Cloudera Manager 7.10.1

Encrypting Data at Rest in Cloudera Manager

Date published: 2020-11-30
Date modified: 2023-01-03

https://docs.cloudera.com/

Legal Notice

Contents

Encrypting Data at Rest

Secure data at rest using encryption mechanisms and key management.

Cloudera clusters can use a combination of data at rest encryption mechanisms, including HDFS transparent encryption and Cloudera Navigator Encrypt. Both of these data at rest encryption mechanisms can be augmented with key management using Cloudera Navigator Key Trustee Server and Cloudera Navigator Key HSM.

Before configuring encryption for data at rest, familiarize yourself with the requirements in "Data at Rest Encryption Requirements".

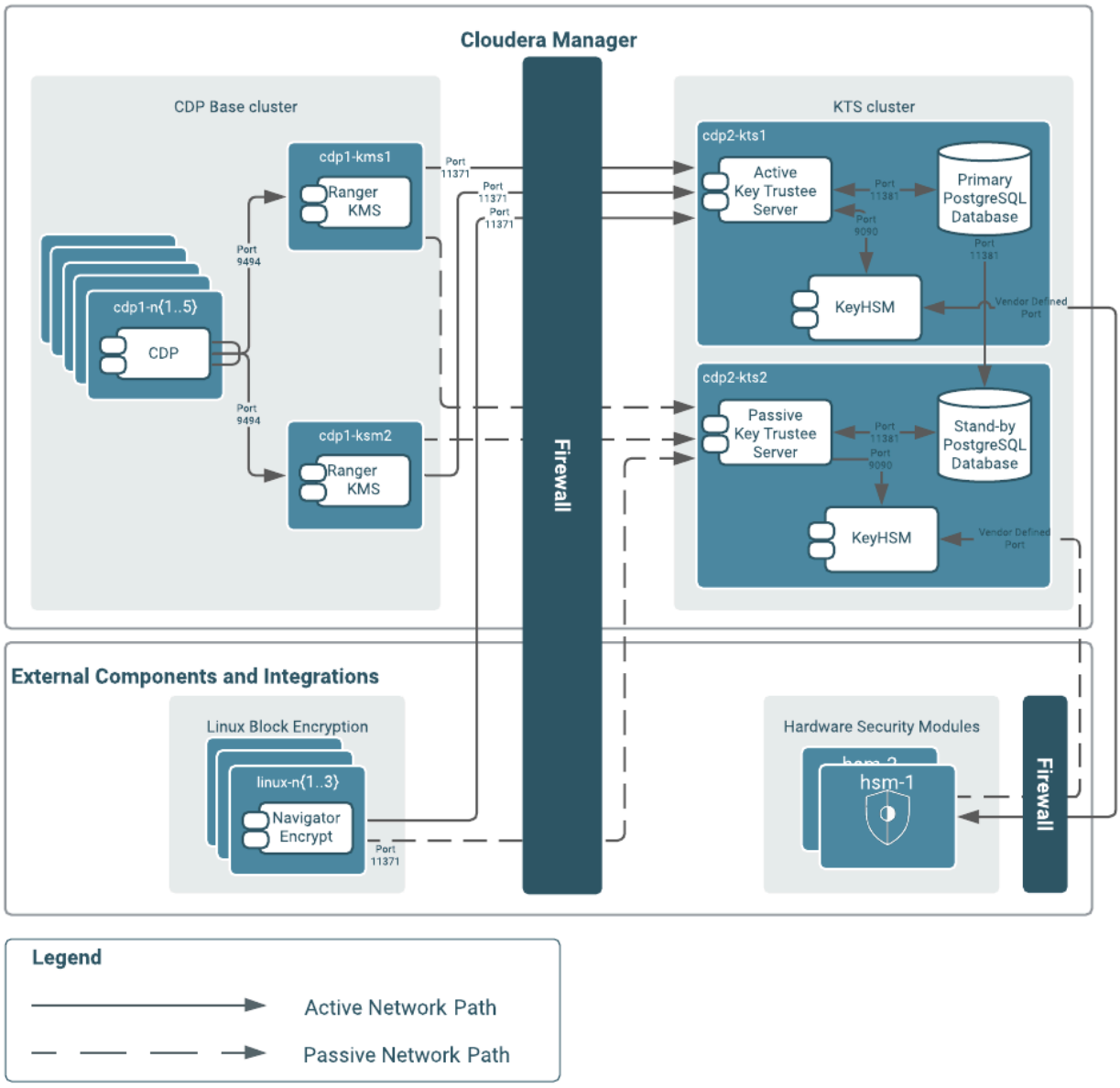
Related Information

Data at Rest Encryption Requirements

Data at Rest Encryption Reference Architecture

Deploying Cloudera Navigator for encrypting data at rest

The following diagram illustrates product component functional relationships:



To isolate Key Trustee Server from other CDP services, you must deploy Key Trustee Server on dedicated hosts in a separate cluster in Cloudera Manager. Deploy Ranger KMS on dedicated hosts in the same cluster as the CDP services that require access to Key Trustee Server. This provides the following benefits:

- You can restart your CDP cluster without restarting Key Trustee Server, avoiding interruption to other clusters or clients that use the same Key Trustee Server instance.
- You can manage the Key Trustee Server upgrade cycle independently of other cluster components.
- You can limit access to the Key Trustee Server hosts to authorized key administrators only, reducing the attack surface of the system.
- Resource contention is reduced. Running Key Trustee Server and Ranger KMS services on dedicated hosts prevents other cluster services from reducing available resources (such as CPU and memory) and creating bottlenecks.

If you are using virtual machines for the Key Trustee Server or Ranger KMS hosts, see "Resource Planning for Data at Rest Encryption".

Related Information

Resource Planning for Data at Rest Encryption

Data at Rest Encryption Requirements

Encryption comprises several components, each with its own requirements.

Overview

Data at rest encryption protection can be applied at a number of levels within Hadoop:

- OS filesystem-level
- Network-level
- HDFS-level (protects both data at rest and in transit)

For more information on the components, concepts, and architecture for encrypting data at rest, see "Encrypting Data at Rest".

Product Compatibility Matrix

See "Product Compatibility Matrix for Cloudera Navigator Encryption" for the individual compatibility matrices for each Cloudera Navigator encryption component.

Entropy Requirements

Cryptographic operations require entropy to ensure randomness.

You can check the available entropy on a Linux system by running the following command:

```
cat /proc/sys/kernel/random/entropy_avail
```

The output displays the entropy currently available. Check the entropy several times to determine the state of the entropy pool on the system. If the entropy is consistently low (500 or less), you must increase it by installing rng-tools and starting the rngd service.

Note: Due to a recent Linux kernel change the 'entropy_avail' reported by the kernel will always be 256. This will lead to erroneous alerts being issued if you changed the default settings for 'Host Entropy Thresholds'. These values will need to be changed to reflect the 256 value for 'entropy_avail'.

For RHEL 7, run the following commands:

```
sudo yum install rng-tools
cp /usr/lib/systemd/system/rngd.service /etc/systemd/system/
systemctl daemon-reload
systemctl start rngd
systemctl enable rngd
```

Make sure that the hosts running Key Trustee Server, Ranger KMS, and Navigator Encrypt have sufficient entropy to perform cryptographic operations.

Key Trustee Server Requirements

Recommended Hardware and Supported Distributions

Key Trustee Server must be installed on a dedicated server or virtual machine (VM) that is not used for any other purpose. The backing PostgreSQL database must be installed on the same host as the Key Trustee Server, and must

not be shared with any other services. For high availability, the active and passive Key Trustee Servers must not share physical resources. See "Resource Planning for Data at Rest Encryption" for more information.

The recommended minimum hardware specifications are as follows:

- Processor: 1 GHz 64-bit quad core
- Memory: 8 GB RAM
- Storage: 20 GB on moderate- to high-performance disk drives

For information on the supported Linux distributions, see "Product Compatibility Matrix for Cloudera Navigator Encryption".

Cloudera Manager Requirements

Installing and managing Key Trustee Server using Cloudera Manager requires Cloudera Manager 5.4.0 and higher. Key Trustee Server does not require Cloudera Navigator Audit Server or Metadata Server.

umask Requirements

Key Trustee Server installation requires the default umask of 0022.

Network Requirements

For new Key Trustee Server installations (5.4.0 and higher) and migrated upgrades (see "Cloudera Enterprise Upgrade Guide"> for more information), Key Trustee Server requires the following TCP ports to be opened for inbound traffic:

- 11371

  Clients connect to this port over HTTPS.
- 11381 (PostgreSQL)

  The passive Key Trustee Server connects to this port for database replication.

For upgrades that are not migrated to the CherryPy web server, the pre-upgrade port settings are preserved:

- 80

  Clients connect to this port over HTTP to obtain the Key Trustee Server public key.
- 443 (HTTPS)

  Clients connect to this port over HTTPS.
- 5432 (PostgreSQL)

  The passive Key Trustee Server connects to this port for database replication.

