Migrating to AWS CloudHSM

February 2019

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction

This guide is intended to assist developers who are upgrading to CloudHSM from CloudHSM Classic. This same information will be useful if you are planning to migrate from on-premises or other cloud-based Gemalto HSMs to AWS CloudHSM.

CloudHSM provides FIPS 140-2 validated HSMs under your control in the AWS Cloud. The service simplifies your day-to-day operations by automating common management tasks such as backups, failover and high availability. There are some differences in how you provision, initialize, administrate, and use CloudHSM clusters when compared to traditional HSMs such as the Gemalto Luna SA 5 HSMs provided by CloudHSM Classic. While most migrations will be straightforward, some will require special consideration.

This guide is intended to assist both administrators and application developers in selecting the migration path and development framework that best suits compliance considerations and workload requirements.

Options for Cryptography in AWS

AWS offers three alternatives to CloudHSM Classic or on-premises HSMs:

- AWS CloudHSM provides fully managed, FIPS 140-2 level 3 validated, single-tenant, customer-controlled HSMs. These are general purpose HSMs offering a wide range of common cryptographic algorithms. CloudHSM also provides industry-standard PKCS#11, JCE and OpenSSL SDKs to simplify integration with third-party and custom applications. CloudHSM provides the greatest flexibility but also requires the most development and management overhead, as well as potentially increased cost when compared to the other options.
- AWS Key Management Service (AWS KMS) provides FIPS 140-2 Level 2 validated key management in a multi-tenant service for data protection integrated with most AWS services and can be used with custom applications via SDK. KMS provides granular policies and access controls via AWS Identity and Access Management (IAM), audit logging via AWS CloudTrail, and automates common tasks such as key rotation. You can integrate KMS into your application though the AWS SDK directly or via the AWS Encryption SDK. By default, KMS is backed by FIPS 140-2 level 2 validated HSMs managed by AWS. You may also choose the Custom Key Store feature which enables you to store KMS keys on an AWS CloudHSM cluster.
- AWS Certificate Manager (ACM) Private CA provides a highly-available private CA service without the upfront investment and ongoing maintenance costs of operating your own private CA and HSMs. ACM Private CA, backed by FIPS 140-2 Level 3 validated HSMs, provides an intermediate CA for issuing private certificates. ACM Private CA provides APIs to create and deploy private certificates programmatically. You also have the flexibility to create private certificates for applications that require custom certificate lifetimes or resource names. ACM Private CA does not provide custom object signing.

This document provides information about migrating to AWS CloudHSM.

Identifying Workloads That May Require Special Attention

Some CloudHSM Classic workloads may require additional work in order to upgrade to the new CloudHSM. This section describes scenarios where an upgrade may not be straightforward, and suggests some potential workarounds.

Workloads Reliant on Algorithms that are not NIST-Approved

If your workload relies on legacy algorithms no longer approved by NIST, or new algorithms not yet approved by NIST, you will be unable to migrate without application changes and potentially re-encrypting any data stored using legacy algorithms. Examples include:

- Deriving keys using encryption. In PKCS#11 terms, NIST no longer allows using the C_DERIVE function with a CKM_*_ENCRYPT_DATA mechanism such as CKM_AES_ECB_ENCRYPT_DATA.
- Generating or using RSA 1024 private keys for encryption or signing.
- Deriving keys with proprietary algorithms or non-standard key exchange protocols.
- Specifying your own initialization vector (IV) for AES-GCM encryption.

CloudHSM Classic could be operated in FIPS-mode and non-FIPS mode, as explained in this documentation on using an HSM in non-FIPS mode. When running in non-FIPS mode, FIPS-validated hardware can provide algorithms that are not NIST-approved. AWS CloudHSM operates all HSMs in FIPS mode. This means only algorithms currently approved by NIST are available with CloudHSM.

If you have the flexibility to adapt your application to use currently approved cryptographic methods, you can upgrade to CloudHSM. In this case, reach out to us to let us know of your situation and we will work with you on a reasonable timeline for making these changes as well as providing any additional support you may need.

You may not have the flexibility to adapt your application if, for example, you work with partners who use legacy cryptography or if you must support old hardware in the field. In this case, reach out to us to discuss alternatives.

Workloads Reliant on Keys Locked Into the HSM

Private asymmetric keys cannot be exported from CloudHSM Classic HSMs. This is a design limitation of Gemalto HSMs in cloning mode. See the Rotating Keys section below for suggestions to rotate keys if possible. In some cases, you may be locked in to Gemalto HSMs without recourse. One example is if the key pair for your root CA resides on a Classic HSM, and you may not be able to securely update clients with a new trust store. Another example is if your workload delivers signed code or data to long-lived devices in the field, where these devices verify authenticity of this code or data using a hard-coded public key and you did not implement a mechanism to rotate the signing key.

For these uses cases, you need to acquire a Gemalto HSM on-premises or in a co-location facility. For a CA or root signing key, we recommend obtaining an inexpensive USB-attached Gemalto HSM (possibly

two for redundancy) to store your root keys offline. Intermediate certificates can then be generated in CloudHSM and used as issuing CA or signing certificates in AWS, leveraging the elasticity and low-latency of CloudHSM for online operations. Once you have acquired new Gemalto HSMs, follow the instructions to clone your Classic HSMs to the new HSMs. This ensures your private keys are in your control and safe on supported hardware. If you are able to upgrade your infrastructure to utilize intermediate CAs, then you can set up new keys on the new CloudHSM and upgrade your workload accordingly. If you are unable to upgrade your infrastructure, you can use AWS Direct Connect to connect your new HSMs to your AWS workload.

