user 4 can no longer use key 6 in cryptographic operations.

```
aws-cloudhsm> shareKey 6 4 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)? y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

**Arguments**

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

```
shareKey <key handle> <user id> <(share/unshare key?) 1/0>
```

**<key-handle>**

Specifies the key handle of a key that you own. You can specify only one key in each command. To get the key handle of a key, use `findKey` (p. 87) in key_mgmt_util. To verify that you own a key, use `getKeyInfo` (p. 61).

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. 67).

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. 61)

key_mgmt_util

The key_mgmt_util command line tool helps Crypto Users (CU) manage keys in the HSMs. It includes multiple commands that generate, delete, import, and export keys, get and set attributes, find keys, and perform cryptographic operations.

For a quick start, see Getting Started with key_mgmt_util (p. 74). For detailed information about the commands, see key_mgmt_util Command Reference (p. 76). For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

To use key_mgmt_util, connect to your client instance (p. 30) and then install and configure the AWS CloudHSM client software (p. 30).

Topics

- Getting Started with key_mgmt_util (p. 74)
- key_mgmt_util Command Reference (p. 76)

Getting Started with key_mgmt_util

The key_mgmt_util tool in AWS CloudHSM helps crypto users (CUs) create and manage the keys (p. 137) in the HSMs in a cluster. Before you can use key_mgmt_util, you need to connect to your client instance (p. 30) and then install and configure the AWS CloudHSM client software (p. 30). Then you can set up and begin to use key_mgmt_util.

Topics

- Set Up key_mgmt_util (p. 74)
- Basic Usage of key_mgmt_util (p. 75)

If you encounter an error message or unexpected outcome for any command, see the Troubleshooting AWS CloudHSM (p. 208) topics for help. For details about the key_mgmt_util commands, see key_mgmt_util Command Reference (p. 76)

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.

$ sudo start cloudhsm-client

Start key_mgmt_util

After you start the AWS CloudHSM client, use the following command to start key_mgmt_util.

$ /opt/cloudhsm/bin/key_mgmt_util

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. 208).

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. 75)
- Log Out from the HSMs (p. 75)
- Stop key_mgmt_util (p. 76)

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. 11) 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.
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. 74). For help interpreting the key attributes, see the Key Attribute Reference (p. 130). For information about the cloudhsm_mgmt_util command line tool, which includes commands to manage the HSM and users in your cluster, see cloudhsm_mgmt_util (p. 42).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) to the HSM as a crypto user (CU).

To list all key_mgmt_util commands, type:

Command: **help**

To get help for a particular key_mgmt_util command, type:

Command: **<command-name> -h**

To end your key_mgmt_util session, type:

Command: **exit**

The following topics describe commands in key_mgmt_util.

**Note**

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aesWrapUnwrap (p. 77)</td>
<td>Encrypts and decrypts the contents of a key in a file on disk.</td>
</tr>
<tr>
<td>deleteKey (p. 80)</td>
<td>Deletes a key from the HSMs.</td>
</tr>
<tr>
<td>Error2String (p. 81)</td>
<td>Gets the error that corresponds to a key_mgmt_util hexadecimal error code.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>exSymKey (p. 82)</td>
<td>Exports a plaintext copy of a symmetric key from the HSMs to a file on disk.</td>
</tr>
<tr>
<td>findKey (p. 87)</td>
<td>Search for keys by key attribute value.</td>
</tr>
<tr>
<td>findSingleKey (p. 90)</td>
<td>Verifies that a key exists on all HSMs in the cluster.</td>
</tr>
<tr>
<td>genDSAKeyPair (p. 91)</td>
<td>Generates a Digital Signing Algorithm (DSA) key pair in your HSMs.</td>
</tr>
<tr>
<td>genECCKeyPair (p. 95)</td>
<td>Generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs.</td>
</tr>
<tr>
<td>genPBEKey (p. 100)</td>
<td>(This command is not supported on the FIPS-validated HSMs.)</td>
</tr>
<tr>
<td>genRSAKeyPair (p. 100)</td>
<td>Generates an RSA asymmetric key pair in your HSMs.</td>
</tr>
<tr>
<td>genSymKey (p. 104)</td>
<td>Generates a symmetric key in your HSMs.</td>
</tr>
<tr>
<td>getKeyInfo (p. 113)</td>
<td>Gets the HSM user IDs of users who can use the key.</td>
</tr>
<tr>
<td></td>
<td>If the key is quorum controlled, it gets the number of users in the quorum.</td>
</tr>
<tr>
<td>imSymKey (p. 116)</td>
<td>Imports a plaintext copy of a symmetric key from a file into the HSM.</td>
</tr>
<tr>
<td>listAttributes (p. 122)</td>
<td>Lists the attributes of an AWS CloudHSM key and the constants that represent them.</td>
</tr>
<tr>
<td>listUsers (p. 123)</td>
<td>Gets the users in the HSMs, their user type and ID, and other attributes.</td>
</tr>
<tr>
<td>setAttribute (p. 124)</td>
<td>Converts a session key to a persistent key.</td>
</tr>
<tr>
<td>unWrapKey (p. 126)</td>
<td>Imports a wrapped (encrypted) key from a file into the HSMs.</td>
</tr>
<tr>
<td>wrapKey (p. 129)</td>
<td>Exports an encrypted copy of a key from the HSM to a file on disk</td>
</tr>
</tbody>
</table>

**aesWrapUnwrap**

The **aesWrapUnwrap** command encrypts or decrypts the contents of a file on disk. This command is designed to wrap and unwrap encryption keys, but you can use it on any file that contains less than 4 KB (4096 bytes) of data.

**aesWrapUnwrap** uses AES Key Wrap. It uses an AES key on the HSM as the wrapping or unwrapping key. Then it writes the result to another file on disk.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) 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. 82) 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. 116) 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: 78
```
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. Type the key handle of an AES key on the HSM. This parameter is required. To find key handles, use the findKey (p. 87) command.

To create a wrapping key, use genSymKey (p. 104) to create an AES key (type 31). To verify that a key can be used as a wrapping key, use getAttribute (p. 110) to get the value of the OBJ_ATTR_WRAP attribute, which is represented by constant 262.

Note
Key handle 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

-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.
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. 75) and login (p. 75) 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. 87).
  - Required: Yes

**Related Topics**

- `findKey` (p. 87)

**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. 75) and `login` (p. 75) 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: Cfm3command-name returned:.

In this example, getKeyInfo (p. 113) indicates that the current user (user 4) can use the key in cryptographic operations. Nevertheless, when the user tries to use deleteKey (p. 80) 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

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. 129). To import a plaintext key, like the ones that exSymKey exports, use imSymKey (p. 116).
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. 130), 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 **exPubKey**. To export private keys, use **exportPrivateKey**.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) 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. 104)` 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. 87)` 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. 130).

```
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
0x94

// 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.
```
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 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` 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
```

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 8 -w 262168 -out aes256_H8.key
```

Wrapped Symmetric Key written to file "aes256_H8.key"
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. 87) command.

To verify that a key can be exported, use the getAttribute (p. 110) 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. 113) command.
Required: Yes

-w
Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey (p. 87) 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. 110) 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. 104) 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. 104)
- imSymKey (p. 116)
- wrapKey (p. 129)

**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. 110).

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. 75) and login (p. 75) to the HSM as a crypto user (CU).

**Syntax**

```plaintext
findKey -h
findKey [-c <key class>] [-t <key type>] [-l <key label>] [-id <key ID>] [-sess {0 | 1}] [-u <user-ldas>] [-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. 110)`. To determine whether the current user owns or shares a particular key, use `getKeyInfo (p. 113)` or `findAllKeys (p. 55)` in `cloudhsm_mgmt_util`.

**Command:** `findKey`

Total number of keys present 13

number of keys matched from start index 0::12
6, 7, 524296, 9, 262154, 262155, 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 --label 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. 123).

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. 55) 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 an 8-byte hash or checksum of a key. The HSM calculates a KCV when it generates the 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. 110).
AWS CloudHSM uses the following standard method to generate a key check value:

- **Symmetric keys**: First 8 bytes of the result of encrypting 16 zero-filled bytes with the key.
- **Asymmetric key pairs**: First 8 bytes of the modulus hash.

**Required**: No

**Output**

The **findKey** output lists the total number of matching keys and their key handles.

<table>
<thead>
<tr>
<th>Command: findKey</th>
<th>Total number of keys present 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of keys matched from start index 0::9</td>
<td>6, 7, 8, 9, 10, 11, 262156, 262157, 262158, 262159</td>
</tr>
</tbody>
</table>

**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. 90)
- getKeyInfo (p. 113)
- getAttribute (p. 110)
- findAllKeys (p. 55) in cloudhsm_mgmt_util
- Key Attribute Reference (p. 130)

**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. 75) and login (p. 75) to the HSM as a crypto user (CU).

**Syntax**

```bash
findSingleKey -h
findSingleKey -k <key-handle>
```

**Example**

**Example**

This command verifies that key 252136 exists on all three HSMs in the cluster.

<table>
<thead>
<tr>
<th>Command: findSingleKey -k 252136</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

Cluster Error Status
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. 123) command.
Required: Yes

Related Topics
• findKey (p. 123)
• getKeyInfo (p. 123)
• getAttribute (p. 87)

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. 75) and login (p. 75) 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. 110). To find the keys for a particular user, use getKeyInfo (p. 113). To find keys based on their attribute values, use findKey (p. 87).

Syntax

```
genDSAKeyPair -h

ngenDSAKeyPair -m <modulus length>
  -l <label>
  [-id <key ID>]
  [-minsrv <minimum number of servers>]
  [-m_value <0..8>]
  [-nex]
  [-rel]
  [-timeo <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`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genDSAKeyPair:</td>
<td>returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>genDSAKeyPair:</td>
<td>public key handle: 19 private key handle: 21</td>
</tr>
<tr>
<td>Cluster Error Status</td>
<td>Node id 0 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

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`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genDSAKeyPair:</td>
<td>returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>genDSAKeyPair:</td>
<td>public key handle: 12 private key handle: 14</td>
</tr>
<tr>
<td>Cluster Error Status</td>
<td>Node id 0 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>To confirm that the key pair exists only in the session, use the <code>-sess</code> parameter of <code>findKey (p. 87)</code> with a value of 1 (true).</td>
<td></td>
</tr>
</tbody>
</table>

Command: `findKey -sess 1`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findKey:</td>
<td>returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>Total number of keys present 2</td>
<td></td>
</tr>
<tr>
<td>number of keys matched from start index 0::1</td>
<td></td>
</tr>
<tr>
<td>12, 14</td>
<td></td>
</tr>
<tr>
<td>Cluster Error Status</td>
<td>Node id 0 and err state 0x00000000 : HSM Return: SUCCESS</td>
</tr>
</tbody>
</table>

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`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>genDSAKeyPair:</td>
<td>returned: 0x00 : HSM Return: SUCCESS</td>
</tr>
<tr>
<td>genDSAKeyPair:</td>
<td>public key handle: 11 private key handle: 19</td>
</tr>
</tbody>
</table>
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 \(-\text{attest}\) 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]
```

This command uses `getKeyInfo (p. 113)` on the private key (key handle 17). The output confirms that the key is owned by the current user (user 3) and that it is shared with users 4 and 6 (and no others). The output also shows that quorum authentication is enabled and the quorum size is two.

```
Command:  getKeyInfo -k 17
Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS
Owned by user 3
also, shared to following 2 user(s):
     4
     6
   2 Users need to approve to use/manage this key
```

Parameters

\(-h\)

Displays help for the command.

Required: Yes

\(-m\)

Specifies the length of the modulus in bits. The only valid value is 2048.

Required: Yes
-l

Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id

Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from 0 to 8.

This parameter establishes a quorum authentication requirement for the private key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the m_value of a key, use getKeyInfo (p. 113).

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. 139). 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. 124).

Default: The key is persistent.

Required: No

-timout

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. 123)`. To share and unshare existing keys, use `shareKey`.

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. 100)`
- `genSymKey (p. 104)`
- `genECCKeyPair (p. 95)`

`genECCKeyPair`

The `genECCKeyPair` command in the `key_mgmt_util` tool generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs. You must specify the elliptic curve type and a label for the keys. 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. 75) and login (p. 75) 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. 110). To find the keys for a particular user, use `getKeyInfo` (p. 113). To find keys based on their attribute values, use `findKey` (p. 87).

**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**

These examples show how to use `genECCKeyPair` to create ECC key pairs in your HSMs.

**Example : Create and Examine an ECC Key Pair**

This command creates an ECC key pair using an NID_sect571r1 elliptic curve and an ecc12 label. 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 12 -l ecc12`

```
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 might want to examine its attributes. The next command uses `getAttribute` (p. 110) 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
```

This command gets the content of the `attr_262177` attribute file. The output shows that 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
```
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 (1 - 15).

Valid values:
- 1: NID_X9_62_prime192v1
- 2: NID_X9_62_prime256v1
- 3: NID_sect163k1
- 4: NID_sect163r2
- 5: NID_sect233k1
- 6: NID_sect233r1
- 7: NID_sect283k1
- 8: NID_sect283r1
- 9: NID_sect409k1
- 10: NID_sect409r1
- 11: NID_sect571k1
- 12: NID_sect571r1
- 13: NID_secp224r1
- 14: NID_secp384r1
- 15: NID_secp521r1

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 getKeysInfo (p. 113).

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. 139). 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. 124).

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. 123). To share and unshare existing keys, use shareKey.
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. 104)
• genRSAKeyPair (p. 100)
• genDSAKeyPair (p. 91)

**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. 104). To create asymmetric key pairs, use `genRSAKeyPair` (p. 100), `genDSAKeyPair` (p. 91), or `genECCKeyPair` (p. 95).

**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. 75) and `login` (p. 75) 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. 110). To find the keys for a particular user, use `getKeyInfo` (p. 113). To find keys based on their attribute values, use `findKey` (p. 87).

**Syntax**

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>genRSAKeyPair</td>
<td>-h</td>
</tr>
<tr>
<td></td>
<td>-m &lt;modulus length&gt;</td>
</tr>
<tr>
<td></td>
<td>-e &lt;public exponent&gt;</td>
</tr>
<tr>
<td></td>
<td>-l &lt;label&gt;</td>
</tr>
<tr>
<td></td>
<td>[-id &lt;key ID&gt;]</td>
</tr>
<tr>
<td></td>
<td>[-min_srv &lt;minimum number of servers&gt;]</td>
</tr>
<tr>
<td></td>
<td>[-m_value &lt;0..8&gt;]</td>
</tr>
<tr>
<td></td>
<td>[-nex]</td>
</tr>
<tr>
<td></td>
<td>[-sess]</td>
</tr>
<tr>
<td></td>
<td>[-timeout &lt;number of seconds&gt;]</td>
</tr>
</tbody>
</table>
Examples

These examples show how to use `genRSAKeyPair` to create asymmetric key pairs in your HSMs.

**Example : Create and Examine an RSA Key Pair**

This command creates an RSA key pair with a 2048-bit modulus and an exponent of 65541. The output shows that the public key handle is 262159 and the private key handle is 262160.

```
Command: genRSAKeyPair -m 2048 -e 65541 -l rsa_test
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS
Cfm3GenerateKeyPair: public key handle: 262159 private key handle: 262160
Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

The next command uses `getAttribute` to get the attributes of the public key that we just created. It writes the output to the `attr_262159` file. It is followed by a `cat` command that gets the content of the attribute file. For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

```
The resulting hexadecimal values confirm that it is a public key (OBJ_ATTR_CLASS 0x02) with a type of RSA (OBJ_ATTR_KEY_TYPE 0x00). You can use this public key to encrypt (OBJ_ATTR_ENCRYPT 0x01), but not to decrypt (OBJ_ATTR_DECRYPT 0x00) or wrap (OBJ_ATTR_WRAP 0x00). The results also include the key length (512, 0x200), the modulus, the modulus length (2048, 0x800), and the public exponent (65541, 0x10005).

```

```
Command: getAttribute -o 262159 -a 512 -out attr_262159
got all attributes of size 731 attr cnt 20
Attributes dumped into attr_262159 file

      Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

$ cat attr_262159
OBJ_ATTR_CLASS
0x02
OBJ_ATTR_KEY_TYPE
0x00
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x00
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x01
OBJ_ATTR_LOCAL
0x01
Example: Generate a Shared RSA Key Pair

This command generates an RSA key pair and shares the private key with user 4, another CU on the HSM. The command uses the m_value parameter to require at least two approvals before the private key in the pair can be used in a cryptographic operation. When you use the m_value parameter, you must also use -u in the command and the m_value cannot exceed the total number of users (number of values in -u + owner).