Differences between CloudHSM and CloudHSM Classic

While both CloudHSM Classic and the new CloudHSM provide single-tenant, FIPS-validated HSMs under your control in your VPC, there are important differences between the two services. This section highlights important differences, and considerations when configuring your HSM cluster and architecting your application.

Partitions vs. Users

In CloudHSM Classic, each HSM can have up to 20 partitions. Each Gemalto HSM partition has separate keys, credentials, and policies. In CloudHSM, this concept is replaced by Cryptographic Users (CUs), a richer capability where each user has its unique credential and owns its own keys. Instead of multiple users or applications sharing a partition, CUs can instead "share" specific keys to be used (but not managed) by other CUs. Similarly, Gemalto HSMs are configured and managed by security officers (SOs), while in CloudHSM the HSMs are administered by Cryptographic Officers (COs). COs can create other user accounts, manage user passwords, and set policy on the CloudHSM cluster, while a CU can create and utilize keys. With CloudHSM you can have up to 1024 users (COs and CUs combined) per HSM.

Managed Backups

With CloudHSM Classic, you typically maintain one or more dedicated devices for backup purposes, often across regions for disaster recovery. This is not required with the new CloudHSM.

A CloudHSM backup is a snapshot of the HSM, including users, keys, policies and certificates. If you are not actively using your HSMs, you can delete them, and the service will take a backup of the HSM before deleting it. If you accidentally misconfigure an HSM or unintentionally delete a key, you can recover using an older backup. If you require cross-region disaster recovery for the data on your HSM, you can copy any backup to another region. You do not need an active HSM to store a backup. There is no charge associated with backups at this time, and they are currently maintained permanently (unless you explicitly delete them).

A backup is routinely generated once daily, as well as when an HSM is added or removed from a cluster. Backups do not ensure durability of newly created keys until the daily backup runs. Therefore, consider your short-term durability requirements in sizing your CloudHSM cluster. Best practice is to maintain at least 2 active HSMs in any production cluster, spread across availability zones in the region. For

maximum durability and availability for mission-critical workloads, we recommend at least 3 HSMs across different Availability Zones.

Cross-Region Clustering

With the new CloudHSM, an HSM is always part of a CloudHSM cluster. HSMs within a cluster are clones of each other, sharing the same users, keys, policies and certificates. HSMs in a cluster are automatically synchronized, and any client automatically recognizes when HSMs are added or deleted from a cluster and balances the application load transparently. Clusters in CloudHSM are equivalent to high availability (HA) groups in CloudHSM Classic, with the exception that CloudHSM clusters are limited to regional clusters.

You can place Gemalto HSMs across different regions in a single HA group, and once configured properly the HSMs will stay synchronized. Currently, with the new CloudHSM you must implement the synchronization yourself in order to perform cross-region replication. One option is to set up independent clusters in each region and transfer keys between the clusters in real-time using asymmetric key wrapping. The other option is to clone a cluster to another region, and periodically synchronize keys across these cloned clusters directly using syncKey with cloudhsm_mgmt_util, or indirectly using masked objects with key_mgmt_util.

Steps to Migrate Your Application to AWS CloudHSM

Your migration to CloudHSM will generally involve:

1. Configuring your cluster and users
2. Transferring or rotating your keys
3. Integrating the appropriate CloudHSM development library with your application
4. Monitoring and scaling your workload
5. Deprovisioning unused HSMs

We will deep dive into each step in subsequent sections. This document assumes you are familiar with CloudHSM concepts. Refer to the Appendix for an overview of AWS CloudHSM software and tools and links to more information.

Configuring Your Cluster and Users

This section assumes you are familiar with the CloudHSM getting started documentation to create and initialize your cluster, and create cryptographic officer (CO) and cryptographic user (CU) users. In this section, we will describe strategies to achieve common administrative security goals with CloudHSM Clusters.

Considerations when Initializing a new Cluster

Creating a new CloudHSM cluster is slightly more complex than when first initializing a CloudHSM Classic HSM (which only requires a domain string and SO password) because the new CloudHSM adds advanced security features by leveraging X.509 certificates in addition to simple passwords. When you initialize a cluster, you sign the cluster certificate with a key pair (we call this CustomerCA) that you generate and own. For a production cluster, you should generate this key pair on a secure device (such as an offline

HSM or existing PKI infrastructure). A new CloudHSM cluster will provide you with a certificate signing request (CSR) which you should sign with this offline key pair. Your company may have key ceremony procedures you wish to use to govern and document this process.

Note that you must provide a self-signed (root) certificate to the HSM as your CustomerCA certificate. This must be a root certificate as opposed to an intermediate certificate, because the cluster certificate cannot be revoked and the CustomerCA certificate can never be changed. The cluster certificate allows you to cryptographically validate that you are interacting with your own cluster by verifying that the cluster public key (generated in hardware when you initialized your cluster) is signed by your offline private key. If you wish to incorporate an enterprise root certificate into this trust architecture, you can generate a separate CSR for the CustomerCA key pair, and endorse it separately with the enterprise root. When connecting to your HSM, you may optionally force validation of the client certificate to ensure it was issued by your CustomerCA root or trusted intermediate CA. You may then connect to the HSM which is endorsed by the CustomerCA as usual.

Considerations when Setting up CO accounts: You conduct cluster administration tasks, such as creating new users and setting quorum policies, through the CO account. AWS cannot reset account credentials if you lose them, so we strongly recommend you maintain at least 2 CO accounts on your cluster. If you set MofN quorum policies, you should have at least M+1 COs. Delete CO accounts with care. If you fall below the minimum quorum number, you will no longer be able to administer your cluster.

Considerations when Setting up CU accounts: Generally, you will use one CU account per application that will utilize the CloudHSM cluster, and optionally one CU per "key custodian" if you choose to implement separation of duties. If you do not use key custodians, you may simply generate or import keys for the application to use as the application CU directly. With key custodians, the responsible individuals would generate keys under their own CU and "share" those keys with the appropriate application user(s).

For example, you may wish to prevent working keys from being wrapped out under application credentials. You can achieve this with key sharing as follows. First, create a CU account operated by your cryptographic officer or key custodian. Use this CU account to create or import the working keys. Next, share this key with the application CU account. A CU with access to a shared key can perform cryptographic operations as allowed by the key's attributes, but is unable to wrap it out or change the key's attributes. This solution lets you keep keys exportable while also protecting keys from being compromised by a bug in the application. Note that key sharing is only supported through cloudhsm_mgmt_util at this time.

Coming soon: We expect to release support for trusted keys, unwrap templates and non-exportable clusters in the coming months. We also expect to release quorum controls for management commands on individual keys. These features will offer your security officers additional options for limiting the privileges of HSM users.

Creating, Transferring and Rotating Keys

As you migrate an existing workload, your use case will determine whether you can simply create new keys and cut over, or if you will need to transfer or rotate keys.

For use cases not tied to specific keys, such as web server SSL offload or signing with intermediate certificate authorities (CAs), you can create new keys on your CloudHSM cluster. For use cases where you need to continue using existing keys, such as data encryption applications, you may need to transfer cryptographic keys from your existing HSM to CloudHSM. For use cases where key transfer is not possible, such as non-exportable keys and all asymmetric private keys on CloudHSM Classic, you will need to rotate your keys. This section explains how to securely transfer keys between your current HSM and CloudHSM where possible. It also explains how to rotate or replace your keys where necessary.

Setting up New Keys on CloudHSM

You can generate new keys on CloudHSM using the CloudHSM SDK, the key_mgmt_util CLI, the CloudHSM OpenSSL engine, or via third party tools.

If you need to generate a certificate signing request (CSR) for the new key pair and have it signed by another certificate authority (CA), we recommend you work with the OpenSSL engine. You can find instructions on how to set up SSL offload with a new key pair in the Generate or Import Key documentation for NGINX on Linux and Create a CSR documentation for IIS Server on Windows.

If you will be using CloudHSM on Windows for any authentication or signing operations, we recommend you utilize the appropriate Microsoft tools along with CloudHSM's CNG/KSP SDK, to generate and use new keys. For example, you can run a Certificate Authority (CA) using Active Directory Certificate Services (ADCS), by configuring Windows Server to serve as a certificate authority using ADCS and CloudHSM. As another example, you can sign code using signtool on Windows with CloudHSM.

For all other applications, you can create symmetric keys or asymmetric key pairs using key_mgmt_util's key generation functions. The utility provides separate functions to generate symmetric keys (genSymKey), RSA key pairs (genRSAKeyPair) and ECC key pairs (genECCKeyPair).

Transferring Keys into CloudHSM

You can transfer a key out of your CloudHSM Classic or on-premise HSM if the key is marked exportable and if your HSM policy allows you to export (wrap) the key. Generally, you can transfer symmetric keys created as exportable keys. CloudHSM supports RSA-OAEP wrapping and RSA-AES wrapping. Both methods allow you to safely transfer keys between HSMs by wrapping them using a public key on your current HSM, and unwrapping them using the corresponding private key in CloudHSM.

This section demonstrates how to use RSA-OAEP wrapping to transfer keys securely between a Luna 5 HSM provided by CloudHSM Classic. We demonstrate this using these command line utilities for convenience:

- Key_mgmt_util (KMU), documented in the Key Management Utility user guide.
- Cloudhsm_mgmt_util (CMU), documented in the CloudHSM Management Utility user guide.
- Ckdemo, documented in the ckdemo user guide.

Step 1: Generate the wrapping keys on CloudHSM

Create an asymmetric RSA 2048 key pair in your CloudHSM cluster using key_mgmt_util. This is the key pair you will use to securely transfer keys across HSMs. The example below generates the key pair with the label "classic_wrap":


```
/opt/cloudhsm/bin/key_mgmt_util singlecmd loginHSM -u CU -s $HSM_USER -
p $HSM_PASSWORD genRSAKeyPair -m 2048 -e 65537 -l classic_wrap
```
Let's say the public key is generated with handle 407 and private key with handle 408.

Step 2: Export the public key from this key pair.