Command:  
```
genRSAKeyPair -m 2048 -e 195193 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2
```

Cfm3GenerateKeyPair returned: 0x00  HSM Return: SUCCESS

Cfm3GenerateKeyPair:  
public key handle: 27  
private key handle: 28

Cluster Error Status
Node id 0 and err state 0x00000000  HSM Return: SUCCESS
Node id 1 and err state 0x00000000  HSM Return: SUCCESS

Parameters

-h
Displays help for the command.
Required: Yes

-m
Specifies the length of the modulus in bits. The minimum value is 2048.
Required: Yes

-e
Specifies the public exponent. The value must be an odd number greater than or equal to 65537.
Required: Yes
-l
Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id
Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv
Specifies the minimum number of HSMs on which the key is synchronized before the value of the -timeout parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of min_srv to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value
Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from 0 to 8.

This parameter establishes a quorum authentication requirement for the private key. The default value, 0, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the m_value of a key, use getKeyInfo (p. 113).

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. 139). 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. 124).

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. 123). To share and unshare existing keys, use shareKey.

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. 104)
- createKeyPair
- genDSAKeyPair (p. 91)
- genECCKeyPair (p. 95)

**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. 75) and login (p. 75) 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. 110). To find the keys for a particular user, use getKeyInfo (p. 113). To find keys based on their attribute values, use findKey (p. 87).

**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.

**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 Cfm3Util singlecmd loginHSM -u CU -s example_user -p example_pwd genSymKey -l aes -t 31 -s 32 -min_srv 1 -timeout 1;
done;
```

Example: Create a Quorum Authorized Generic Key

This command creates a 2048-bit generic secret key with the label `generic-mV2`. The command uses the `-u` parameter to share the key with another CU, user 6. It uses the `-m_value` parameter to require a quorum of at least two approvals for any cryptographic operations that use the key. The command also uses the `-attest` parameter to verify the integrity of the firmware on which the key is generated.

The output shows that the command generated a key with key handle 9 and that the attestation check on the cluster firmware passed.

```
Command: genSymKey -t 16 -s 2048 -l generic-mV2 -m_value 2 -u 6 -attest
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Created. Key Handle: 9
Attestation Check: [PASS]
Cluster Error Status
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example: Create and Examine a Key

This command creates a Triple DES key with a `3DES_shared` label and an ID of `IT-02`. The key can be used by the current user, and users 4 and 5. The command fails if the ID is not unique in the cluster or if the current user is user 4 or 5.

The output shows that the new key has key handle 7.

```
Command: genSymKey -t 21 -s 24 -l 3DES_shared -id IT-02 -u 4,5
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Created. Key Handle: 7
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

To verify that the new 3DES key is owned by the current user and shared with users 4 and 5, use `getKeyInfo (p. 113)`. 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. 110). The first command uses getAttribute to get all attributes (-a 512) of key handle 7 (-o 7). It writes them to the attr_7 file. The second command uses cat to get the contents of the attr_7 file.

This command confirms that key 7 is a 192-bit (OBJ_ATTR_VALUE_LEN 0x00000018 or 24-byte) 3DES (OBJ_ATTR_KEY_TYPE 0x15) symmetric key (OBJ_ATTR_CLASS 0x04) with a label of 3DES_shared (OBJ_ATTR_LABEL 3DES_shared) and an ID of IT-02 (OBJ_ATTR_ID IT-02). The key is persistent (OBJ_ATTR_TOKEN 0x01) and extractable (OBJ_ATTR_EXTRACTABLE 0x01) and can be used for encryption, decryption, and wrapping.

For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

Command:  

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
107
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 `-s 24`.
Valid values for each key type:
- AES: 16 (128 bits), 24 (192 bits), 32 (256 bits)
- 3DES: 24 (192 bits)
- RC4: <256 (2048 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. 113).

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. 139).

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. 124).

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 \texttt{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

\textbf{-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 \texttt{-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 \texttt{listUsers (p. 123)}. To share and unshare existing keys, use \texttt{shareKey}.

Default: Only the current user can use the key.

Required: No

**Related Topics**

- \texttt{exSymKey (p. 82)}
- \texttt{genRSAKeyPair (p. 100)}
- \texttt{genDSAKeyPair (p. 91)}
- \texttt{genECCKeyPair (p. 95)}

**getAttribute**

The \texttt{getAttribute} command in \texttt{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, \texttt{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 \texttt{getAttribute} only on keys that you own and key that are shared with you. You can run this command or the \texttt{getAttribute (p. 58)} command in \texttt{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 \texttt{listAttributes (p. 122)} command. To change the attribute values of existing keys, use \texttt{setAttribute (p. 124)} in \texttt{key\_mgmt\_util} and \texttt{setAttribute (p. 69)} in \texttt{cloudhsm\_mgmt\_util}. For help interpreting the key attributes, see the \texttt{Key Attribute Reference (p. 130)}.

Before you run any \texttt{key\_mgmt\_util} command, you must \texttt{start key\_mgmt\_util (p. 75)} and \texttt{login (p. 75)} 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. 122)`, 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. 130).

Command: `listAttributes`

Description

The following are all of the possible attribute values for `getAttribute`.

<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ATTR_CLASS</td>
<td>0</td>
</tr>
<tr>
<td>OBJ_ATTR_TOKEN</td>
<td>1</td>
</tr>
<tr>
<td>OBJ_ATTR_PRIVATE</td>
<td>2</td>
</tr>
<tr>
<td>OBJ_ATTR_LABEL</td>
<td>3</td>
</tr>
<tr>
<td>OBJ_ATTR_KEY_TYPE</td>
<td>256</td>
</tr>
<tr>
<td>OBJ_ATTR_ID</td>
<td>258</td>
</tr>
<tr>
<td>OBJ_ATTR_SENSITIVE</td>
<td>259</td>
</tr>
<tr>
<td>OBJ_ATTR_ENCRYPT</td>
<td>260</td>
</tr>
<tr>
<td>OBJ_ATTR_DECRYPT</td>
<td>261</td>
</tr>
<tr>
<td>OBJ_ATTR_WRAP</td>
<td>262</td>
</tr>
<tr>
<td>OBJ_ATTR_UNWRAP</td>
<td>263</td>
</tr>
<tr>
<td>OBJ_ATTR_SIGN</td>
<td>264</td>
</tr>
<tr>
<td>OBJ_ATTR_VERIFY</td>
<td>266</td>
</tr>
<tr>
<td>OBJ_ATTR_LOCAL</td>
<td>355</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS</td>
<td>288</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS_BITS</td>
<td>289</td>
</tr>
<tr>
<td>OBJ_ATTR_PUBLIC_EXPONENT</td>
<td>290</td>
</tr>
<tr>
<td>OBJ_ATTR_VALUE_LEN</td>
<td>353</td>
</tr>
<tr>
<td>OBJ_ATTR_EXTRACTABLE</td>
<td>354</td>
</tr>
<tr>
<td>OBJ_ATTR_KCV</td>
<td>371</td>
</tr>
</tbody>
</table>

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. 130).

```bash
$ 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`
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. 130).

```bash
$ 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. 87).

  Also, you must own the specified key or it must be shared with you. To find the users of a key, use `getKeyInfo` (p. 113).
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. 122). For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

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. 58) in cloudhsm_mgmt_util
- listAttributes (p. 122)
- setAttribute (p. 124)
- findKey (p. 87)
- Key Attribute Reference (p. 130)

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. 123). To find the keys for a particular user, use findKey (p. 87) –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. 71) in cloudhsm_mgmt_util.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) 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 an Asymmetric Key

This command gets the users who can use the AES (asymmetric) 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 a Symmetric Key Pair

These commands use `getKeyInfo` to get the users who can use the keys in an RSA (symmetric) 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. 87).

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. 100) 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. 123) command.
Required: Yes

Related Topics

- getKeyInfo (p. 61) in cloudhsm_mgmt_util
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` 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` and `unWrapKey` commands.

Also, the `imSymKey` command exports only symmetric keys. To import public keys, use `importPubKey`. To import private keys, use `importPrivateKey` or `wrapKey`.

Imported keys work very much like keys generated in the HSM. However, the value of the `OBJ_ATTR_LOCAL` attribute is zero, which indicates that it was not generated locally. The `imSymKey` command does not have a parameter that shares the key with other users, but you can use the `shareKey` command in `cloudhsm_mgmt_util` to share the key after it is imported.

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` and `login` to the HSM as a crypto user (CU).

### Syntax

```
imSymKey -h
nimSymKey -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> ]
```

### 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.
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.

```
openssl rand -out aes256-forImport.key 32
```

```
The command:

```
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. 110) to get the OBJ_ATTR_LOCAL attribute (attribute 355 (p. 130)) of the newly imported key and writes it to the `attr_262180` file.
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
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` 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. 110)` 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 -outattr_14_kcv
Attributes dumped into attr_14_kcv file
$  cat attr_14_kcv
OBJ_ATTR_KCV
0xc33cbd
```

This similar command uses `getAttribute (p. 110)` 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. 110) or findKey (p. 87) 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. A 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

-h

Displays help for the command.

Required: Yes

-f

Specifies the file that contains that key to import.

The file must contain a plaintext copy of an AES or Triple DES key of the specified length. RC4 and DES keys are not valid on FIPS-mode HSMs.
• **AES**: 16, 24 or 32 bytes  
• **Triple DES (3DES)**: 24 bytes

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the `findKey (p. 87)` 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. 110)` to get the value of the `OBJ_ATTR_WRAP` attribute (262). To create a wrapping key, use `genSymKey (p. 104)` 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

-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

-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

-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

-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. 124)`.

Default: The key is persistent.

Required: No

-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

-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

-minsrv

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 `minsrv` 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 `minsrv` parameter.

This parameter is valid only when the `minsrv` 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

Related Topics

- `genSymKey (p. 104)`
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. 110) and setAttribute (p. 124) commands. For help interpreting the key attributes, see the Key Attribute Reference (p. 130).

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) 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. 130).

To represent all attributes in the getAttribute (p. 110) 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. 66) in cloudhsm_mgmt_util
- getAttribute (p. 110)
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. 123) command in cloudhsm_mgmt_util.

The user commands in key_mgmt_util, listUsers and getKeyInfo, are read-only commands that crypto users (CUs) have permission to run. The remaining user management commands are part of cloudhsm_mgmt_util. They are run by crypto officers (CO) who have user management permissions.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) 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. 87), getAttribute (p. 110), and getKeyInfo (p. 113).

```
Command:  listUsers
Number Of Users found 4

Index       User ID     User Type       User Name           MofnPubKey  LoginFailureCnt       2FA
0            NO          PCO             admin                    NO          0                    1
0            NO          AU              app_user                  NO          0                    2
0            NO          CU             alice                     YES         0                    3
0            NO          CU             bob                        NO          0                    4
0            NO          CU             trent                     YES         0                    5

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. 42) commands.
- **User type (p. 11)**: 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. 142).
• **LoginFailureCnt:**
  * 2FA: Indicates that the user has enabled multi-factor authentication.

### Parameters

- **-h**
  Displays help for the command.
  Required: Yes

### Related Topics

- listUsers (p. 123) in cloudhsm_mgmt_util
- findKey (p. 87)
- getAttribute (p. 110)
- getKeyInfo (p. 113)

### 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. 75) and login (p. 75) 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. 104) 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
```
This command uses `findKey (p. 87)` 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

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. 130).

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

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

Command: `findKey -sess 0`

Total number of keys present 5

number of keys matched from start index 0::4
6, 7, 524296, 9, 262154
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. 87).
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. 122).
Required: Yes

Related Topics

- setAttribute (p. 69) in cloudhsm_mgmt_util
- getAttribute (p. 110)
- listAttributes (p. 122)
- Key Attribute Reference (p. 130)

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 from files that were created by the wrapKey (p. 129) command.

During the import process, unWrapKey uses an AES key on the HSM that you specify to unwrap (decrypt) the key in the file. Then it saves the key in the HSM with a key handle and the attributes that you specify. To export and import plaintext keys, use the exSymKey (p. 82) and imSymKey (p. 116) commands.

Imported keys work very much like keys generated in the HSM. However, the value of the OBJ_ATTR_LOCAL attribute (p. 130) is zero, which indicates that it was not generated locally. The unWrapKey command does not have parameters that assign a label or share the key with other users, but you can use the shareKey command in cloudhsm_mgmt_util (p. 42) to add those attributes after the key is imported.

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. 75) and login (p. 75) to the HSM as a crypto user (CU).

Syntax

```
unWrapKey -h
unWrapKey -f <key-file-name>
  -w <wrapping-key-handle> 4
    [-sess]
    [-min_srv <minimum-number-of-HSMs>]
    [-timeout <number-of-seconds>]
    [-attest]
```

Example

This command imports an wrapped (encrypted) copy of a Triple DES (3DES) symmetric key from the 3DES.key file into the HSMs. To unwrap (decrypt) the key, the command uses the -w parameter to specify key 6, an AES key on the HSM. The AES key that unwraps during import must be the same key that wrapped during export, or a cryptographically identical copy.

The output shows that the key in the file was unwrapped and imported. The new key has key handle 29.

If you are using unWrapKey to move a key between clusters, you must first create an AES wrapping key that exists on both clusters. You can generate a key outside of the HSMs and then use imSymKey to import it to the HSMs on both cluster. Or, generate an AES key in the HSMs on one cluster, use exSymKey (p. 82) to export it in plaintext to a file. Then use imSymKey to import the plaintext key into the other cluster. Once the wrapping key is established on both clusters, you can use wrapKey and unWrapKey to move encrypted keys between clusters without ever exposing the plaintext key.

```
Command: unWrapKey -f 3DES.key -w 6
CfmUnWrapKey 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
```

Parameters

- **-h**
  Displays help for the command.
  Required: Yes

- **-f**
  Specifies the path and name of the file that contains the wrapped key.
  Required: Yes

- **-w**
  Specifies the wrapping key. Type the key handle of an AES key on the HSM. This parameter is required. To find key handles, use the findKey (p. 87) command.
To create a wrapping key, use `genSymKey (p. 104)` to create an AES key (type 31). To verify that a key can be used as a wrapping key, use `getAttribute (p. 110)` to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by constant 262.

**Note**

Key handle 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

**-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. 124)`.

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

**Related Topics**

- `wrapKey (p. 129)`
• exSymKey (p. 82)
• imSymKey (p. 116)

wrapKey

The `wrapKey` command in key_mgmt_util exports an encrypted copy of a symmetric or private key from the HSM to a file on disk. When you run `wrapKey`, you specify the key to export, an AES key on the HSM to encrypt (wrap) the key to be exported, 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, change its key attributes (p. 130), 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 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.

To import (and unwrap) the encrypted key from the file to an HSM, use `unWrapKey` (p. 126). To export a plaintext key from the HSM, use `exSymKey` (p. 82). The `aesWrapUnwrap` (p. 77) command cannot decrypt (unwrap) keys that `wrapKey` encrypts.

Before you run any key_mgmt_util command, you must start key_mgmt_util (p. 75) and login (p. 75) to the HSM as a crypto user (CU).

Syntax

```
wrapKey -h
wrapKey -k <exported-key-handle>
  -w <wrapping-key-handle>
  -out <output-file>
```

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
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.
-k

Specifies the key handle of the key to export. Type the key handle of a symmetric or private key that you own. To find key handles, use the findKey (p. 87) command.

To verify that a key can be exported, use the getAttribute (p. 110) 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. 130).

Also, you can export only keys that you own. To find the owner of a key, use the getKeyInfo (p. 113) command.

Required: Yes

-w

Specifies the wrapping key. Type the key handle of an AES key on the HSM. This parameter is required. To find key handles, use the findKey (p. 87) command.

To create a wrapping key, use genSymKey (p. 104) to create an AES key (type 31). To verify that a key can be used as a wrapping key, use getAttribute (p. 110) to get the value of the OBJ_ATTR_WRAP attribute, which is represented by constant 262.

Note
Key handle 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 an encrypted copy of the exported key. If the file already exists, the command overwrites it without warning.