A public key is safe to share in the clear. Public keys are typically exchanged in the PEM format. You can retrieve your public key from the HSM using this command:

```
/opt/cloudhsm/bin/key_mgmt_util singlecmd loginHSM -u CU -s $HSM_USER
-p $HSM_PASSWORD exportPubKey -k 407 -out wrapping_public.pem
```

Step 3: Copy over the public key

Transfer the public key to the client instance you use to communicate with your Classic HSM. You can do this using secure copy "scp", or through an S3 bucket.

Step 4: Load the public key for wrapping

You must import the public key into the HSM partition where your keys reside. We suggest using the import function of certificate management utility "cmu" provided by Gemalto to import the key. Notice we're specifying the label "classic_wrap" for the public key here as well.

```
# cmu import -inputFile=wrapping_public.pem -label classic_wrap
```

```
Select token
 [1] Token Label: partition1
Enter choice: 1
```

Once the key is imported into your Luna HSM, you can find the handle assigned to it. You can do this with the list function of cmu.

```
# cmu list -label classic_wrap
```

```
Select token
 [1] Token Label: partition1
Enter choice: 1
handle=149    label=classic_wrap
```

Step 5: Wrap out the key from the Gemalto Luna SA

You can wrap out data keys on the HSM with the public key you just imported using the ckdemo utility. ckdemo is an interactive command line tool for the Luna HSM. Highlights of the process are as follows for users conversant with ckdemo. A detailed trace for this step is in the Appendix.

1. Use option 1 to select the partition where your keys are stored and you imported the public key, followed by option 3 to login to the partition
2. Use option 25 to set the CKA_WRAP attribute (attribute number 16) of the newly imported public key to 1. Setting the CKA_WRAP attribute allows you to use this public key for wrapping.
3. Use option 60 to begin wrapping the data key. Provide the data key and wrapping key handles when prompted. Choose mechanism [26] which is RSA_OAEP as the wrapping mechanism. The output will be written to wrapped.key

Step 6: Copy back the wrapped key

Transfer the wrapped key back to the client instance you use to communicate with your CloudHSM Cluster. As before, you can do this using the secure copy "scp", or through an S3 bucket.

Step 7: Unwrap the key into CloudHSM

We will use key_mgmt_util to unwrap the key with the private key generated in step 1. The sample command below is for unwrapping an AES-256 bit secret key. For other key types, you can find the appropriate arguments in the key_mgmt_util documentation for unwrapKey.

```
/opt/cloudhsm/bin/key_mgmt_util singlecmd loginHSM -u CU -s $HSM_USER -
p $HSM_PASSWORD unWrapKey -f wrapped.key -w 408 -m 8 -noheader -l
unwrapped_aes -kc 4 -kt 31
```

Optional: Customizing Attributes for the Unwrapped Key

This completes the key transfer process. Your key is now ready to use in the cryptographic user (CU) account you used to unwrap the key. If you need to customize the attributes of the unwrapped key, you have two options:

- You can use the setAttribute function in cloudhsm_mgmt_util to adjust select attributes of the key.
- You can unwrap the key using PKCS#11 which enables a custom unwrap template for the key, or using JCE code which allows you to specify the key label and key extractable attributes. We have provided sample PKCS#11 code and sample JCE code for wrapping and unwrapping, for your reference.

Rotating Keys

If your cryptographic keys are not exportable, you will need to rotate your keys as you migrate. Three common cases involving non-exportable keys, and corresponding rotation strategies, are discussed below.

Third-party Signed Certificates

Private keys cannot be exported from Gemalto Luna SA HSMs in cloning mode. All HSMs in CloudHSM Classic are in cloning mode, and therefore your asymmetric private keys are locked to the Gemalto ecosystem. Generally the private key on the HSM corresponds to an intermediate certificate, which is in turn signed by an offline root. You can rotate keys by obtaining a new intermediate certificate corresponding to a new key pair on CloudHSM. You can do this as follows:

- Create a new private key and generate the corresponding CSR using CloudHSM's OpenSSL engine, using the instructions in our SLL Offload tutorial.
- Sign the CSR with your offline root, or submit it to the appropriate external or enterprise CA for signing.
- You may have to register this new certificate with any partners who do not automatically verify the entire certificate chain.

Moving forward, you can sign all new requests (such as for documents, code or other certificates) with the new private key, corresponding to the new certificate. You can continue to verify signatures from

the original private key using the corresponding public key. You may choose to archive the older signing key and/or revoke the older certificate per your organization's policies.

Database Encryption

Transparent Data Encryption (TDE) for Oracle is a common use case for CloudHSM Classic. Oracle TDE uses envelope encryption. Data is encrypted with table keys, which are in turn encrypted by a master key on the HSM. This master key is non-exportable. In order to transition HSMs, you must switch your "hardware wallet" to CloudHSM. You do this in three steps. First, switch from the current hardware wallet (sometimes referred to as a keystore) which is your original HSM to a software wallet, by reverse migrating to a local wallet. Next, replace the PKCS#11 provider of your original HSM with the CloudHSM PKCS#11 library, as you set up prerequisites for TDE with CloudHSM. Third, switch the encryption wallet for your database to your CloudHSM cluster, as you configure TDE with CloudHSM. The database will automatically re-encrypt all data keys using the new master key.

If you maintain identical databases across multiple regions, you should do the above transition in one region first. Then, create or delete an HSM to force a new backup of your cluster, and then copy this backup to your secondary regions. Create a cluster in the secondary region using this copied backup as described here. As the new cluster is a clone of the original cluster, it already contains the hardware wallet for your database. You can simply connect your database to this cluster, and utilize the database in both regions as usual. You need to go through the process of cloning clusters because CloudHSM clusters are region-specific. If you rotate keys in the future, you must manually synchronize the keys between your cloned clusters across regions using the syncKey function of cloudhsm_mgmt_util.

Symmetric Keys for Envelope Encryption

Envelope encryption refers to the key architecture where one master key on the HSM encrypts/decrypts many data keys on the application host. If your master encryption key is exportable, simply use the instructions provided earlier to securely transfer the key. Typically, however, this master key is created as a non-exportable key. Key rotation involves creating a new master key on your CloudHSM cluster, decrypting all data keys with the old master key and re-encrypting them with the new master keyYou must re-encrypt all data keys because older wrapping keys will not be available on the new HSM. We suggest the following process:

1. Create the new master key in your new CloudHSM cluster using key_mgmt_util's genSymKey function with the –nex specified.
2. Set up an asymmetric key pair to serve as the transport key between your Classic HSM and new CloudHSM cluster, as described in steps 1-4 of the key transfer section above.
3. Building a PKCS#11 application or scripting ckdemo and key_mgmt_util, for each data key:
    a. On the Classic HSM: Unwrap or decrypt each data key using the old master key as appropriate to your application. Wrap out with the transport key as described in steps 5 and 6 of the key transfer section above.
    b. On the new CloudHSM: Unwrap with the transport key as described in section 7 of the key transfer section above, then encrypt or wrap with the new master key as appropriate to your application.

Using the CloudHSM SDK in Your Application

If you are using your HSMs through a custom application, you will need to adapt this application to utilize CloudHSM's PKCS#11 or JCE libraries. (Please note these libraries are not presently available for Windows).

CloudHSM's PKCS#11 and JCE libraries conform to the OASIS PKCS#11 2.40 and to Java Cryptography Extension (JCE) of the Java Cryptography Architecture (JCA) specification respectively. To migrate your application, you drop in the CloudHSM library in place of your current library, and typically make some minor adjustments as described in the sections below.

Note: If you are using your HSM through a third-party application, this application will need to interoperate with CloudHSM. You can reach out to your vendor to confirm support for the new CloudHSM, or reach out to AWS for more information.

Using CloudHSM PKCS#11

Authentication

Your application will need a cryptographic user (CU) account to sign in to the HSM. To authenticate your application to your cluster, you will need to provide the username and password for the CU account as the pin parameter of the PKCS #11 C_Login() function using the format <CU_user_name>:<password>

Supported Key Types and Algorithms

Carefully review the key types, mechanisms and algorithms supported by CloudHSM's PKCS#11 library, listed in CloudHSM's PKCS#11 documentation. If you see any missing algorithms, please reach out to us immediately.

Note that CloudHSM operates all HSMs in FIPS mode, and does not provide algorithms which are not FIPS approved. You can learn more at http://csrc.nist.gov/. As one implication, we do not permit user-supplied initialization vectors (IVs) when using AES-GCM. Please review the list of supported mechanisms carefully, and go through corresponding sample code and known issues, to ensure you understand the behavior and functionality of CloudHSM.

Helpful Links

- Getting started with PKCS#11: https://docs.aws.amazon.com/cloudhsm/latest/userguide/pkcs11-library.html
- Sample code: https://github.com/aws-samples/aws-cloudhsm-pkcs11-examples
- Known issues: https://docs.aws.amazon.com/cloudhsm/latest/userguide/KnownIssues.html#ki-pkcs11-sdk

Using CloudHSM JCE

Authentication

The JCE supports several options for you to provide login credentials, including implicit login via environment variables and explicit login via a login manager class that you implement. The benefit of

implicit login is the client will automatically handle re-login if the connection to HSMs breaks for any reason. The benefit of explicit login is your credentials are not stored anywhere on the system. However, since credentials are not cached, your application will not automatically reconnect if an HSM was disconnected for any reason and the connection needs to be re-established. Your application will need to detect disconnects and re-login as necessary. We have published sample code for managing reconnects for your reference.

Supported Key Types and Algorithms

Carefully review the key types, mechanisms and algorithms supported by CloudHSM's JCE library, documented here. If you see any missing algorithms, please reach out to us immediately.

Note that the JCE provides vendor-proprietary methods for certain methods including RSA-AES key wrapping. Please review the list of supported mechanisms carefully, and go through corresponding sample code and known issues, to ensure you understand the behavior and functionality of CloudHSM.

The JCE specification only supports specifying the attributes of label and non-extractable on a key that you generate, import or unwrap. CloudHSM assigns all other attributes a default value of true wherever appropriate for the given key type. CloudHSM does not currently support editing key attributes through the SDK once a key is generated. You can manually toggle the attributes OBJ_ATTR_ENCRYPT, OBJ_ATTR_DECRYPT, OBJ_ATTR_WRAP, OBJ_ATTR_UNWRAP, and OBJ_ATTR_LABEL using cloudhsm_mgmt_util. If you need to specify additional key attributes, please reach out to us. We are working to support additional attributes through the JCE.