Required: Yes

Related Topics
- exSymKey (p. 82)
- imSymKey (p. 116)
- unWrapKey (p. 126)

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. 124) 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. 122). To get the attribute values for a key, use getAttribute (p. 110).
The following table lists the key attributes, their constants, and their valid values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Constant</th>
<th>Values</th>
</tr>
</thead>
</table>
| OBJ_ATTR_CLASS    | 0        | 2: Public key in a public–private key pair.  
                       |          | 3: Private key in a public–private key pair.  
<pre><code>                   |          | 4: Secret (symmetric) key.                        |
</code></pre>
<p>| OBJ_ATTR_TOKEN    | 1        | 0: False. Session key.                                                |
|                   |          | 1: True. Persistent key.                                              |
| OBJ_ATTR_PRIVATE  | 2        | 0: False.                                                              |
|                   |          | 1: True. Private key in a public–private key pair.                     |
| OBJ_ATTR_LABEL    | 3        | User-defined string. It does not have to be unique in the cluster.    |
| 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.                     |
| OBJ_ATTR_ENCRYPT  | 260      | 0: False.                                                              |
|                   |          | 1: True. The key can be used to encrypt data.                         |
| 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.                         |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Constant</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ_ATTR_UNWRAP</td>
<td>263</td>
<td>0: False. 1: True. The key can be used to decrypt keys.</td>
</tr>
<tr>
<td>OBJ_ATTR_SIGN</td>
<td>264</td>
<td>0: False. 1: True. The key can be used for signing (private keys).</td>
</tr>
<tr>
<td>OBJ_ATTR_VERIFY</td>
<td>266</td>
<td>0: False. 1: True. The key can be used for verification (public keys).</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS</td>
<td>288</td>
<td>The modulus that was used to generate an RSA key pair. For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_MODULUS_BITS</td>
<td>289</td>
<td>The length of the modulus used to generate an RSA key pair. For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_PUBLIC_EXPONENT</td>
<td>290</td>
<td>The public exponent used to generate an RSA key pair. For other key types, this attribute does not exist.</td>
</tr>
<tr>
<td>OBJ_ATTR_VALUE_LEN</td>
<td>353</td>
<td>Key length in bits.</td>
</tr>
<tr>
<td>OBJ_ATTR_EXTRACTABLE</td>
<td>354</td>
<td>0: False. 1: True. The key can be exported from the HSMs.</td>
</tr>
<tr>
<td>OBJ_ATTR_LOCAL</td>
<td>355</td>
<td>0: False. The key was imported into the HSMs. 1: True.</td>
</tr>
<tr>
<td>OBJ_ATTR_KCV</td>
<td>371</td>
<td>Key check value of the key. For more information, see Additional Details (p. 132).</td>
</tr>
<tr>
<td>OBJ_ATTR_ALL</td>
<td>512</td>
<td>Represents all attributes.</td>
</tr>
</tbody>
</table>

**Additional Details**

**Key check value (kcv)**

The key check value (KCV) is an 8-byte hash or checksum of a key. The HSM calculates a KCV when it generates the 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. 110).` 

AWS CloudHSM uses the following standard method to generate a key check value:

- **Symmetric keys**: First 8 bytes of the result of encrypting 16 zero-filled bytes with the key.
- **Asymmetric key pairs**: First 8 bytes of the modulus hash.
Managing HSM Users and Keys in AWS CloudHSM

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

**Topics**

- Managing HSM Users in AWS CloudHSM (p. 134)
- Managing Keys in AWS CloudHSM (p. 137)
- Enforcing Quorum Authentication (M of N Access Control) (p. 142)

Managing HSM Users in AWS CloudHSM

To manage users on the HSMs in your AWS CloudHSM cluster, use the AWS CloudHSM command line tool known as cloudhsm_mgmt_util. Before you can manage users, you must start cloudhsm_mgmt_util, enable end-to-end encryption, and log in to the HSMs. For more information, see cloudhsm_mgmt_util (p. 42).

To manage HSM users, log in to the HSM with the user name and password of a crypto officer (p. 12) (CO). Only COs can manage other users. The HSM contains a default CO named admin. You set this user's password when you activated the cluster (p. 31).

**Topics**

- Create Users (p. 134)
- List Users (p. 135)
- Change a User's Password (p. 136)
- Delete Users (p. 136)

Create Users

Use the `createUser` (p. 49) command to create a user on the HSM. The following examples create new CO and CU users, respectively. For information about user types, see HSM Users (p. 11).

```bash
aws-cloudhsm> createUser CO example_officer <password>
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
******************************************************************************
Do you want to continue(y/n)?y
Creating User example_officer(CO) on 3 nodes

aws-cloudhsm> createUser CU example_user <password>
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster.
******************************************************************************
```
cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.

****************************************************************

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

Creating User example_user(CU) on 3 nodes

The following shows the syntax for the createUser (p. 49) command. User types and passwords are case-sensitive in cloudhsm_mgmt_util commands, but user names are not.

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

List Users

Use the listUsers (p. 67) command to list the users on each HSM in the cluster. All HSM user types (p. 11) can use this command; it’s not restricted to COs.

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

aws-cloudhsm>listUsers

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

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

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

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

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

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

Use the `changePswd` command to change the password for the any user. All HSM user types (p. 11) can issue this command, but only COs can change the password for other users. Crypto users (CUs) and appliance users (AUs) can change only their own password. The following examples change the password for the CO and CU users that were created in the Create Users (p. 134) examples.

/aws-cloudhsm>changePswd CO example_officer <new password>
*******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
*******************************************************************************
Do you want to continue(y/n)?y
Changing password for example_officer(CO) on 3 nodes

/aws-cloudhsm>changePswd CU example_user <new password>
*******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
*******************************************************************************
Do you want to continue(y/n)?y
Changing password for example_user(CU) on 3 nodes

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

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

Delete Users

Use the `deleteUser` command to delete a user. The following examples delete the CO and CU users that were created in the Create Users (p. 134) examples.

/aws-cloudhsm>deleteUser CO example_officer
Deleting user example_officer(CO) on 3 nodes
deleteUser success on server 0(10.0.2.9)
deleteUser success on server 1(10.0.3.11)
deleteUser success on server 2(10.0.1.12)

/aws-cloudhsm>deleteUser CU example_user
Deleting user example_user(CU) 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)

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

/aws-cloudhsm>deleteUser <user type> <user name>
Managing Keys in AWS CloudHSM

To manage keys on the HSMs in your AWS CloudHSM cluster, use 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.

To manage keys, log in to the HSM (p. 75) with the user name and password of a crypto user (CU). Only CUs can create keys. Keys are inherently owned and managed by the CU who created them.

Topics
- Generate Keys (p. 137)
- Import Keys (p. 138)
- Export Keys (p. 139)
- Delete Keys (p. 141)
- Share and Unshare Keys (p. 141)

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` 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` 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
Generate ECC (Elliptic Curve Cryptography) Key Pairs

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

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

```
Command: genECCKeyPair -i 14 -l ecc-p384
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS
Cfm3GenerateKeyPair: public key handle: 524297 private key handle: 524298
```

Import Keys

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

Import Secret Keys

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

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

To import a secret key

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

```
Command: genSymKey -t 31 -s 16 -sess -l import-wrapping-key
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS
Symmetric Key Created. Key Handle: 524299
```

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

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

```
Command: imSymKey -f aes256.key -t 31 -l aes256-imported -w 524299
Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS
```
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. 82) 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. 129) command.
Export Secret Keys

Complete the following steps to export a secret key.

To export a secret key

1. Use the `genSymKey` (p. 104) 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:
   ```bash
   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. 82) 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:
     ```bash
     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. 129) 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:
     ```bash
     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:
```bash
exportPubKey -k 524294 -out rsa2048.pub.exp
```

PEM formatted public key is written to rsa2048.pub.key
Delete Keys

Use the `deleteKey` (p. 80) command to delete a key, as in the following example. Only the key owner can delete a key.

```bash
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. 104) or `genRSAKeyPair` (p. 100) commands. To share existing keys with a different HSM user, use the `cloudhsm_mgmt_util` (p. 42) command line tool. This is different from most of the tasks documented in this section, which use the `key_mgmt_util` (p. 74) 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.

```bash
aws-cloudhsm>shareKey 524295 4 1
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
Do you want to continue(y/n)?y
shareKey success on server 0(10.0.2.9)
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.

```bash
aws-cloudhsm>shareKey <key handle> <user ID> <Boolean: 1 for share, 0 for unshare>
```
Enforcing Quorum Authentication (M of N Access Control)

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

Quorum authentication can control the following operations:

- HSM user management by crypto officers (COs) (p. 12) – Creating and deleting HSM users, and changing a different HSM user's password. For more information, see Using Quorum Authentication for Crypto Officers (p. 147).
- Cryptographic operations by crypto users (CUs) (p. 12) – For example:
  - Using asymmetric private keys on the HSM to cryptographically sign messages.
  - Using AES symmetric keys on the HSM for AES wrap and unwrap.

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

Topics

- Overview of Quorum Authentication (p. 142)
- Additional Details about Quorum Authentication (p. 143)
- Using Quorum Authentication for Crypto Officers: First Time Setup (p. 143)
- Using Quorum Authentication for Crypto Officers (p. 147)
- Change the Quorum Minimum Value for Crypto Officers (p. 153)

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. 147).

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). For cryptographic operations by CUs, the largest number you can choose is eight (8).
- The HSM can store up to 1024 quorum tokens. If the HSM already has 1024 tokens when you try to create a new one, the HSM purges one of the expired tokens. By default, tokens expire ten minutes after their creation.

Using Quorum Authentication for Crypto Officers: First Time Setup

The following topics describe the steps that you must complete to configure your HSM so that crypto officers (COs) (p. 12) 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. 147).

Topics
- Prerequisites (p. 143)
- Create and Register a Key for Signing (p. 144)
- Set the Quorum Minimum Value on the HSM (p. 146)

Prerequisites

To understand this example, you should be familiar with the cloudhsm_mgmt_util command line tool (p. 42). 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. 134).

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

<table>
<thead>
<tr>
<th>User Id</th>
<th>User FailureCnt</th>
<th>User Type</th>
<th>User Name</th>
<th>MofnPubKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>PCO</td>
<td>admin</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>AU</td>
<td>app_user</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>CO</td>
<td>officer1</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>CO</td>
<td>officer2</td>
<td>NO</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>CO</td>
<td>officer3</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>CO</td>
<td>officer4</td>
<td>NO</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>CO</td>
<td>officer5</td>
<td>NO</td>
</tr>
</tbody>
</table>
```
Users on server 1(10.0.1.4):
Number of users found:7

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

Create and Register a Key for Signing

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

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

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

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

```
$ openssl genrsa -out officer1.key -aes256 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.e is 65537 (0x10001)
Enter pass phrase for officer1.key:
Verifying - Enter pass phrase for officer1.key:
```

Each CO should create his or her own key.

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

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

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

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

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

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 44).

4. Use the `registerMofnPubKey` command to register the public key. For more information, see the following example or use the `help registerMofnPubKey` command.
Example – Register a public key with the HSM

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

- **key_match_string** – An arbitrary string that is used to match the public and private keys. You can use any string for this value. The `cloudhsm_mgmt_util` command line tool encrypts this string with the private key, and then sends the encrypted blob and the plaintext string to the HSM. The HSM uses the public key to decrypt the encrypted blob, and then compares the decrypted string to the plaintext string. If the strings match, the HSM registers the public key; otherwise it doesn’t.

- **officer1** – The user name of the CO who is registering the public key. This must be the same CO who is logged in to the HSM and is running this command.

- **officer1.key** – The name of the file that contains the CO’s key. This file must contain the complete key (not just the public part) because the `cloudhsm_mgmt_util` command line tool uses the private key to encrypt the `key_match_string`.

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

```
aws-cloudhsm> registerMofnPubKey CO key_match_string officer1 officer1.key
**********************************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
**********************************************************************************************
Do you want to continue(y/n)?y
Enter PEM pass phrase:
registerMofnPubKey success on server 0(10.0.2.14)
registerMofnPubKey success on server 1(10.0.1.4)
```

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

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

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

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

145
Set the Quorum Minimum Value on the HSM

To use quorum authentication for COs, a CO must log in to the HSM and then set the quorum minimum value, also known as the \( m \) value. This is the minimum number of CO approvals that are required to perform HSM user management operations. Any CO on the HSM can set the quorum minimum value, including COs that have not registered a key for signing. You can change the quorum minimum value at any time; for more information, see Change the Quorum Value for Crypto Officers (p. 153).

To set the quorum minimum value on the HSM

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

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

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

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 44).

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

Example – Set the quorum minimum value on the HSM

This example uses a quorum minimum value of two. You can choose any value from two to twenty, up to the total number of COs on the HSM. In this example, the HSM has six COs (the PCO user (p. 12) 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.

```bash
aws-cloudhsm> setMValue 3 2
****************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
****************************************************************
Do you want to continue(y/n)? y
Setting M Value(2) for 3 on 2 nodes
```

In the preceding example, the first number (3) identifies the HSM service whose quorum minimum value you are setting.
The following table lists the HSM service identifiers along with their names, descriptions, and the commands that are included in the service.

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

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

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

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

After you complete these steps, see Using Quorum Authentication for Crypto Officers (p. 147).

Using Quorum Authentication for Crypto Officers

A crypto officer (CO) 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
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
******************************************************************************
Do you want to continue(y/n)?y
Creating User user1(CU) on 2 nodes
createUser failed: RET_MXN_AUTH_FAILED
creating user on server 0(10.0.2.14) failed
Retry/Ignore/Abort?(R/I/A): A
```
To perform an HSM user management operation, a CO must complete the following tasks:

1. Get a quorum token (p. 148).
2. Get approvals (signatures) from other COs (p. 149).
3. Approve the token on the HSM (p. 149).
4. Perform the HSM user management operation (p. 151).

If you have not yet configured the HSM for quorum authentication for COs, do that now. For more information, see First Time Setup for Crypto Officers (p. 143).

**Get a Quorum Token**

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

**To get a quorum token**

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

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

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

3. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 44).

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

**Example – Get a quorum token**

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

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

In the following command, 3 identifies the service for which you can use the token that you are getting. In this case, the token is for HSM user management operations (service 3). For more information, see Set the Quorum Minimum Value on the HSM (p. 146).

```
aws-cloudhsm> getToken 3 officer1 officer1.token
getToken success on server 0(10.0.2.14)
Token:
   Id:1
   Service:3
   Node:1
   Key Handle:0
   User:officer1
ggetToken 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.

```bash
$ 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.

```bash
$ 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.

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

3. Use the `enable_e2e` command to establish end-to-end encrypted communication.
4. Use the `loginHSM` command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 44).
5. Use the `approveToken` command to approve the signed token, passing the token approval file. For more information, see the following example.

Example – Create a token approval file and approve the signed token on the HSM

The token approval file is a text file in a particular format that the HSM requires. The file contains information about the token, its approvers, and the approvers' signatures. The following shows an example token approval file.
# For "Multi Token File Path", type the path to the file that contains
# the token. You can type the same value for "Token File Path", but
# that's not required. The "Token File Path" line is required in any
# case, regardless of whether you type a value.
Multi Token File Path = officer1.token;
Token File Path = ;

# Total number of approvals
Number of Approvals = 2;

# Approver 1
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer1;
Approval File = officer1.token.sig1;

# Approver 2
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer2;
Approval File = officer1.token.sig2;

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

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

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

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. 49) 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. 134).