Persistent and Session keys in the JCE

By default, any key created in the JCE is treated as a session key. Session keys are automatically deleted when the session is closed. A session is closed when the application logs out of the HSM, or when the network connection between the client and the HSM is broken for any reason.

To create a persistent or token key in the JCE, ensure that the persistent value in the key generation parameter specification is set to true. We have published sample code demonstrating the use of the persistent attribute for your reference.

Helpful Links

- Getting started with JCE: https://docs.aws.amazon.com/cloudhsm/latest/userguide/java-library.html
- Sample code: https://github.com/aws-samples/aws-cloudhsm-jce-examples
- Known issues: https://docs.aws.amazon.com/cloudhsm/latest/userguide/KnownIssues.html#ki-jce-sdk

Monitoring and Scaling your Workload

Monitoring Logs in AWS CloudHSM

CloudHSM provides three types of logs, introduced in our documentation on getting logs with CloudHSM.

Client logs are generated on your client instance, and contain information about connectivity status, any errors in communicating with the HSMs in your cluster, and success status. You should monitor these logs for errors. You should also monitor the cloudhsm_client daemon process to ensure it stays running. We recommend that you set up log rotation, for example using the logrotate tool, for client logs.

CloudHSM service API calls are logged to AWS CloudTrail.

Communication with your HSM occurs within an end-to-end encrypted channel between your client and HSM, and is not visible to the AWS CloudHSM service or any other AWS personnel. The audit logs generated by the HSM are delivered to CloudWatch. You can monitor these logs for management events such as login and key wrapping, as described in our documentation on monitoring CloudHSM Audit logs in Amazon CloudWatch. We will be adding more granular audit features in the future. Please open a support case if you require additional audit log functionality.

Monitoring Cluster and HSM Health

We publish various CloudHSM metrics in CloudWatch which enable you to monitor the health and status of your HSMs. If an HSM becomes unhealthy, the CloudHSM service will automatically replace it for you. This replacement typically takes 15 minutes. You may wish to proactively add an HSM to your cluster if you detect an HSM unhealthy metric. You may also wish to add an HSM if you see the HsmKeysSessionOccupied value approaching the maximum of 3,500, which is the maximum number of keys (session and persistent/token) that can be on a single HSM in the cluster. Unlike persistent/token keys, session keys do not replicate between HSMs, so adding additional HSMs allows you to use additional session keys.

CloudHSM does not currently support autoscaling. We recommend that you monitor the latency of calls to the HSM. If you see a consistent increase in latency, this means your HSMs are approaching their throughput limit, and you should add an HSM to your cluster.

We recommend at least 2 active HSMs, spread across availability zones in a region, for any production cluster. This ensures durability of token keys and availability of your workload. Note that the CloudHSM SLA does not apply to clusters with a single HSM or all HSMs in a single availability zone.

Deprovisioning Unused CloudHSM Classic HSMs

There are two steps to deleting a CloudHSM Classic device which is no longer necessary. First, you erase all your cryptographic material from the HSM by forcing a zeroization of the HSM. You force a zeroization by emptying all operational data including logs, and then performing a factory reset. In case you do not recall your login information, you can also force a factory reset by trying to login as administrator with an incorrect password three times in a row. Once your HSM is zeroized, you can deprovision your HSM so you are no longer billed for its use. You deprovision a device using delete-hsm in the AWS CLI.

CloudHSM Classic protects you from accidentally deprovisioning your HSMs. It is important that you carefully follow every step to properly deprovision your HSMs, as laid out in our knowledge center article to deprovision Classic HSMs. If you do not properly deprovision your instances, you will continue to be billed for the HSMs.

Note that CloudHSM Classic devices are not visible in the AWS console. To discover the CloudHSM Classic devices you are currently using, you can use list-hsms.

Avoid Common Errors when Deprovisioning CloudHSM Classic HSMs

- Do not tear down the CloudFormation stack before deprovisioning the CloudHSM Classic device: Tearing down the CloudFormation stack does not terminate the HSM. However, it will remove the network infrastructure you require to reach your HSM. As a result, you will be unable to deprovision the HSM, and will have to reach out to us for assistance.
- Do not deprovision the CloudHSM Classic device before zeroizing it: CloudHSM Classic will fail to terminate an HSM that has not been zeroized. After you run delete-hsm, be sure to run describe-hsm on the HSM to verify that it has correctly been placed in the TERMINATED state.
- Do not suspend or close your account before deprovisioning your HSMs: Unlike other AWS resources, CloudHSM Classic resources are not automatically deleted when you suspend or close an account. You should zeroize and deprovision your HSMs before you close your account.
- Do not delete the elastic network interface (ENI) for the CloudHSM Classic device before you confirm the device is no longer in your account: Once the ENI is deleted, you will be unable to deprovision the HSM, and will have to reach out to us for assistance.

Appendix 1: CloudHSM Tools and Software

CloudHSM's control plane offers functions to create and delete your clusters and HSMs, and manage backups. You will interact with your HSMs over the data plane, over an end-to-end encrypted channel between your client and your HSM, using software provided by CloudHSM. This software includes:

- CloudHSM Management Utility (CMU): Cloudhsm_mgmt_util is a command line tool, primarily intended for use by a CO. CMU enables you to manage users and policies for the cluster.
- Key Management Utility (KMU): key_mgmt_util is a command line tool, primarily intended for use by a CU. KMU is a convenient way to get started with CloudHSM, for example to generate new key pairs or migrate select keys.
- OpenSSL engine: The CloudHSM OpenSSL engine offloads select operations to the HSM, allowing you to conveniently use OpenSSL for common certificate generation and signing tasks, and for SSL offload. You can learn about using the OpenSSL engine for SSL offload using our Tutorial on SSL/TLS offload using CloudHSM.
- Software Development Kits (SDKs): CloudHSM provides standards-compliant PKCS#11 and JCE SDKs for your application development. These SDKs serve as drop-in replacements for the SDK provided by your current HSM vendor. Your custom application may require some adjustments to utilize these SDKs, typically around login management and key attributes. SDKs are discussed further later in this document. Note that PKCS#11 and JCE SDKs are not yet available for Microsoft Windows. Please reach out to us if you require this.
- Windows libraries: CloudHSM provides CNG and KSP libraries allowing you to use common Microsoft software with keys on your HSM. You can learn about how to sign certificates in our guide to configure Windows Server as a Certificate Authority (CA) with AWS CloudHSM; how to sign code in our guide on Signing executables with Microsoft SignTool.exe using AWS CloudHSM-backed certificates; and how to offload SSL/TLS in our tutorial on Using SSL/TLS Offload with AWS CloudHSM on Windows.

- Client daemon: CloudHSM relies on a client daemon to provide zero-configuration high availability. The daemon is used by KMU and all SDKs/libraries to communicate with all HSMs in the cluster. The daemon handles load balancing, manages disconnections and failures, and reconfigures itself to changes in the cluster configuration. Your application communicates with the client daemon running on the same instance via IPC over sockets. Communication between the client daemon and each HSM is protected by an end-to-end encrypted channel.

Appendix 2: Step-by-step Instructions to Wrap a Key using ckdemo

This section shows you how to use the ckdemo utility to wrap an exportable key on a Luna HSM using a public key that has already been imported to the same partition. The resulting wrapped key can then be unwrapped onto an HSM which has the corresponding public key.

Step 1: Run the ckdemo utility.

```
# ckdemo
```

Step 2: Open a session to the partition or HA partition group slot.

```
Enter your choice : 1


Slots available:
        slot#1 - LunaNet Slot
        slot#2 - LunaNet Slot
        ...
Select a slot: 1


SO[0], normal user[1], or audit user[2]? 1


Status: Doing great, no errors (CKR_OK)
```

Step 3: Login using the partition or HA partition group pin.

```
Enter your choice : 3
Security Officer[0]
```



```
Crypto-Officer  [1]
Crypto-User     [2]:
Audit-User      [3]: 1
Enter PIN         : password

Status: Doing great, no errors (CKR_OK)
```

Step 4: Permit CKA_WRAP on the RSA Public key you will use for wrapping. Use the key handle for your wrapping key in the place of the key handle highlighted below (149).

```
Enter your choice : 25

Which object do you want to modify (-1 to list available objects) : 149

Edit template for set attribute operation.

(1) Add Attribute   (2) Remove Attribute   (0) Accept Template :1

 0 - CKA_CLASS              1 - CKA_TOKEN
 2 - CKA_PRIVATE            3 - CKA_LABEL
 4 - CKA_APPLICATION        5 - CKA_VALUE
 6 - CKA_XXX                7 - CKA_CERTIFICATE_TYPE
 8 - CKA_ISSUER             9 - CKA_SERIAL_NUMBER
10 - CKA_KEY_TYPE          11 - CKA_SUBJECT
12 - CKA_ID                13 - CKA_SENSITIVE
14 - CKA_ENCRYPT           15 - CKA_DECRYPT
16 - CKA_WRAP              17 - CKA_UNWRAP
18 - CKA_SIGN              19 - CKA_SIGN_RECOVER
20 - CKA_VERIFY            21 - CKA_VERIFY_RECOVER
22 - CKA_DERIVE            23 - CKA_START_DATE
24 - CKA_END_DATE          25 - CKA_MODULUS
26 - CKA_MODULUS_BITS      27 - CKA_PUBLIC_EXPONENT
```



```
28 - CKA_PRIVATE_EXPONENT      29 - CKA_PRIME_1
30 - CKA_PRIME_2               31 - CKA_EXPONENT_1
32 - CKA_EXPONENT_2            33 - CKA_COEFFICIENT
34 - CKA_PRIME                 35 - CKA_SUBPRIME
36 - CKA_BASE                  37 - CKA_VALUE_BITS
38 - CKA_VALUE_LEN             39 - CKA_LOCAL
40 - CKA_MODIFIABLE            41 - CKA_ECDSA_PARAMS
42 - CKA_EC_POINT              43 - CKA_EXTRACTABLE
44 - CKA_ALWAYS_SENSITIVE      45 - CKA_NEVER_EXTRACTABLE
46 - CKA_CCM_PRIVATE           47 - CKA_FINGERPRINT_SHA1
48 - CKA_OUID                  49 - CKA_X9_31_GENERATED
50 - CKA_PRIME_BITS            51 - CKA_SUBPRIME_BITS
52 - CKA_USAGE_COUNT           53 - CKA_USAGE_LIMIT
54 - CKA_EKM_UID               55 - CKA_GENERIC_1
56 - CKA_GENERIC_2             57 - CKA_GENERIC_3
58 - CKA_FINGERPRINT_SHA256
Select which one: 16
Enter boolean value: 1


CKA_WRAP=01


(1) Add Attribute   (2) Remove Attribute   (0) Accept Template :0


Status: Doing great, no errors (CKR_OK)
```