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

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

```bash
aws-cloudhsm> createUser CU user1 password
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
******************************************************************************
Do you want to continue(y/n)? y
Creating User user1(CU) on 2 nodes
```

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

```bash
aws-cloudhsm> listUsers
Users on server 0(10.0.2.14):
Number of users found: 8

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

151
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. 146) so that crypto officers (COs) (p. 12) can use quorum authentication, you might want to change the quorum minimum value. The HSM allows you to change the quorum minimum value only when the number of approvers is the same or higher than the current quorum minimum value. For example, if the quorum minimum value is two, at least two COs must approve to change the quorum minimum value.

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

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

<table>
<thead>
<tr>
<th>Service Identifier</th>
<th>Service Name</th>
<th>Service Description</th>
<th>HSM Commands</th>
</tr>
</thead>
</table>
| 3                  | USER_MGMT    | HSM user management | • createUser  
|                    |              |                     | • deleteUser  
|                    |              |                     | • changePswd (applies only when changing the password of a different HSM user) |
| 4                  | MISC_CO      | Miscellaneous CO service | • 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 enable_e2e command to establish end-to-end encrypted communication.

3. Use the loginHSM command to log in to the HSM as a CO. For more information, see Log in to the HSMs (p. 44).

4. Use the getMValue command to get the quorum minimum value for service 3. For more information, see the following example.

5. Use the getMValue command to get the quorum minimum value for service 4. For more information, see the following example.

6. If the quorum minimum value for service 4 is lower than the value for service 3, use the setMValue command to change the value for service 4. Change the value for service 4 to one that is the same or higher than the value for service 3. For more information, see the following example.

7. Get a quorum token (p. 148), taking care to specify service 4 as the service for which you can use the token.

8. Get approvals (signatures) from other COs (p. 149).

9. Approve the token on the HSM (p. 149).

10. Use the setMValue command to change quorum minimum value for service 3 (user management operations performed by COs).
Example – Get quorum minimum values and change the value for service 4

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

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

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

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

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

```
aws-cloudhsm> setMValue 4 2
******************************************************************************
This is a CRITICAL operation, should be done on all nodes in the cluster. Cav server does NOT synchronize these changes with the nodes on which this operation is not executed or failed, please ensure this operation is executed on all nodes in the cluster.
******************************************************************************

Do you want to continue(y/n)? y
Setting M Value(2) for 4 on 2 nodes
```

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

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

aws-cloudhsm> getMValue 4
MValue of service 4[MISC_CO] on server 0 : [2]
MValue of service 4[MISC_CO] on server 1 : [2]
```
Using the AWS CloudHSM Software Libraries

To use your AWS CloudHSM cluster for cryptoprocessing, you use the AWS CloudHSM software libraries to integrate your applications with the HSMs in your cluster. See the following topics for more information about the available software libraries.

Topics
- AWS CloudHSM Software Library for PKCS #11 (p. 155)
- AWS CloudHSM Software Library for OpenSSL (p. 160)
- AWS CloudHSM Software Library for Java (p. 162)

AWS CloudHSM Software Library for PKCS #11

The AWS CloudHSM software library for PKCS #11 is a PKCS #11 standard implementation that communicates with the HSMs in your AWS CloudHSM cluster. The library supports PKCS #11 version 2.40, including the following key types, mechanisms, and APIs.

Topics
- Supported PKCS #11 Key Types (p. 155)
- Supported PKCS #11 Mechanisms (p. 155)
- Supported PKCS #11 APIs (p. 157)
- Install and Use the AWS CloudHSM Software Library for PKCS #11 (p. 158)

Supported PKCS #11 Key Types

The AWS CloudHSM software library for PKCS #11 supports the following key types.

- **RSA** – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.
- **ECDSA** – Generate keys with the P-224, P-256, P-384, and P-521 curves. Only the P-256 and P-384 curves are supported for sign/verify.
- **AES** – 128, 192, and 256-bit AES keys.
- **Triple DES (3DES)** – 192-bit keys.
- **GENERIC_SECRET** – 1 to 64 bytes.

Supported PKCS #11 Mechanisms

The AWS CloudHSM software library for PKCS #11 supports the following PKCS #11 mechanisms.

**Generate, Create, Import Keys**

- **CKM_AES_KEY_GEN**
• CKM_DES3_KEY_GEN
• CKM_EC_KEY_PAIR_GEN
• CKM_GENERIC_SECRET_KEY_GEN
• CKM_RSA_X9_31_KEY_PAIR_GEN

Note
This mechanism is functionally identical to the CKM_RSA_PKCS_KEY_PAIR_GEN mechanism, but offers stronger guarantees for $p$ and $q$ generation. If you need the CKM_RSA_PKCS_KEY_PAIR_GEN mechanism, use CKM_RSA_X9_31_KEY_PAIR_GEN.

Sign/Verify
• CKM_RSA_PKCS
• CKM_RSA_PKCS_PSS
• CKM_SHA256_RSA_PKCS
• CKM_SHA224_RSA_PKCS
• CKM_SHA384_RSA_PKCS
• CKM_SHA512_RSA_PKCS
• CKM_SHA1_RSA_PKCS_PSS
• CKM_SHA256_RSA_PKCS_PSS
• CKM_SHA224_RSA_PKCS_PSS
• CKM_SHA384_RSA_PKCS_PSS
• CKM_SHA512_RSA_PKCS_PSS
• CKM_MD5_HMAC
• CKM_SHA_1_HMAC
• CKM_SHA224_HMAC
• CKM_SHA256_HMAC
• CKM_SHA384_HMAC
• CKM_SHA512_HMAC
• CKM_ECDSA
• CKM_ECDSA_SHA1
• CKM_ECDSA_SHA224
• CKM_ECDSA_SHA256
• CKM_ECDSA_SHA384
• CKM_ECDSA_SHA512

Digest
• CKM_SHA1
• CKM_SHA224
• CKM_SHA256
• CKM_SHA384
• CKM_SHA512

Encrypt/Decrypt
• CKM_AES_CBC
**Supported PKCS #11 APIs**

The AWS CloudHSM software library for PKCS #11 supports the following PKCS #11 APIs.

- C_CreateObject
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DestroyObject
- C_DigestInit
- C_Digest
- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_Finalize
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo

**Note**

When performing AES-GCM encryption, the HSM ignores the initialization vector (IV) in the request and uses an IV that it generates. The HSM writes the generated IV to the memory reference pointed to by the `pAAD` element of the `CK_GCM_PARAMS` parameters structure that you supply.

- CKM_DES3_CBC
- CKM_DES3_CBC_PAD
- CKM_RSA_OAEP_PAD
- CKM_RSA_PKCS

**Key Derive**

- CKM_ECDH1_DERIVE

**Key Wrap**

- CKM_AES_KEY_WRAP

- CKM_AES_CBC_PAD
- CKM_AES_GCM
• C_GetMechanismInfo
• C_GetMechanismList
• C_GetOperationState
• C_GetSessionInfo
• C_GetSlotInfo
• C_GetSlotList
• C_GetTokenInfo
• C_Initialize
• C_Login
• C_Logout
• C_OpenSession
• C_Sign
• C_SignFinal
• C_SignInit
• C_SignRecover
• C_SignRecoverInit
• C_SignUpdate
• C_UnWrapKey
• C_Verify
• C_VerifyFinal
• C_VerifyInit
• C_VerifyRecover
• C_VerifyRecoverInit
• C_VerifyUpdate
• C_WrapKey

Install and Use the AWS CloudHSM Software Library for PKCS #11

AWS CloudHSM provides two software libraries for PKCS #11. One uses Redis to create a local cache for efficiency, which can increase performance. However, consider the following before you choose the library with Redis:

**Considerations**

- Redis caches all operations performed with the PKCS #11 library running on the same host, but it's not aware of operations that are performed outside the library. You can use another interface to modify keys on the HSMs in your cluster—for example, the command line tools (p. 42) or software library for Java (p. 162). But if you do, the Redis cache can fall out of sync with the HSMs. You can rebuild the cache to bring it back into sync, but it doesn't happen automatically.

- The PKCS #11 library expects that it's the only Redis consumer on the host, and it modifies some of the Redis configuration accordingly. Don't use the PKCS #11 library with Redis when you have other applications that use Redis on the same host.

**Topics**
Prerequisites

Before you can use the AWS CloudHSM software library for PKCS #11, 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 PKCS #11 library communicates locally with the client. If you haven't installed and configured the AWS CloudHSM client package, do that now by following the steps at Install and Configure the Client (p. 30). After you install and configure the client, use the following command to start it.

```
$ sudo start cloudhsm-client
```

Install the PKCS #11 Library

Complete the following steps to install or update the AWS CloudHSM software library for PKCS #11.

Use the following commands to download and install the PKCS #11 library.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-pkcs11-latest.x86_64.rpm
$ sudo yum install -y ./cloudhsm-client-pkcs11-latest.x86_64.rpm
```

Ubuntu

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-pkcs11_latest_amd64.deb
$ sudo dpkg -i cloudhsm-client-pkcs11_latest_amd64.deb
```

After you complete the preceding steps, you can find the PKCS #11 libraries in /opt/cloudhsm/lib.

To install (or update) the PKCS #11 library with Redis

1. Complete the preceding steps to install the PKCS #11 library.
2. Complete the following steps to enable the repository named Extra Packages for Enterprise Linux.
   a. Use a text editor to open the file /etc/yum.repos.d/epel.repo. This requires administrative permissions (sudo).
   b. In the [epel] configuration, ensure that enabled is set to 1, as shown in the following example.

```
[epel]
name=Extra Packages for Enterprise Linux 6 - $basearch
#baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch
```
c. Save the file, and then close it.

3. Use the following command to change your working directory to /opt/cloudhsm.

```bash
$ cd /opt/cloudhsm
```

4. Use the following command to install Redis and configure it for the PKCS #11 library.

```bash
$ sudo /opt/cloudhsm/bin/setup_redis
```

5. Use the following command to start the Redis service.

```bash
$ sudo service redis start
```

6. Use the following command to build the Redis cache, specifying the user name and password of a crypto user (CU) on the HSM.

```bash
$ /opt/cloudhsm/bin/build_keystore -s <CU user name> -p <password>
```

**Specify a PIN for PKCS #11**

The PKCS #11 interface defines a PIN (personal identification number) for users of a cryptographic token. To specify a PKCS #11 PIN in the context of the AWS CloudHSM software library for PKCS#11, use the following format.

```
<HSM_user_name>:<password>
```

For example, the following is the PKCS #11 PIN for an HSM crypto user (CU) (p. 11) with user name CryptoUser and password CUPassword123!.

```
CryptoUser:CUPassword123!
```

**AWS CloudHSM Software Library for OpenSSL**

The AWS CloudHSM software library for OpenSSL is an OpenSSL dynamic engine that supports the OpenSSL command line interface and EVP APIs. The software library allows applications that are integrated with OpenSSL, such as the Nginx and Apache web servers, to offload their cryptographic processing to the HSMs in your AWS CloudHSM cluster. The engine supports the following key types and ciphers:

- RSA key generation for 2048, 3072, and 4096-bit keys.
- RSA sign/verify.
- RSA encrypt/decrypt.
- Random number generation that is cryptographically secure and FIPS-validated.

For more information, see the following topic.
Install and Use the AWS CloudHSM Software Library for OpenSSL

Before you can use the AWS CloudHSM software library 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 library communicates locally with the client. If you haven’t installed and configured the AWS CloudHSM client package, do that now by following the steps at Install and Configure the Client (p. 30). After you install and configure the client, use the following command to start it.

$ sudo start cloudhsm-client

Install and Configure the OpenSSL Library

Complete the following steps to install (or update) and configure the AWS CloudHSM software library for OpenSSL.

To install (or update) and configure the OpenSSL library

1. Use the following command to download the OpenSSL library.

    Amazon Linux

    $ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-dyn-latest.x86_64.rpm

    Ubuntu

    $ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-dyn_latest_amd64.deb

2. Use the following command to install the OpenSSL library.

    Amazon Linux

    $ sudo yum install -y ./cloudhsm-client-dyn-latest.x86_64.rpm

    Ubuntu

    $ sudo dpkg -i cloudhsm-client-dyn_latest_amd64.deb

After you complete the preceding step, you can find the OpenSSL library at /opt/cloudhsm/lib/libcloudhsm_openssl.so.
3. Use the following command to set an environment variable named `n3fips_password` that contains the credentials of a crypto user (CU).

```
$ export n3fips_password=<HSM user name>:<password>
```

Use the OpenSSL Library

To use the AWS CloudHSM software library 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
```

To use the AWS CloudHSM software library for OpenSSL from an OpenSSL-integrated application, ensure that your application uses the OpenSSL dynamic engine named `cloudhsm`. The shared library for the dynamic engine is located at `/opt/cloudhsm/lib/libcloudhsm_openssl.so`.

AWS CloudHSM Software Library for Java

The AWS CloudHSM software library for Java is a provider implementation for the Sun Java JCE (Java Cryptography Extension) provider framework. It includes implementations for interfaces and engine classes in the JCA (Java Cryptography Architecture) standard. For more information about the supported provider classes and interfaces, see the following topics.

Topics

- Supported Keys (p. 162)
- Supported Ciphers (p. 163)
- Supported Digests (p. 164)
- Supported Hash-Based Message Authentication Code (HMAC) Algorithms (p. 164)
- Supported Sign/Verify Mechanisms (p. 164)
- Install and Use the AWS CloudHSM Software Library for Java (p. 165)
- Example Code for the AWS CloudHSM Software Library for Java (p. 168)

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.
- **EC** key pairs for NIST curves P256 and P384.

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. (Imported keys are never extractable.)
- **isPersistent**: Indicates whether the key remains on the HSM when the current session ends.
## Supported Ciphers

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mode</th>
<th>Padding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>CBC</td>
<td>AES/CBC/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE, Cipher.DECRYPT_MODE, Cipher.WRAP_MODE, Cipher.UNWRAP_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AES/CBC/PKCS5Padding</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>GCM</td>
<td>AES/GCM/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When performing AES-GCM encryption, the HSM ignores the initialization vector (IV) in the request and uses an IV that it generates. When the operation completes, you must call Cipher.getIV() to get the IV.</td>
</tr>
<tr>
<td>DESede (Triple DES)</td>
<td>CBC</td>
<td>DESede/CBC/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESede/CBC/PKCS5Padding</td>
<td>The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.</td>
</tr>
<tr>
<td>RSA</td>
<td>ECB</td>
<td>RSA/ECB/NoPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/PKCS1Padding</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>ECB</td>
<td>RSA/ECB/OAEPPadding</td>
<td>Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/OAEPWithSHA-1ANDMGFPadding</td>
<td>OAEPPadding is OAEP with the SHA-1 padding type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSA/ECB/PKCS1Padding</td>
<td></td>
</tr>
</tbody>
</table>
## Supported Digests

The AWS CloudHSM software library for Java supports the following message digests.

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

## 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mode</th>
<th>Padding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA/ECB/</td>
<td></td>
<td>RSA/ECB/OAEPEncWithSHA-224andMGF1Padding</td>
<td></td>
</tr>
<tr>
<td>RSA/ECB/</td>
<td></td>
<td>RSA/ECB/OAEPEncWithSHA-256andMGF1Padding</td>
<td></td>
</tr>
<tr>
<td>RSA/ECB/</td>
<td></td>
<td>RSA/ECB/OAEPEncWithSHA-384andMGF1Padding</td>
<td></td>
</tr>
<tr>
<td>RSA/ECB/</td>
<td></td>
<td>RSA/ECB/OAEPEncWithSHA-512andMGF1Padding</td>
<td></td>
</tr>
</tbody>
</table>
ECDSA Signature Types

- NONEwithECDSA
- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA
- SHA384withECDSA
- SHA512withECDSA

Install and Use the AWS CloudHSM Software Library for Java

Before you can use the AWS CloudHSM software library for Java, you need the AWS CloudHSM client. The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster, and the Java library communicates locally with the client. If you haven't installed and configured the AWS CloudHSM client package, do that now by following the steps at Install and Configure the Client (p. 30). After you install and configure the client, use the following command to start it.

```
$ sudo start cloudhsm-client
```

Topics

- Installing the Java Library (p. 165)
- Testing the Java Library (p. 166)
- Providing Credentials to the Java Library (p. 167)
- Key Management Basics in the Java Library (p. 168)

Installing the Java Library

Complete the following steps to install or update the AWS CloudHSM software library for Java.

Use the following commands to download and install the Java library.

**Amazon Linux**

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-jce-latest.x86_64.rpm

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

**Ubuntu**

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-jce_latest_amd64.deb
```
$ sudo dpkg -i cloudhsm-client-jce_latest_amd64.deb

After you complete the preceding steps, you can find the following Java library files:

- /opt/cloudhsm/java/cloudhsm-1.0.jar
- /opt/cloudhsm/java/cloudhsm-test-1.0.jar
- /opt/cloudhsm/java/hamcrest-all-1.3.jar
- /opt/cloudhsm/java/junit.jar
- /opt/cloudhsm/java/log4j-api-2.8.jar
- /opt/cloudhsm/java/log4j-core-2.8.jar
- /opt/cloudhsm/lib/libcaviumjca.so

Testing the Java Library

To test that the AWS CloudHSM software library for Java works with the HSMs in your cluster, complete the following steps.

To test the Java library with your cluster

1. (Optional) If you don't already have Java installed in your environment, use the following command to install it.

   Amazon Linux

   $ sudo yum install -y 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 RSA test.

   $ java8 -cp "/usr/share/java/junit4.jar:/opt/cloudhsm/java/*" \
          org.junit.runner.JUnitCore \
          com.cavium.unittest.TestRSA

   To run a different test, replace TestRSA in the preceding command with one of the following values:

   - TestAESKeyGen
Providing Credentials to the Java Library

Your Java application must be authenticated by the HSMs in your cluster before it can use them. Each application can use one session, which is established by providing credentials in one of the following ways. In the following examples, replace `<HSM user name>` and `<password>` with the credentials of a crypto user (CU).

The first of the following examples shows how to use the `LoginManager` class to manage sessions in your code. Instead, you can let the library implicitly manage sessions when your application starts, as shown in the remaining examples. However in these latter cases it might be difficult to understand error conditions when the provided credentials are invalid or the HSMs are having problems. When you use the `LoginManager` class, you have more control over how your application deals with failures.

- Use the `LoginManager` class to provide credentials directly in your application. For example:

```
LoginManager lm = LoginManager.getInstance();
lm.loadNative();
lm.login("PARTITION_1", "<HSM user name>", "<password>");
```

- Provide a file named `HsmCredentials.properties` in your application's `CLASSPATH`. The file's contents should look like the following:

```
HSM_PARTITION = PARTITION_1
HSM_USER = <HSM user name>
HSM_PASSWORD = <password>
```

- Provide Java 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>");
```

- Set system environment variables. For example:

```
$ export HSM_PARTITION=PARTITION_1
```
Key Management Basics in the Java Library

The following key management basics can help you get started with the AWS CloudHSM software library for Java.

**To import a key implicitly**

To implicitly import a key, pass the key to any API that accepts one. If the key is the correct type for the specified operation, the HSMs automatically import and use the provided key.

**To import a key explicitly**

Use the utility class named `ImportKey` to import a key and set its attributes.

**To make a session key persist**

Use the `Util.persistKey()` method to make a session key into a token key—that is, to persist the key in the HSMs.

**To delete a key**

Use the `Util.deleteKey()` method to delete a key.

---

Example Code for the AWS CloudHSM Software Library for Java

**Example code only - Not for production use**

This page includes example code that has not been fully tested. It is designed for test environments. Do not run this code in production.

The following Java code examples show you how to use the AWS CloudHSM software library for Java (p. 162) to perform basic tasks in AWS CloudHSM.

Before running the examples, set up your environment:

- Install and configure the AWS CloudHSM software library for Java (p. 165) and the AWS CloudHSM client package (p. 30).
- Set up a valid HSM user name and password (p. 134). Crypto user (CU) permissions are sufficient for these tasks. Your application uses these credentials to log in to the HSM in each example. The examples use the `loginWithExplicitCredentials()` method (p. 169) to log in to an HSM, but you can use the method that you prefer.
- Decide how to specify the Cavium provider (p. 169).

**Topics**

- Specifying the Cavium Provider (p. 169)
- Logging Into and Out of an HSM (p. 169)
Specifying the Cavium Provider

The examples that follow use the Cavium provider in the AWS CloudHSM client package. To specify the Cavium provider, use either of the following techniques:

- Create an instance of the Cavium provider and pass it to the methods that take a provider, such as this `KeyGenerator.getInstance()` method.

```java
CaviumProvider cp = new CaviumProvider();
keyGen = KeyGenerator.getInstance("AES", cp);
```

- Add the Cavium provider to the `$JAVA_HOME/jre/lib/security/java.security` file. Then use the `Cavium` string to refer to the provider. If you do not specify a provider, Java uses the first provider in the file, but it's best to specify the provider explicitly.

```java
//Add the Cavium provider to the Java provider file
Security.addProvider(new CaviumProvider());
//Or, add the provider to the first position in the file
Security.insertProviderAt(new CaviumProvider(), 1);

//Then, you can use "Cavium" to specify the provider.
keyGen = KeyGenerator.getInstance("AES","Cavium");
```

Logging Into and Out of an HSM

This example demonstrates three ways for your Java application to log in to the HSMs in your cluster. These methods all use the `LoginManager` class to manage sessions in the code. Each provides credentials in a different way.

The remaining examples in this section use the `loginWithExplicitCredentials()` method to log in to an HSM, but you can change them to use the method that you prefer.

**Note**

To run this example, you must replace `CryptoUser` and `CUPassword123!` with a valid AWS CloudHSM user name and password. Also, the `loginWithEnvVariables()` method fails unless you have set the HSM environment variables in advance.

For more details about providing credentials to the Java library, see Providing Credentials to the Java Library (p. 167)

**Example code only - Not for production use**

```java
package com.amazonaws.cloudhsm.examples;
```
import com.cavium.cfm2.CFM2Exception;
import com.cavium.cfm2.LoginManager;

public class LoginLogoutExample {
    public static void main(String[] args) {
        System.out.println("Test three methods of logging into the HSMs in your cluster");
        System.out.println("*********** Logging in using hard-coded credentials***********");
        loginWithExplicitCredentials();
        System.out.println("Logging out");
        logout();
        System.out.println("*********** Logging in using Java system properties***********");
        loginUsingJavaProperties();
        System.out.println("Logging out");
        logout();
        System.out.println("*********** Logging in using environment variables************");
        loginWithEnvVariables();
        System.out.println("Logging out");
        logout();
    }

    /**
     * Method #1: Use hard-coded credentials
     * Replace "CryptoUser" and "CUPassword123!" with a valid user name and password.
     */
    public static void loginWithExplicitCredentials() {
        LoginManager lm = LoginManager.getInstance();
        lm.loadNative();
        try {
            lm.login("PARTITION_1", "CryptoUser", "CUPassword123!");
            int appID = lm.getAppid();
            System.out.println("App ID = " + appID);
            int sessionID = lm.getSessionid();
            System.out.println("Session ID = " + sessionID);
        } catch (CFM2Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Method #2: Use Java system properties
     * Replace "CryptoUser" and "CUPassword123!" with a valid user name and password.
     */
    public static void loginUsingJavaProperties() {
        System.setProperty("HSM_PARTITION","PARTITION_1");
        System.setProperty("HSM_USER","CryptoUser");
        System.setProperty("HSM_PASSWORD","CUPassword123!");
        LoginManager lm = LoginManager.getInstance();
        lm.loadNative();
        try {
            lm.login();
            int appID = lm.getAppid();
            System.out.println("App ID = " + appID);
            int sessionID = lm.getSessionid();
            System.out.println("Session ID = " + sessionID);
        } catch (CFM2Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Method #3: Use environment variables
     */
    public static void loginWithEnvVariables() {
        System.setProperty("HSM_PARTITION","PARTITION_1");
        System.setProperty("HSM_USER","CryptoUser");
        System.setProperty("HSM_PASSWORD","CUPassword123!");
        LoginManager lm = LoginManager.getInstance();
        lm.loadNative();
        try {
            lm.login();
            int appID = lm.getAppid();
            System.out.println("App ID = " + appID);
            int sessionID = lm.getSessionid();
            System.out.println("Session ID = " + sessionID);
        } catch (CFM2Exception e) {
            e.printStackTrace();
        }
    }
}
Before invoking the JVM, set the following environment variables using a valid user name and password:

- `export HSM_PARTITION=PARTITION_1`
- `export HSM_USER=<hsm-user-name>`
- `export HSM_PASSWORD=<password>`

```java
public static void loginWithEnvVariables() {
    LoginManager lm = LoginManager.getInstance();
    lm.loadNative();
    try {
        lm.login();
        int appID = lm.getAppid();
        System.out.println("App ID = " + appID);
        int sessionID = lm.getSessionid();
        System.out.println("Session ID = " + sessionID);
    } catch (CFM2Exception e) {
        e.printStackTrace();
    }
}
```

```java
public static void logout() {
    try {
        LoginManager.getInstance().logout();
    } catch (CFM2Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

**Generating a Symmetric Key**

This example shows how to generate a 256-bit Advanced Encryption Standard (AES) symmetric key and save it in an HSM. By default, the keys that the HSM generates are not saved in the HSM ("persistent"). To make a key persistent, that is, to convert a session key into a token key, call the `Util.persistKey()` method.

This example does not return any output, but you can save the key object and use the key handle in other operations.

This example uses the `loginWithExplicitCredentials()` method of the `LoginLogoutExample` (p. 169) class to log in to the HSM. You can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider, and substitute it for the Cavium string.

**Example code only - Not for production use**

```java
package com.amazonaws.cloudhsm.examples;
import java.io.IOException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
```
import com.cavium.cfm2.CFM2Exception;
import com.cavium.cfm2.Util;
import com.cavium.key.CaviumAESKey;
import com.cavium.key.CaviumKey;
import com.cavium.provider.CaviumProvider;

public class SymmetricKeyGeneration {
    // Generate a 256-bit AES symmetric key and save it in the HSM
    public static void main(String[] args) {
        LoginLogoutExample.loginWithExplicitCredentials();
        new SymmetricKeyGeneration().generateAESKey(256, true);
        LoginLogoutExample.logout();
    }

    public Key generateAESKey(int keySize, boolean isPersistent) {
        KeyGenerator keyGen;
        try {
            keyGen = KeyGenerator.getInstance("AES", "Cavium");
            keyGen.init(keySize);
            SecretKey aesKey = keyGen.generateKey();
            System.out.println("Generated the AES key");
            if(aesKey instanceof CaviumAESKey) {
                System.out.println("Key is of type CaviumAESKey");
                CaviumAESKey ck = (CaviumAESKey) aesKey;
                //Save the key handle. You'll need it to encrypt/decrypt in the future.
                System.out.println("Key handle = " + ck.getHandle());
                //Get the key label. The SDK generates this label for the key.
                System.out.println("Key label = " + ck.getLabel());
                System.out.println("Is the key extractable? : " + ck.isExtractable());
                //By default, keys are not saved in the HSM.
                System.out.println("Is the key persistent? : " + ck.isPersistent());
                // Save the key in the HSM, if requested
                if(isPersistent){
                    System.out.println("Make the key persistent:");
                    makeKeyPersistent(ck);
                }
                System.out.println("Is key persistent? : " + ck.isPersistent());
                //Verify key type and size
                System.out.println("Key algorithm : " + ck.getAlgorithm());
                System.out.println("Key size : " + ck.getSize());
            }
            // Return the key
            return aesKey;
        } catch (NoSuchAlgorithmException e) { // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NoSuchProviderException e) { // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    public static void makeKeyPersistent(Key key) {
        CaviumAESKey caviumAESKey = (CaviumAESKey) key;
        try {
            Util.persistKey(caviumAESKey);
            System.out.println("Added Key to HSM");
        } catch (CFM2Exception e) { // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
Encrypting and Decrypting with a Symmetric Key

This example shows how to encrypt and decrypt a string using a 256-bit Advanced Encryption Standard (AES) symmetric key.

The example uses the AES algorithm with Galois Counter Mode (GCM), which uses authenticated encryption with associated data (AEAD). The code specifies an additional authenticated data (AAD) string and an initialization vector (IV), which is an arbitrary number, like a nonce. The encryption operation changes the AAD and IV, so you need to save the new AAD and IV and use them to decrypt the ciphertext. The encrypt() method in this example returns an object (FinalResult) that includes the ciphertext, the IV (and its length) and the AAD (and its length).

To generate the symmetric key, this example calls the generateAESKey() method of the SymmetricKeyGeneration class in the section called “Generating a Symmetric Key” (p. 171) example. It uses the loginWithExplicitCredentials() method in the LoginLogoutExample (p. 169) class to log in to the HSM, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider and substitute it for the Cavium string.

** Example code only - Not for production use **

```java
package com.amazonaws.cloudhsm.examples.crypto.symmetric;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.GCMParameterSpec;
import com.amazonaws.cloudhsm.examples.key.symmetric.SymmetricKeyGeneration;
import com.amazonaws.cloudhsm.examples.operations.LoginLogoutExample;
import com.cavium.key.CaviumAESKey;

public class SymmetricEncryptDecryptExample {
    String plainText = "This is a sample plaintext message";
    String aad = "AAD data";
    String transformation = "AES/GCM/NoPadding";
    int ivSizeInBytes=12;
    int tagLengthInBytes = 16;
    /*
     * AEAD modes, such as GCM and CCM, authenticate the AAD before authenticating the ciphertext.
     * To avoid buffering the ciphertext internally, supply all AAD data to GCM/CCM implementations
     */
```
public static void main(String[] args) {
    SymmetricEncryptDecryptExample obj = new SymmetricEncryptDecryptExample();
    LoginLogoutExample.loginWithExplicitCredentials();
    // Generate a 256-bit AES key and save it in the HSM
    Key key = new SymmetricKeyGeneration().generateAESKey(256, true);
    // Generate an initialization vector (IV)
    byte[] iv = obj.generateIV(obj.ivSizeInBytes);
    System.out.println("Performing AES encryption operation");
    // Encrypt the plaintext with the specified algorithm, key, IV, and the AAD
    byte[] result = obj.encrypt(obj.transformation, (CaviumAESKey) key, obj.plainText, iv, obj.aad, obj.tagLengthInBytes);
    System.out.println("Plaintext is encrypted");
    System.out.println("Base64-encoded encrypted text = " + Base64.getEncoder().encodeToString(result));
    System.out.println("Decrypting the ciphertext");
    // Extract the IV for the decrypt operation
    iv = Arrays.copyOfRange(result, 0, 16);
    byte[] cipherText = Arrays.copyOfRange(result, 16, result.length);
    // Decrypt the ciphertext using the algorithm, key, IV, and AAD
    byte[] decryptedText = obj.decrypt(obj.transformation, (CaviumAESKey) key, cipherText, iv, obj.aad, obj.tagLengthInBytes);
    System.out.println("Plaintext = " + new String(decryptedText));
    LoginLogoutExample.logout();
}

// This encrypt operation uses an initialization vector (IV) and additional authenticated data (AAD)
public byte[] encrypt(String transformation, CaviumAESKey key, String plainText, byte[] iv, String aad, int tagLength) {
    try {
        // Create an encryption cipher
        Cipher encCipher = Cipher.getInstance(transformation, "Cavium");
        // Create a parameter spec
        GCMParameterSpec gcmSpec = new GCMParameterSpec(tagLengthInBytes * 8, iv);
        // Configure the encryption cipher
        encCipher.init(Cipher.ENCRYPT_MODE, key, gcmSpec);
        encCipher.updateAAD(aad.getBytes());
        encCipher.update(plainText.getBytes());
        // Encrypt the plaintext data
        byte[] ciphertext = encCipher.doFinal();

        // Save the new IV and AADTag from the HSM.
        // You'll need them to create the GCMParameterSpec for the decrypt operation.
        byte[] finalResult = new byte[encCipher.getIV().length + ciphertext.length];
        System.arraycopy(encCipher.getIV(), 0, finalResult, 0, encCipher.getIV().length);
        System.arraycopy(ciphertext, 0, finalResult, encCipher.getIV().length, ciphertext.length);
        return finalResult;
    } catch (NoSuchAlgorithmException | NoSuchProviderException | NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidAlgorithmParameterException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
Generating an Asymmetric Key Pair

This example shows how to generate an RSA key pair and then save the public and private keys in the HSM. To make a key persistent, that is, to convert a session key into a token key, call the Util.persistKey() method.

The example does not return any output, but you can save the key pair object and use the public and private key handles in other operations.
To log in to the HSM, this example uses the `loginWithExplicitCredentials()` method of the `LoginLogoutExample` (p. 169) class, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider, and substitute it for the `Cavium` string.