Step 5: Verify exportability of your data key:

Check whether the Symmetric Key you want to migrate is exportable. In this example this is an AES Key with the handle 120. Make sure to replace the key handle highlighted below (120) with the actual handle of the key you want to migrate.

```
Enter your choice : 27
```



```
Enter handle of object to display (-1 to list available objects) : 120
Object handle=120
CKA_CLASS=00000004
CKA_TOKEN=01
CKA_PRIVATE=01
CKA_LABEL=Generated AES Key
CKA_KEY_TYPE=0000001f
CKA_ID=
CKA_SENSITIVE=01
CKA_ENCRYPT=01
CKA_DECRYPT=01
CKA_WRAP=01
CKA_UNWRAP=01
CKA_SIGN=01
CKA_VERIFY=01
CKA_DERIVE=01
CKA_START_DATE=
CKA_END_DATE=
CKA_VALUE_LEN=00000020
CKA_LOCAL=01
CKA_MODIFIABLE=01
CKA_EXTRACTABLE=01
CKA_ALWAYS_SENSITIVE=01
CKA_NEVER_EXTRACTABLE=00
CKA_CCM_PRIVATE=00
CKA_FINGERPRINT_SHA1=f8babf341748ba5810be21acc95c6d4d9fac75aa
CKA_OUID=29010002f90900005e850700
CKA_EKM_UID=
CKA_GENERIC_1=
```



```
CKA_GENERIC_2=

CKA_GENERIC_3=

CKA_FINGERPRINT_SHA256=7a8efcbff27703e281617be3c3d484dc58df6a78f6b144207c1a54ad32a98c0
0


Status: Doing great, no errors (CKR_OK)
```

As shown above the value of the CKA_EXTRACTABLE attribute should be 1. Otherwise the Key can't be exported.

Step 6: Wrap the key using an RSA Public Key. This will create a file called "wrapped.key" that contains the wrapped key. Make sure to replace the key handles highlighted below (149 and 120) with the actual handles of Public Key you wish to wrap with, and the key you want to migrate, respectively.

```
Enter your choice : 60

[1]DES-ECB      [2]DES-CBC      [3]DES3-ECB      [4]DES3-CBC  [7]CAST3-ECB      [8]CAST3-CBC

[9]RSA          [10]TRANSLA     [11]DES3-CBC-PAD  [12]DES3-CBC-PAD-IPSEC

[13]SEED-ECB    [14]SEED-CBC    [15]SEED-CBC-PAD  [16]DES-CBC-PAD

[17]CAST3-CBC-PAD [18]CAST5-CBC-PAD [19]AES-ECB      [20]AES-CBC

[21]AES-CBC-PAD   [22]AES-CBC-PAD-IPSEC [23]ARIA-ECB  [24]ARIA-CBC

[25]ARIA-CBC-PAD

[26]RSA_OAEP    [27]SET_OAEP

Select mechanism for wrapping: 26


Enter filename of OAEP Source Data [0 for none]: 0


Enter handle of wrapping key (-1 to list available objects) : 149


Enter handle of key to wrap (-1 to list available objects) : 120

Wrapped key was saved in file wrapped.key


Status: Doing great, no errors (CKR_OK)
```


Contributors

The following individuals and organizations contributed to this document:

- Ben Grubin, General Manager, AWS Cryptography
- Avni Rambhia, Senior Product Manager, AWS Cryptography
- Rohit Mathur, Software Development Manager, AWS Cryptography
- Somesh Chakrabarti, Security Engineer, AWS Cryptography
- Ryan Day, Software Development Engineer, AWS Cryptography
- Jason Fritcher, Software Development Engineer, AWS Cryptography
- Gautham Mudambi, Software Development Engineer, AWS Cryptography
- Patrick Palmer, Cloud Support Team Lead, AWS Premium Support
- Tracy Pierce, Senior Cloud Support Engineer, AWS Premium Support
- Mohamed AboElKheir, Cloud Support Engineer, AWS Premium Support

Further Reading

- CloudHSM guser uide: https://aws.amazon.com/documentation/cloudhsm/
- CloudHSM product overview, benefits and resources: https://aws.amazon.com/cloudhsm/
- Blog - Cost Effective Hardware Key Management at Cloud Scale for Sensitive & Regulated Workloads: https://aws.amazon.com/blogs/aws/aws-cloudhsm-update-costeffective-hardware-key-management/
- Webinar - Secure Scalable Key Storage in AWS: https://www.youtube.com/watch?v=hEVks207ALM
- Documentation - Verify the Identity and Authenticity of Your Cluster's HSM: http://docs.aws.amazon.com/cloudhsm/latest/userguide/verify-hsmidentity.html
- Documentation - AWS CloudHSM Client Tools and Software Libraries: http://docs.aws.amazon.com/cloudhsm/latest/userguide/client-toolsand-libraries.html#client