** Example code only - Not for production use **

```java
package com.amazonaws.cloudhsm.examples;

import java.math.BigInteger;
import java.security.InvalidAlgorithmParameterException;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import com.cavium.cfm2.CFM2Exception;
import com.cavium.cfm2.Util;
import com.cavium.key.CaviumAESKey;
import com.cavium.key.CaviumKey;
import com.cavium.key.CaviumRSAPrivateKey;
import com.cavium.key.parameter.CaviumRSAKeyGenParameterSpec;

public class AsymmetricKeyGeneration {

    // Generate a 2048-bit RSA key pair and save it in the HSM
    public static void main(String[] args) {
        LoginLogoutExample.loginWithExplicitCredentials();
        new AsymmetricKeyGeneration().generateRSAKeyPair(2048, true);
        LoginLogoutExample.logout();
    }

    public KeyPair generateRSAKeyPair(int keySize, boolean isPersistent) {
        KeyPairGenerator keyPairGen;
        try {
            // Create and configure a key pair generator
            keyPairGen = KeyPairGenerator.getInstance("rsa", "Cavium");
            keyPairGen.initialize(new CaviumRSAKeyGenParameterSpec(keySize, new
            BigInteger("65537")));
            // Generate the key pair
            KeyPair kp = keyPairGen.generateKeyPair();

            if (kp == null) {
                System.out.println("Failed to generate key pair");
            }
            RSAPrivateKey privKey = (RSAPrivateKey) kp.getPrivate();
            RSAPublicKey pubKey = (RSAPublicKey) kp.getPublic();
            System.out.println("Generated RSA key pair");
            // Write out properties of RSA private key
            if (privKey instanceof CaviumRSAPrivateKey) {
                CaviumRSAPrivateKey cavRSAKey = (CaviumRSAPrivateKey) privKey;
                System.out.println("Private key handle = " + cavRSAKey.getHandle());
                System.out.println("Private key label = " + cavRSAKey.getLabel());
                System.out.println("Is private key extractable = " +
                cavRSAKey.isExtractable());
                System.out.println("Is private key persistent = " +
                cavRSAKey.isPersistent());
            }
        } catch (CFM2Exception e) {
            System.out.println("CFM2 Exception: " + e.getMessage());
        } catch (NoSuchAlgorithmException e) {
            System.out.println("NoSuchAlgorithmException: " + e.getMessage());
        } catch (NoSuchProviderException e) {
            System.out.println("NoSuchProviderException: " + e.getMessage());
        } catch (InvalidAlgorithmParameterException e) {
            System.out.println("InvalidAlgorithmParameterException: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```
Encrypting and Decrypting with an Asymmetric Key Pair

This example shows how to encrypt and decrypt a string using a 2048-bit RSA key pair. First it encrypts with the public key and decrypts with the private key. Then it encrypts with the private key and decrypts with the public key.

To generate the key pair, this example calls the generateRSAKeyPair() method of the AsymmetricKeyGeneration class in the the section called “Generating an Asymmetric Key Pair” (p. 175) example. It uses the loginWithExplicitCredentials() method in the LoginLogoutExample (p. 169) class to log in to the HSM, but you can substitute the login method
that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider and substitute it for the Cavium string.

** Example code only - Not for production use **

```java
package com.amazonaws.cloudhsm.examples;

import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import org.bouncycastle.util.encoders.Base64;
import com.cavium.key.CaviumRSAPrivateKey;
import com.cavium.key.CaviumRSAPublicKey;

public class AsymmetricEncryptDecryptExample {

    String plainText = "This is a plaintext string";
    // Specify the encryption algorithm
    String transformation = "RSA/ECB/OAEPWithSHA-224ANDMGF1Padding";

    public static void main(String[] args) {
        // Log into the HSM
        LoginLogoutExample.loginWithExplicitCredentials();
        // Generate a 2048-bit RSA key pair and save it in the HSM
        KeyPair kp = new AsymmetricKeyGeneration().generateRSAKeyPair(2048, true);
        // Create an example object
        AsymmetricEncryptDecryptExample obj = new AsymmetricEncryptDecryptExample();
        //Get the private key
        CaviumRSAPrivateKey privKey = (CaviumRSAPrivateKey) (RSAPrivateKey) kp.getPrivate();
        //Get the public key
        CaviumRSAPublicKey pubKey = (CaviumRSAPublicKey) (RSAPublicKey) kp.getPublic();
        System.out.println("Use the private key to encrypt; use the public key to decrypt");
        // Encrypt the plaintext with the private key
        byte[] cipherText = obj.asymmetricKeyEncryption(obj.transformation, privKey, obj.plainText);
        System.out.println("CipherText = " + Base64.toBase64String(cipherText));
        // Decrypt the ciphertext with the public key
        String plainText = obj.asymmetricKeyDecryption(obj.transformation, pubKey, cipherText);
        System.out.println("PlainText = " + plainText);
        System.out.println("Encrypt with public key; decrypt with private key");
        // Encrypt with the public key
        cipherText = obj.asymmetricKeyEncryption(obj.transformation, pubKey, obj.plainText);
        System.out.println("CipherText = " + Base64.toBase64String(cipherText));
        // Decrypt with private key
        plainText = obj.asymmetricKeyDecryption(obj.transformation, privKey, cipherText);
        System.out.println("PlainText = " + plainText);
        LoginLogoutExample.logout();
    }
}
```
// Encrypt with the specified algorithm and key
public byte[] asymmetricKeyEncryption(String transformation, Key key, String plainText) {
    try {
        Cipher cipher = Cipher.getInstance(transformation, "Cavium");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        cipher.update(plainText.getBytes()); // Encrypt the plaintext
        byte[] cipherText = cipher.doFinal(plainText.getBytes());
        return cipherText;
    } catch (NoSuchAlgorithmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchProviderException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchPaddingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (BadPaddingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}

// Decrypt with the specified algorithm and key
public String asymmetricKeyDecryption(String transformation, Key key, byte[] cipherText) {
    try {
        Cipher cipher = Cipher.getInstance(transformation, "Cavium");
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] plainText = cipher.doFinal(cipherText);
        return new String(plainText);
    } catch (NoSuchAlgorithmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchProviderException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchPaddingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IllegalBlockSizeException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (BadPaddingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}
Signing a Message

This example shows how to sign a message with a key in an HSM. The example generates a 4096-bit asymmetric key pair. It uses the private key to sign the message. Then it uses the public key to verify the message signature.

To generate the key pair, this example calls the `generateRSAKeyPair()` method of the `AsymmetricKeyGeneration` class in the section called “Generating an Asymmetric Key Pair” (p. 175) example. It uses the `loginWithExplicitCredentials()` method in the `LoginLogoutExample` (p. 169) class to log in to the HSM, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider, and substitute it for the `Cavium` string.

** Example code only - Not for production use **

```java
package com.amazonaws.cloudhsm.examples;

import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyPair;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Signature;
import java.security.SignatureException;
import java.util.Base64;
import com.cavium.key.CaviumRSAPrivateKey;
import com.cavium.key.CaviumRSAPublicKey;

public class SignatureExample {
    String sampleMessage = "This is a sample message.";
    String signingAlgorithm = "SHA512withRSA/PSS";

    public static void main(String[] args) {
        LoginLogoutExample.loginUsingJavaProperties();
        SignatureExample obj = new SignatureExample();

        //Generate a 4096-bit pair and save it in the HSM
        KeyPair kp = new AsymmetricKeyGeneration().generateRSAKeyPair(4096, true);
        System.out.println("Generated key pair");

        //Sign the message with the private key and the specified signing algorithm
        byte[] signature = obj.signMessage(obj.sampleMessage, obj.signingAlgorithm,
                                            (CaviumRSAPrivateKey)kp.getPrivate());
        System.out.println("Signature : " + Base64.getEncoder().encodeToString(signature));

        //Verify the signature
        boolean isVerificationSuccessful = obj.verifySign(obj.sampleMessage,
                                                    obj.signingAlgo, (CaviumRSAPublicKey)kp.getPublic(), signature);
        System.out.println("Verification result : " + isVerificationSuccessful);
    }

    //Use the private key to sign the message
    public byte[] signMessage(String message, String signingAlgorithm, CaviumRSAPrivateKey privateKey) {
        try {
            Signature sig = Signature.getInstance(signingAlgorithm, "Cavium");
            sig.initSign(privateKey);
            sig.update(message.getBytes());
            byte[] signature = null;
            signature = sig.sign();
        }
```
return signature;
}

public boolean verifySign(String message, String signingAlgorithm, CaviumRSAPublicKey publicKey, byte[] signature) {
    try {
        Signature sig = Signature.getInstance(signingAlgorithm, "Cavium");
        sig.initVerify(publicKey);
        sig.update(message.getBytes());

        boolean isVerificationSuccessful = sig.verify(signature);
        return isVerificationSuccessful;
    } catch (NoSuchAlgorithmException | NoSuchProviderException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SignatureException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return false;
}

Generating a Hash

This example shows how to generate a hash of a message using an HSM and the SHA-512 hash algorithm.

To log in to the HSM, this example uses the loginWithExplicitCredentials() method of the LoginLogoutExample (p. 169) class, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider and substitute it for the Cavium string.

** Example code only - Not for production use **

```java
package com.amazonaws.cloudhsm.examples;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import javax.xml.bind.DatatypeConverter;
public class HashExample {
    String plainText = "This is a sample plaintext message.";
    String hashAlgorithm = "SHA-512";
```
Generating an HMAC

This example shows how to generate a hash-based message authentication code (HMAC) in the HSM and use it to hash a message. Unlike a typical hash, an HMAC uses a hash function and a cryptographic key.

To generate the key pair, this example calls the `generateRSAKeyPair()` method of the `AsymmetricKeyGeneration` class in the section called “Generating an Asymmetric Key Pair” (p. 175) example. It uses the `loginWithExplicitCredentials()` method in the `LoginLogoutExample` (p. 169) class to log in to the HSM, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider and substitute it for the `Cavium` string.

** Example code only - Not for production use **
Managing Keys in an HSM

This example shows how to manage keys in an HSM. It demonstrates the following operations:

- **Get** a reference to a key in the HSM.
- **Export** a key from the HSM. This operation returns the key, not just a reference, so you can import the key into a different HSM and use it in other operations. It does not delete the key from the HSM.
- **Delete** a key from the HSM.
- **Import** a key into the HSM. This example returns a key handle that you can use to identify the key in other operations.

**Note**
To use a key in an encryption operation, just specify the key handle. You do not need to get or export the key.

To log into the HSM, this example uses the `loginWithExplicitCredentials()` method of the `LoginLogoutExample` (p. 169) class.

To generate the key pair, this example calls the `generateRSAKeyPair()` method of the `AsymmetricKeyGeneration` class in the section called “Generating an Asymmetric Key Pair” (p. 175) example. It uses the `loginWithExplicitCredentials()` method in the `LoginLogoutExample` (p. 169) class to log in to the HSM, but you can substitute the login method that you prefer. Also, the example assumes that the Cavium provider (p. 169) is included in your Java provider file. If it is not, create an instance of the provider and substitute it for the Cavium string.

You can also use the `key_mgmt_util` command line tool to manage keys in AWS CloudHSM (p. 137).

**Example code only - Not for production use**

```java
package com.amazonaws.cloudhsm.examples;
```
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import java.util.Vector;
import javax.crypto.BadPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.KeyGenerator;
import org.bouncycastle.util.Arrays;
import com.cavium.cfm2.CFM2Exception;
import com.cavium.cfm2.ImportKey;
import com.cavium.cfm2.Util;
import com.cavium.key.CaviumAESKey;
import com.cavium.key.CaviumKey;
import com.cavium.key.CaviumKeyAttributes;
import com.cavium.key.CaviumRSAPrivateKey;
import com.cavium.key.CaviumRSAPublicKey;
import com.cavium.key.parameter.CaviumKeyGenAlgorithmParameterSpec;

public class KeyManagement {

    public static void main(String[] args) {
        LoginLogoutExample.loginWithExplicitCredentials();
        // Get a reference to a key in the HSM
        // Replace the placeholder with an actual key handle value
        long keyHandle = 262194;
        CaviumKey ck = getKey(keyHandle);

        // Delete the specified key from the HSM
        // Replace the placeholder with an actual key handle value
        deleteKey(51);
        Key key = exportKey(keyHandle);

        // Import a key
        // Generate a 256-bit AES symmetric key
        KeyGenerator kg = KeyGenerator.getInstance("AES");
        kg.init(256);
        Key keyToBeImported = kg.generateKey();
        // Import the key as extractable and persistent
        // You can use the key handle to identify the key in other operations
        long importedKeyHandle = importKey(keyToBeImported, "Test", true, true);
        System.out.println("Imported Key Handle : " + importedKeyHandle);
        LoginLogoutExample.logout();
    }

    // Gets an existing key from the HSM
    // The type of the object that is returned depends on the key type
    public static CaviumKey getKey(long handle) {
        try {
            byte[] keyAttribute = Util.getKeyAttributes(handle);
            CaviumKeyAttributes cka = new CaviumKeyAttributes(keyAttribute);
            if (cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_AES) {
                CaviumAESKey aesKey = new CaviumAESKey(handle, cka);
                return aesKey;
            }
        }
    }
}
else if(cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_RSA &&
ck.getKeyClass() == CaviumKeyAttributes.CLASS_PRIVATE_KEY) {
    CaviumRSAPrivateKey privKey = new CaviumRSAPrivateKey(handle, cka);
    return privKey;
}
else if(cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_RSA &&
ck.getKeyClass() == CaviumKeyAttributes.CLASS_PUBLIC_KEY) {
    CaviumRSAPublicKey pubKey = new CaviumRSAPublicKey(handle, cka);
    return pubKey;
}
} catch (CFM2Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
return null;

//Deletes an existing persisted key
public static void deleteKey(long handle) {
    CaviumKey ck = getKey(handle);
    try {
        Util.deleteKey(ck);
        System.out.println("Key Deleted!");
    } catch (CFM2Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

//Exports an existing persisted key
//The type of the object that is returned depends on the key type
public static Key exportKey(long handle) {
    try {
        byte[] encoded = Util.exportKey(handle);
        byte[] keyAttribute = Util.getKeyAttributes(handle);
        CaviumKeyAttributes cka = new CaviumKeyAttributes(keyAttribute);
        if(cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_AES) {
            Key aesKey = new SecretKeySpec(encoded, 0, encoded.length, "AES");
            return aesKey;
        } else if(cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_RSA &&
ck.getKeyClass() == CaviumKeyAttributes.CLASS_PRIVATE_KEY) {
            PrivateKey privateKey = KeyFactory.getInstance("RSA").generatePrivate(new PKCS8EncodedKeySpec(encoded));
            return privateKey;
        } else if(cka.getKeyType() == CaviumKeyAttributes.KEY_TYPE_RSA &&
ck.getKeyClass() == CaviumKeyAttributes.CLASS_PUBLIC_KEY) {
            PublicKey publicKey = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(encoded));
            return publicKey;
        }
        System.out.println(new String(encoded));
    } catch (BadPaddingException | CFM2Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidKeySpecException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}

//Imports a key explicitly
public static void importKey(Key key, String keyLabel, boolean isExtractable, boolean isPersistent) {

//Create a new key parameter spec to identify the key. Specify a label and Boolean values for extractable and persistent.
CaviumKeyGenAlgorithmParameterSpec spec = new CaviumKeyGenAlgorithmParameterSpec(keyLabel, isExtractable, isPersistent);
try {
    ImportKey.importKey(key, spec);
} catch (InvalidKeyException e) {
    e.printStackTrace();
}
Integrating Third-Party Applications with AWS CloudHSM

Some of the use cases (p. 2) for AWS CloudHSM involve integrating third-party software applications with the HSMs in your AWS CloudHSM cluster. By integrating third-party software with AWS CloudHSM, you can accomplish a variety of security related goals. The following topics describe how to accomplish some of these goals.

Topics
- Improve Your Web Server's Security with SSL/TLS Offload in AWS CloudHSM (p. 187)
- Oracle Database Transparent Data Encryption (TDE) with AWS CloudHSM (p. 201)

Improve Your Web Server's Security with SSL/TLS Offload in AWS CloudHSM

Web servers and their clients (web browsers) can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS). These protocols confirm the identity of the web server and establish a secure connection to send and receive webpages or other data over the internet. This is commonly known as HTTPS. The web server uses a public–private key pair and a 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 computation to the HSMs in your AWS CloudHSM cluster. This is sometimes known as SSL acceleration. This offloading reduces the computational burden on your web server and provides extra security by storing the server's private key in the HSMs.

The Nginx and Apache HTTP Server web server applications natively integrate with OpenSSL to support HTTPS. The AWS CloudHSM software library for OpenSSL (p. 160) provides an interface that enables the web server to use the HSMs in your cluster for cryptographic offloading and key storage. The AWS CloudHSM software library for OpenSSL is the bridge that connects the web server to your AWS CloudHSM cluster.

In this tutorial, you choose whether to use the Nginx or Apache web server. You install the web server on a Linux instance in Amazon EC2, and then optionally use Amazon EC2 to create a second web server and Elastic Load Balancing to create a load balancer. Using a load balancer can increase performance when both web servers are healthy, because the load is distributed across multiple servers. It can also provide redundancy and higher availability in case one web server fails.

To configure SSL/TLS offload for Nginx or Apache with AWS CloudHSM

1. Follow the steps in Step 1: Set Up the Prerequisites (p. 188) to prepare your environment.
2. Follow the steps in Step 2: Import or Generate a Private Key and Certificate (p. 189) to import or generate a private key and certificate.
3. Follow the steps in Step 3: Configure the Web Server (p. 192) to configure the Nginx or Apache web server and verify that SSL/TLS offload is working.
4. (Optional) Follow the steps in Step 4: Add a Load Balancer (p. 197) to create a second web server and a load balancer that distributes HTTPS requests to your web servers.
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 server's certificate is signed by one of the root certificates that the client trusts.
   b. Extracts the public key from the server's 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.

Web Server SSL/TLS Offload Step 1: Set Up the Prerequisites

To set up web server SSL/TLS offload with AWS CloudHSM, you need the following prerequisites:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Linux operating system with the following software installed:
Step 2: Import or Generate a Private Key and Certificate

To set up all of these prerequisites, complete the following steps.

To set up the prerequisites for web server SSL/TLS offload with AWS CloudHSM

1. Complete the steps in Getting Started: Create A Cluster (p. 14). After you complete these steps, you have an active cluster with one HSM. You also have an Amazon EC2 instance, known as a client instance, with the AWS CloudHSM client and command line tools installed and configured. You use this client instance as your web server.

2. Connect to the client instance (p. 30). On the client instance, do the following:
   a. Choose whether to install the Nginx or Apache web server application. Then complete one of the following steps:
      • To install Nginx, run the following command.
        ```bash
        $ sudo yum install -y nginx
        ```
      • To install Apache HTTP Server, run the following command.
        ```bash
        $ sudo yum install -y httpd24 mod24_ssl
        ```
   b. Install and configure the AWS CloudHSM software library for OpenSSL (p. 161).

3. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 33).

4. To create a CU on the HSMs, on the client instance, do the following:
   a. Start the AWS CloudHSM client (p. 43).
   b. Update the configuration file for the command line tool known as cloudhsm_mgmt_util (p. 43).
   c. Use cloudhsm_mgmt_util to create a CU. For more information, see Managing HSM Users (p. 134). Keep track of the CU's user name and password. You need them later when you generate or import the web server's HTTPS private key.

After you complete these steps, go to Step 2: Import or Generate a Private Key and Certificate (p. 189).

Web Server SSL/TLS Offload Step 2: Import or Generate a Private Key and Certificate

To enable HTTPS, your web server application (Nginx or Apache HTTP Server) needs a private key and a corresponding public key certificate. To use web server SSL/TLS offload with AWS CloudHSM, you must store the private key in the HSMs 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 corresponding certificate, you can generate a private key in the HSMs (p. 190). Then you can use the private key to create a certificate signing request (CSR), which is then signed to produce a certificate.

• If you already have a private key and corresponding certificate, you can import the private key into the HSMs (p. 191).
Regardless of which method you choose, you then export a *fake PEM private key* from the HSMs and save it to a file. The file doesn't contain the actual private key. It contains a reference to the private key that is stored on the HSMs. Your web server application uses the fake PEM private key file, along with the AWS CloudHSM software library for OpenSSL, to offload SSL or TLS processing to the HSMs in your cluster.

**Generate a Private Key and Certificate**

If you don't have a private key and a corresponding certificate to use for HTTPS on your web server, you can generate a private key on the HSMs. Then you use the private key to create a certificate signing request (CSR), which is then signed to produce a certificate.

**To generate a private key on the HSMs**

1. Connect to the client instance (p. 30) that you created previously (p. 188).
2. Run the following command to set an environment variable named `n3fips_password` that contains the user name and password of the crypto user (CU) that you created previously (p. 188). Replace `<CU user name>` with the CU's user name, and replace `<password>` with the CU's password.

   ```
   $ export n3fips_password=<CU user name>:<password>
   ```

3. Run the following command to use the AWS CloudHSM software library for OpenSSL to generate a private key on the HSMs. This command also exports the fake PEM private key and saves it in a file. Replace `<web_server_fake_PEM.key>` with the preferred file name for your exported fake PEM private key.

   ```
   $ openssl genrsa -engine cloudhsm -out <web_server_fake_PEM.key> 2048
   ```

**To create a CSR**

- Run the following command to use the AWS CloudHSM software library for OpenSSL to create a 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 preferred file name for your CSR.

  This is an interactive command. Respond to each instruction, providing information for each field of the CSR. This same information appears in the certificate after it's signed.

  ```
  $ openssl req -engine cloudhsm -new -key <web_server_fake_PEM.key> -out <web_server.csr>
  ```

To produce a certificate, this CSR must be signed. To sign the CSR, you typically use a certificate authority (CA). You give the CSR file (`<web_server.csr>`) to a CA. The CA signs the CSR to create a signed certificate, and then gives you the certificate. Your web server application uses the signed certificate for HTTPS.

As an alternative to using a CA, you can use the AWS CloudHSM software library for OpenSSL to create a self-signed certificate. When you use a self-signed certificate for HTTPS, your web server's clients (web browsers) typically don't trust the web server. You shouldn't use a self-signed certificate in production, but this kind of certificate might be adequate for testing.

**To create a self-signed certificate**

**Important**

The following example is a proof-of-concept demonstration only. For production systems, use a more secure method (such as a CA) to sign the CSR.
AWS CloudHSM User Guide
Step 2: Import or Generate a Private Key and Certificate

- Run the following command to use the AWS CloudHSM software library for OpenSSL to sign your CSR with your private key on the HSM. This creates a self-signed certificate. Replace the following values with your own:
  - `<web_server.csr>` – The name of the file that contains the CSR that you created previously.
  - `<web_server_fake_PEM.key>` – The name of the file that contains your fake PEM private key.
  - `<web_server.crt>` – The preferred file name for your web server certificate.

```bash
$ openssl x509 -engine cloudhsm -req -days 365 -in <web_server.csr> -signkey <web_server_fake_PEM.key> -out <web_server.crt>
```

After you complete these steps, you can configure your web server for SSL/TLS offload with AWS CloudHSM (p. 192).

**Import an Existing Private Key**

If you already have a private key and a corresponding certificate that you use for HTTPS on your web server, you can import the private key into the HSMs.

**To import a private key into the HSMs**

1. Connect to the client instance (p. 30) that you created previously (p. 188). If necessary, copy your existing private key and certificate to the instance.
2. Run the following command to start the AWS CloudHSM client.

```bash
$ sudo start cloudhsm-client
```

3. Run the following command to start the command line tool known as key_mgmt_util.

```bash
$ /opt/cloudhsm/bin/key_mgmt_util
```

4. Run the following command to log in to the HSM. Replace `<user name>` and `<password>` with the user name and password of the crypto user (CU) that you created previously (p. 188).

   **Command:**
   ```bash
   loginHSM -u CU -s <user name> -p <password>
   ```

5. Run the following commands to import your private key into the HSM.
   a. Run the following command to create a symmetric wrapping key that is valid only for the current session.

   ```bash
   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 6 and err state 0x00000000 : HSM Return: SUCCESS

   b. Run the following command to import your existing private key into the HSMs. Replace the following values with your own:

   ```bash
   Command: <web_server_existing.key>
   ```

   - The name of the file that contains your private key.
   - `<web_server_imported_key>` – The preferred label for your imported private key.
Step 3: Configure the Web Server

6. Run the following command to export the private key in fake PEM format and save it to a file. Replace the following values with your own:

   • `<private_key_handle>` – The 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>` – The preferred file name for your exported fake PEM private key.

   ```
   Command: getCaviumPrivKey -k <private_key_handle> -out <web_server_fake_PEM.key>
   ```

7. 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. 192).

Web Server SSL/TLS Offload Step 3: Configure the Web Server

To finish setting up your web server for SSL/TLS offload with AWS CloudHSM, complete the following steps:

1. Update your web server configuration (p. 192) to use the AWS CloudHSM software library for OpenSSL to enable HTTPS.
2. Add your web server to a security group (p. 195) that allows inbound HTTPS connections.
3. Verify that an HTTPS connection (p. 196) from a client gets the certificate whose private key is stored on the HSMs in your AWS CloudHSM cluster.

Update the Web Server Configuration

To update your web server configuration, complete the steps in one of the following procedures. Choose the procedure that corresponds to your web server software.

• Update the configuration for Nginx (p. 193)
Update the configuration for Apache HTTP Server (p. 194)

Update the web server configuration for Nginx

1. Connect to the client instance (p. 30) that you created previously (p. 188). This is the same instance where you installed Nginx.

2. Run the following command to create the required directories for the web server certificate and the fake PEM private key.

   ```bash
   $ sudo mkdir -p /etc/pki/nginx/private
   ```

3. Run the following command to copy your web server certificate to the required location. Replace `<web_server.crt>` with the name of your web server certificate.

   ```bash
   $ 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. You created this file previously (p. 189).

   ```bash
   $ sudo cp <web_server_fake_PEM.key> /etc/pki/nginx/private/server.key
   ```

5. Run the following command to change the ownership of these files so that the user named `nginx` can read them.

   ```bash
   $ sudo chown nginx /etc/pki/nginx/server.crt /etc/pki/nginx/private/server.key
   ```

6. Run the following command to make a backup copy of the file named `/etc/nginx/nginx.conf`.

   ```bash
   $ sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
   ```

7. Use a text editor to edit the file named `/etc/nginx/nginx.conf`. At the top of the file, add the following line:

   ```bash
   ssl_engine cloudhsm;
   ```

   Then uncomment the TLS section of the file so that it looks like the following:

   ```bash
   # Settings for a TLS enabled server.
   server {
       listen 443 ssl http2 default_server;
       listen [::]:443 ssl http2 default_server;
       server_name _;
       root /usr/share/nginx/html;
       ssl_certificate "/etc/pki/nginx/server.crt";
       ssl_certificate_key "/etc/pki/nginx/private/server.key";
       # It is *strongly* recommended to generate unique DH parameters
       # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
       ssl_dhparam "/etc/pki/nginx/dhparams.pem";
       ssl_session_cache shared:SSL:1m;
       ssl_session_timeout 10m;
       ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
       ssl_ciphers HIGH:SEED:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!RSAPSK:!aDH:!aECDH:!EDH-DSS-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA:!SRP;
       ssl_prefer_server_ciphers on;
   }
   ```
# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

location / {
}

error_page 404 /404.html;
  location = /40x.html {
}

error_page 500 502 503 504 /50x.html;
  location = /50x.html {
}

Save the file. This requires Linux root permissions.

8. Run the following command to make a backup copy of the file named /etc/sysconfig/nginx.

```
$ sudo cp /etc/sysconfig/nginx /etc/sysconfig/nginx.backup
```

9. Use a text editor to edit the file named /etc/sysconfig/nginx. Add the following line, specifying the user name and password of the crypto user (CU) that you created previously (p. 188). Replace <CU user name> with the CU's user name, and replace <password> with the CU's password.

```
export n3fips_password=<CU user name>:<password>
```

Then save the file. This requires Linux root permissions.

10. Run the following command to start the Nginx web server.

```
$ sudo service nginx start
```

11. Run the following command to configure your server to start the Nginx application when the server starts.

```
$ sudo chkconfig nginx on
```

Update the web server configuration for Apache HTTP Server

1. Connect to the client instance (p. 30) that you created previously (p. 188). This is the same instance where you installed Apache HTTP Server.

2. Run the following command to make a backup copy of the default certificate.

```
$ sudo cp /etc/pki/tls/certs/localhost.crt /etc/pki/tls/certs/localhost.crt.backup
```

3. Run the following command to make a backup copy of the default private key.

```
$ sudo cp /etc/pki/tls/private/localhost.key /etc/pki/tls/private/localhost.key.backup
```

4. Run the following command to copy your web server certificate to the required location. Replace <web_server.crt> with the name of your web server certificate.

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```
5. Run the following command to copy your fake PEM private key to the required location. Replace `<web_server_fake_PEM.key>` with the name of the file that contains your fake PEM private key. You created this file previously (p. 189).

```
$ sudo cp <web_server_fake_PEM.key> /etc/pki/tls/private/localhost.key
```

6. Run the following command to change the ownership of these files so that the user named apache can read them.

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/localhost.key
```

7. Run the following command to make a backup copy of the file named `/etc/httpd/conf.d/ssl.conf`.

```
$ sudo cp /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.conf.backup
```

8. Use a text editor to edit the file named `/etc/httpd/conf.d/ssl.conf`. Replace the line that starts with `SSLCryptoDevice` so that it looks like the following:

```
SSLCryptoDevice cloudhsm
```

Then save the file. This requires Linux root permissions.

9. Run the following command to make a backup copy of the file named `/etc/sysconfig/httpd`.

```
$ sudo cp /etc/sysconfig/httpd /etc/sysconfig/httpd.backup
```

10. Use a text editor to edit the file named `/etc/sysconfig/httpd`. Add the following line, specifying the user name and password of the crypto user (CU) that you created previously (p. 188). Replace `<CU user name>` with the CU's user name, and replace `<password>` with the CU's password.

```
export n3fips_password=<CU user name>:<password>
```

Then save the file. This requires Linux root permissions.

11. Run the following command to start the Apache HTTP Server.

```
$ sudo service httpd start
```

12. Run the following command to configure your server to start the Apache HTTP Server application when the server starts.

```
$ sudo chkconfig httpd on
```

**Add the Web Server to a Security Group**

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. Then assign this security group to your web server.

**To create a security group for HTTPS and assign it to your web server**

3. For **Create Security Group**, do the following:
   a. For **Security group name**, type a name for the security group that you are creating. For example, **Inbound HTTPS**.
   b. (Optional) For **Description**, type a description of the security group that you are creating. For example, **Allow inbound traffic on port 443**.
   c. For **VPC**, choose the VPC that contains your web server instance.
   d. Choose **Add Rule**.
   e. For **Type**, choose **HTTPS**.
4. Choose **Create**.
5. In the navigation pane, choose **Instances**.
6. Select the check box next to your web server instance. Then choose **Actions**, **Networking**, **Change Security Groups**.
7. Select the check box next to the security group that you created for HTTPS. Then choose **Assign Security Groups**.

**Verify That HTTPS Uses the Private Key in Your AWS CloudHSM Cluster**

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 such as **Mozilla Firefox** or **Google Chrome**, or with a tool such as **OpenSSL s_client**.

**To verify SSL/TLS offload with a web browser**

1. Use a web browser such as Mozilla Firefox or Google Chrome to connect to your web server using the web server's public DNS name or IP address. 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/`.
   
   **Note**
   You can use a DNS service such as 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 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 Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.
3. Ensure that the certificate is the one that you configured the web server to use (p. 192), whose private key is stored in the HSMs in your AWS CloudHSM cluster.

**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

   2. Ensure that the certificate is the one that you configured the web server to use (p. 192), whose private key is stored in the HSMs in your AWS CloudHSM cluster.
You now have a website that is secured with HTTPS, with the web server's private key stored in the HSMs 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 4: Add a Load Balancer (p. 197).

**Web Server SSL/TLS Offload Step 4: Add a Load Balancer with Elastic Load Balancing**

After you set up SSL/TLS offload with one web server (p. 187), you can optionally create a second (and third, and so on) web server and an Elastic Load Balancing load balancer that routes HTTPS traffic to the web servers. A load balancer can reduce the load on your web servers by balancing the traffic across two or more web 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. The load balancer routes traffic only to the healthy web server, which helps ensure that your website is always reachable.

To add a load balancer for your web servers, complete the following steps:

1. Create a subnet for the second web server (p. 197).
2. Create the second web server (p. 198).
3. Create the load balancer (p. 200).

**Create a Subnet for the Second Web Server**

Before you can create a second 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 an identifiable name such as CloudHSM public subnet 2.
   b. For VPC, choose the 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, 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.
Step 4: Add a Load Balancer

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, Create Image**.
4. In the **Create Image** dialog box, do the following:
   a. For **Image name**, type a name for the image that you are creating. For example, `web-server-image`.
   b. For **Image description**, type a description for the image that you are creating. For example, `web server with CloudHSM SSL/TLS offload`.
   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, Launch More Like This.
8. Choose Edit AMI.

[AMI Details]

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>). Then choose Next.
12. Select an instance type, and then choose Next: Configure Instance Details.
13. 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.
14. Change the storage settings as preferred. Then choose Next: Add Tags.
15. Add or edit tags as preferred. Then choose Next: Configure Security Group.
16. For Step 6: Configure Security Group, do the following:
   a. For Assign a security group, choose Select an existing security group.
   b. Select the check box next to the security group named cloudhsm-<cluster ID>-sg. AWS CloudHSM created this security group on your behalf when you created the cluster (p. 17). You must choose this security group to allow the web server instance to connect to the HSMs in the cluster.
   c. Select the check box next to the security group that allows inbound HTTPS traffic. You created this security group previously (p. 195).
   d. (Optional) Select the check box next to a security group that allows inbound SSH traffic from your network. That is, the security group must allow inbound TCP traffic on port 22. Otherwise, you cannot connect to your client instance with SSH. If you don't have a security group like this, you must create one and then assign it to your client instance later. For more information about creating a security group, see Create a Security Group in the Amazon VPC Getting Started Guide.
   e. Then choose Review and Launch.
17. Review your instance details, and then choose Launch.
18. Choose whether to launch your instance with an existing key pair, create a new key pair, or launch your instance without a key pair.
   a. 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.
   b. To create a new key pair, do the following:
1. Choose Create a new key pair.
2. For Key pair name, type an identifiable key pair name such as CloudHSM client instance key pair.
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.

Then 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. For example, web-server-load-balancer.
   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. For example, web-server-target-group.
   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 such as Mozilla Firefox or Google Chrome, or with 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://. For example, https://web-server-load-balancer-89f9789aa91a4ee7.elb.us-east-2.amazonaws.com/.

   **Note**
   You can use a DNS service such as Route 53 to route your website's domain name (for example, https://www.example.com/) to your load balancer. For more information, see Routing Traffic to an ELB Load Balancer in the Amazon Route 53 Developer Guide or 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 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 (p. 192), whose private key is stored in the HSMs in your AWS CloudHSM cluster.

**To verify that your load balancer is working with OpenSSL s_client**

1. Use the following OpenSSL command to connect to your load balancer using HTTPS. Replace `<DNS name>` with the DNS name of your load balancer.

   ```bash
   $ openssl s_client -connect <DNS name>:443
   ```
2. Ensure that the certificate is the one that you configured the web server to use (p. 192), whose private key is stored in the HSMs in your AWS CloudHSM cluster.

You now have a website that is secured with HTTPS, with the web server's private key stored in the HSMs in your AWS CloudHSM cluster. Your website has two web servers and a load balancer to help improve efficiency and availability.

**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. 155) 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. For information about integrating an Oracle instance in Amazon RDS with AWS CloudHSM Classic, see Using AWS CloudHSM Classic to Store Amazon RDS Oracle TDE Keys in the Amazon Relational Database Service User Guide.

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 the Prerequisites (p. 202) to prepare your environment.
2. Follow the steps in Configure the Database and Generate the Master Encryption Key (p. 203) 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 crypto 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: Create A Cluster (p. 14). After you complete these steps, you have an active cluster with one HSM. You also have an Amazon EC2 instance, known as a client instance, running the Amazon Linux operating system and with the AWS CloudHSM client and command line tools installed and configured.

2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM (p. 33).

3. Connect to the client instance (p. 30) that you created previously. On the client instance, do the following:
   a. Install the AWS CloudHSM software library for PKCS #11 (p. 158).
   b. Install Oracle Database. For more information, see the Oracle Database documentation. AWS CloudHSM supports Oracle TDE integration with Oracle Database versions 11 and 12.
   c. Start the AWS CloudHSM client (p. 43).
   d. Update the configuration file for the command line tool known as cloudhsm_mgmt_util (p. 43).
   e. Use the command line tool known as cloudhsm_mgmt_util to create a crypto user (CU) on your cluster. For more information, see Managing HSM Users (p. 134).

After you complete these steps, you can Configure the Database and Generate the Master Encryption Key (p. 203).

Oracle TDE with AWS CloudHSM: Configure the Database and Generate the Master Encryption Key

To integrate Oracle TDE with your AWS CloudHSM cluster, complete the following steps:

1. Update the Oracle Database Configuration (p. 203) 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. 204) on the HSMs in your cluster.

For more information, see the following topics.

Topics
• Update the Oracle Database Configuration (p. 203)
• Generate the Oracle TDE Master Encryption Key (p. 204)

Update the Oracle Database Configuration

To update the Oracle Database configuration, complete the steps in the following procedure.
To update the Oracle configuration

1. Connect to the client instance (p. 30) that you created previously (p. 202). This is the same 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))
   ```

   Then 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 software library for PKCS #11.

   ```
   $ sudo mkdir -p /opt/oracle/extapi/64/hsm
   ```
5. Use one of the following commands to copy the AWS CloudHSM software library for PKCS #11 file to the directory that you created in the previous step.
   - If you installed the PKCS #11 library without Redis, run the following command.
     ```
     $ sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11_standard.so /opt/oracle/extapi/64/hsm/
     ```
   - If you installed the PKCS #11 library with Redis, run the following command.
     ```
     $ sudo cp /opt/cloudhsm/lib/libcloudhsm_pkcs11_redis.so /opt/oracle/extapi/64/hsm/
     ```

   **Note**
   The /opt/oracle/extapi/64/hsm directory must contain only one library file. Copy only the library file that corresponds to how you installed the PKCS #11 library (p. 159). If additional files exist in that directory, remove them.
6. Run the following command to change the ownership of the /opt/oracle directory and everything inside it.

   ```
   $ sudo chown -R oracle:dba /opt/oracle
   ```
7. Start the Oracle Database.

Generate the Oracle TDE Master Encryption Key

To generate the Oracle TDE master key on the HSMs in your cluster, complete the steps in the following procedure.

To generate the master key

1. Use the sqlplus command to open Oracle SQL*Plus. When prompted, type the system password that you set when you installed Oracle Database.
2. Run the SQL statement that creates the master encryption key, as shown in the following examples. Use the statement that corresponds to your version of Oracle Database. Replace `<CU user name>` with the user name of the crypto user (CU) that you created previously (p. 202). Replace `<password>` with the CU’s password.
Important
Run the following command only once. Each time the command is run, it creates a new master encryption key.

- For Oracle Database version 11, run the following SQL statement.

```sql
SQL> alter system set encryption key identified by "<CU user name>:<password>";
```

- For Oracle Database version 12, run the following SQL statement.

```sql
SQL> administer key management set key identified by "<CU user name>:<password>";
```

If the response is `System altered` or `keystore altered`, then you successfully generated and set the master key for Oracle TDE.

3. (Optional) Run the following command to verify the status of the Oracle wallet.

```sql
SQL> select * from v$encryption_wallet;
```

If the wallet is not open, use one of the following commands to open it. Replace `<CU user name>` with the user name of the crypto user (CU) that you created previously (p. 202). Replace `<password>` with the CU's password.

- For Oracle 11, run the following command to open the wallet.

```sql
SQL> alter system set encryption wallet open identified by "<CU user name>:<password>";
```

To manually close the wallet, run the following command.

```sql
SQL> alter system set encryption wallet close identified by "<CU user name>:<password>";
```

- For Oracle 12, run the following command to open the wallet.

```sql
SQL> administer key management set keystore open identified by "<CU user name>:<password>";
```

To manually close the wallet, run the following command.

```sql
SQL> administer key management set keystore close identified by "<CU user name>:<password>";
```
Getting API Logs with AWS CloudTrail

AWS CloudHSM is integrated with AWS CloudTrail, a service that records all AWS CloudHSM API calls in log files, and delivers those files to an Amazon Simple Storage Service (Amazon S3) bucket that you choose. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

CloudTrail records all calls to the AWS CloudHSM API, including those from the AWS CloudHSM console or from your code. For the full list of AWS CloudHSM API operations, see Actions in the AWS CloudHSM API Reference.

From the information recorded by CloudTrail, you can determine the request that was made to AWS CloudHSM, the source IP address from which the request was made, who made the request, when it was made, and so on. CloudTrail log files contain all AWS API calls in your AWS account, not only the AWS CloudHSM API calls.

To get started with CloudTrail, see Getting Started with CloudTrail in the AWS CloudTrail User Guide.

Understanding AWS CloudHSM Log File Entries in CloudTrail

CloudTrail log files contain one or more log entries in JSON (JavaScript Object Notation) format. Each log entry represents a single API call and includes information about the requested operation, the date and time of the operation, the request and response parameters, and so on. Log entries are not an ordered stack trace of API calls, so they do not appear in any particular order.

For more information about CloudTrail log entries, see CloudTrail Event Reference in the AWS CloudTrail User Guide.

The following example shows a CloudTrail log entry for a CreateHsm call to the AWS CloudHSM API.

```json
{
  "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"
      }
    }
  }
}
Troubleshooting AWS CloudHSM

If you encounter problems with AWS CloudHSM, the following topics can help you resolve them.

Topics

• Lost Connection to the Cluster (p. 208)
• Keep HSM Users In Sync Across HSMs In The Cluster (p. 209)
• Verify the Performance of the HSM (p. 209)

Lost Connection to the Cluster

When you configured the AWS CloudHSM client (p. 31), 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.

   # sudo stop cloudhsm-client

3. Use the following command to update the client's configuration file, providing the IP address that you found in a previous step.

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

4. Use the following command to start the client.

   # sudo start cloudhsm-client
Keep HSM Users In Sync Across HSMs In The Cluster

To manage your HSM's users (p. 134), you use an AWS CloudHSM command line tool known as clouphsm_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 clouphsm_mgmt_util to manage HSM users (p. 134), these user changes might affect only some of the cluster's HSMs—the ones that are in the clouphsm_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 clouphsm_mgmt_util configuration file before managing users. For more information, see Update the clouphsm_mgmt_util Configuration File (p. 43).

Verify the Performance of the HSM

To verify the performance of the HSMs in your AWS CloudHSM cluster, you can use the pkpspeed tool that is included with the AWS CloudHSM client software. To use pkpspeed, first connect to your client instance (p. 30) and then install and configure the AWS CloudHSM client software (p. 30).

After you install and configure the AWS CloudHSM client, issue the following command to start it.

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

If you 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.

To use pkpspeed, issue the `pkpspeed` command, specifying the user name and password of a crypto user (CU) on the HSM. Then set the options to use, considering the following recommendations.

**Recommendations**

- To test the performance of RSA sign and verify operations, choose the `RSA_CRT` cipher. Don't choose `RSA`. 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 to test the HSM's performance for RSA and AES operations.

**Example – Using pkpspeed to test RSA performance**

```
$ /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]: 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

Example – Using pkpspeed to test AES performance

$ /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...
AWS CloudHSM Limits

By default, the following limits apply to your AWS CloudHSM resources. These limits apply per AWS Region and per AWS account.

- Clusters: 4
- HSMs: 6

There is no limit on the number of HSMs per cluster, so you can distribute the HSMs across clusters any way you like.

To request an increase to these limits, use the service limit increase form in the AWS Support Center.
AWS CloudHSM Client and Software Version History

The following topics contain the version history for the AWS CloudHSM client and software libraries.

Topics
- AWS CloudHSM Client (p. 212)
- PKCS #11 Library (p. 213)
- OpenSSL Library (p. 214)
- Java Library (p. 215)

AWS CloudHSM Client

This section describes the updates to each version of the AWS CloudHSM client software.

To download the newest version of the AWS CloudHSM client, use one of the following locations.

- Amazon Linux (and compatible distributions): https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-latest.x86_64.rpm
- Ubuntu: https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client_latest_amd64.deb

Current Version: 1.0.11

You can download the current version of the AWS CloudHSM client from the following locations. For more information, see Install and Configure the Client (p. 30).

- Amazon Linux (and compatible distributions): https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-latest.x86_64.rpm
- Ubuntu: https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client_latest_amd64.deb

Significant changes in this version include the following:

- Improved load balancing.
- Improved performance.
- Improved handling of lost server connections.

Previous Versions

Version 1.0.10

You can download version 1.0.10 from https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-1.0-10.x86_64.rpm.

Significant changes in this version include the following:

- Updated the key_mgmt_util command line tool to enable AES wrapped key import.
• Improved performance.
• Fixed various bugs.

Version 1.0.8
You can download version 1.0.8 from https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-1.0-8.x86_64.rpm.

Significant changes in this version include the following:
• Improved setup experience.
• Added respawning to the client upstart service.
• Fixed various bugs.

Version 1.0.7
Significant changes in this version include the following:
• Added the pkpspeed performance testing tool. For more information, see Verify the Performance of the HSM (p. 209).
• Fixed bugs to improve stability and performance.

Version 1.0.0
This is the initial release.

PKCS #11 Library

Current Version: 1.0.11
You can download the current version of the AWS CloudHSM software library for PKCS #11 from one of the following locations. For more information, see Installing the PKCS #11 Library (p. 158).
• Amazon Linux (and compatible distributions): https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-pkcs11-latest.x86_64.rpm
• Ubuntu: https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-pkcs11_latest_amd64.deb

Significant changes in this version include the following:
• Added support for the CKM_RSA_PKCS_PSS sign/verify mechanism.

Previous Versions

Version 1.0.10
You can download version 1.0.10 from https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-pkcs11-1.0-10.x86_64.rpm.

Updated the version number for consistency.

Version 1.0.8
Significant changes in this version include the following:

- Fixed bugs to address relative paths in the Redis setup.

**Version 1.0.7**

Significant changes in this version include the following:

- Added an accelerated version of the library that uses a Redis local cache to improve performance.
- Fixed bugs related to attribute handling.
- Added the ability to generate ECDSA keys.

**Version 1.0.0**

This is the initial release.

**OpenSSL Library**

**Current Version: 1.0.11**

You can download the current version of the AWS CloudHSM software library for OpenSSL from any of the following locations. For more information, see Installing the OpenSSL Library (p. 161).

- Amazon Linux (and compatible distributions): https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-dyn-latest.x86_64.rpm
- Ubuntu: https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-dyn_latest_amd64.deb

Significant changes in this version include the following:

- Updated the version number for consistency.

**Previous Versions**

**Version 1.0.10**

You can download version 1.0.10 from https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-dyn-1.0-10.x86_64.rpm.

Updated the version number for consistency.

**Version 1.0.8**

Significant changes in this version include the following:

- Improved performance.

**Version 1.0.7**

Updated the version number for consistency.

**Version 1.0.0**
Java Library

Current Version: 1.0.11

You can download the current version of the AWS CloudHSM software library for Java from any of the following locations. For more information, see Installing the Java Library (p. 165).

- Amazon Linux (and compatible distributions): https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-jce-latest.x86_64.rpm
- Ubuntu: https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-jce_latest_amd64.deb

Significant changes in this version include the following:
- Improved the performance of several algorithms.
- Added Triple DES (3DES) key import feature.
- Various bug fixes.

Previous Versions

Version 1.0.10

You can download version 1.0.10 from https://s3.amazonaws.com/cloudhsmv2-software/cloudhsm-client-jce-1.0-10.x86_64.rpm.

Significant changes in this version include the following:
- Added support for additional algorithms.
- Improved performance.

Version 1.0.8

Significant changes in this version include the following:
- Updated the version number for consistency.

Version 1.0.7

Significant changes in this version include the following:
- Added support for additional algorithms.
- Signed the JAR files for compatibility with the Sun JCE provider.

Version 1.0.0

This is the initial release.
Document History

The following list contains the dates and descriptions of significant changes to the documentation for AWS CloudHSM.

November 9, 2017 – New and updated documentation for several features and tools

- Added documentation about using quorum authentication (M of N access control) for crypto officers (COs). For more information, see Enforcing Quorum Authentication (M of N Access Control) (p. 142).
- Added detailed documentation about using the key_mgmt_util command line tool. For more information, see key_mgmt_util Command Reference (p. 76).
- Updated the documentation for the latest versions of the AWS CloudHSM client and software libraries. For more information, see Client and Software Version History (p. 212).

October 25, 2017 – Added documentation for new third-party integration and new client and software releases

- Published new documentation for integrating third-party applications. For more information, see Oracle Database Encryption (p. 201).
- Published new documentation that corresponds with the latest versions of the AWS CloudHSM client and software libraries. For more information, see Client and Software Version History (p. 212).

October 12, 2017 – Added documentation for third-party integrations

- Published new documentation for integrating third-party applications. For more information, see Integrating Third-Party Applications (p. 187) and Improve Web Server Security with SSL/TLS Offload (p. 187).

September 20, 2017 – New client and software releases

- Published new documentation that corresponds with the latest versions of the AWS CloudHSM client and software libraries. For more information, see Client and Software Version History (p. 212).

August 29, 2017 – Added Java example code

- Published several code samples in Java (p. 168) that demonstrate how to use the AWS CloudHSM software library for Java (p. 162).

August 14, 2017 – New guide

- Published this guide, the AWS CloudHSM User Guide, to coincide with the release of the new AWS CloudHSM product.