TLS Certificate Requirements

To ensure secure network traffic, Cloudera recommends obtaining Transport Layer Security (TLS) certificates specific to the hostname of your Key Trustee Server. To obtain the certificate, generate a Certificate Signing Request (CSR) for the fully qualified domain name (FQDN) of the Key Trustee Server host. The CSR must be signed by a trusted Certificate Authority (CA). After the certificate has been verified and signed by the CA, the Key Trustee Server TLS configuration requires:

- The CA-signed certificate
- The private key used to generate the original CSR
- The intermediate certificate/chain file (provided by the CA)

Cloudera recommends not using self-signed certificates. If you use self-signed certificates, you must use the --sk ip-ssl-check parameter when registering Navigator Encrypt with the Key Trustee Server. This skips TLS hostname validation, which safeguards against certain network-level attacks. For more information regarding insecure mode, see "Registering Cloudera Navigator Encrypt with Key Trustee Server>Registration Options".

Ranger KMS Requirements

Recommended Hardware and Supported Distributions

The recommended minimum hardware specifications are as follows:

- Processor: 1 GHz 64-bit quad core
- Memory: 8 GB RAM
- Storage: 20 GB on moderate- to high-performance disk drives

For information on the supported Linux distributions, see "Product Compatibility Matrix for Cloudera Navigator Encryption".

The Ranger KMS workload is CPU-intensive. Cloudera recommends using machines with capabilities equivalent to your NameNode hosts, with Intel CPUs that support AES-NI for optimum performance. Also, Cloudera strongly recommends that you enable TLS for both the HDFS and the Ranger services to prevent the passage of plain text key material between the KMS and HDFS data nodes.

Key HSM Requirements

The following are prerequisites for installing Navigator Key HSM:

- Oracle Java Runtime Environment (JRE) 7 or higher with Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files:
  - JCE for Java SE 7
  - JCE for Java SE 8
- A supported Linux distribution. See "Product Compatibility Matrix for Cloudera Navigator Encryption".
- A supported HSM device:
  - SafeNet Luna
    - HSM firmware version: 6.2.1
    - HSM software version: 5.2.3-1
  - SafeNet KeySecure
    - HSM firmware version: 6.2.1
    - HSM software version: 8.0.1
  - Thales nSolo, nConnect
    - HSM firmware version: 11.4.0
    - Client software version: 2.28.9cam136
- Key Trustee Server 3.8 or higher

Important: You must install Key HSM on the same host as Key Trustee Server.

Root access is required to install Navigator Key HSM.

Navigator Encrypt Requirements

Operating System Requirements

- For supported Linux distributions, see "Product Compatibility Matrix for Cloudera Navigator Encryption".

Supported command-line interpreters:

- sh (Bourne)
- bash (Bash)
- dash (Debian)

Note: Navigator Encrypt does not support installation or use in chroot environments.

Network Requirements

For new Navigator Key Trustee Server (5.4.0 and higher) installations, Navigator Encrypt initiates TCP traffic over port 11371 (HTTPS) to the Key Trustee Server.

For upgrades and Key Trustee Server versions lower than 5.4.0, Navigator Encrypt initiates TCP traffic over ports 80 (HTTP) and 443 (HTTPS) to the Navigator Key Trustee Server.

Internet Access

You must have an active connection to the Internet to download many package dependencies, unless you have internal repositories or mirrors containing the dependent packages.

Maintenance Window

Data is not accessible during the encryption process. Plan for system downtime during installation and configuration.

Administrative Access

To enforce a high level of security, all Navigator Encrypt commands require administrative (root) access (including installation and configuration). If you do not have administrative privileges on your server, contact your system administrator before proceeding.

Package Dependencies

Navigator Encrypt requires these packages, which are resolved by your distribution package manager during installation:

- dkms
- keyutils
- ecryptfs-utils
- libkeytrustee
- navencrypt-kernel-module
- openssl
- lsof
- gcc
- cryptsetup

These packages may have other dependencies that are also resolved by your package manager. Installation works with gcc, gcc3, and gcc4.

Related Information

Product Compatibility Matrix for Cloudera Navigator Encryption

Encrypting Data at Rest

Resource Planning for Data at Rest Encryption

Resource Planning for Data at Rest Encryption

High Availablity for Key Trustee Server and Ranger KMS

For production environments, you must configure high availability for:

- Key Trustee Server
- Ranger KMS

Key Trustee Server and Ranger KMS HA Planning

For high availability, you must provision two dedicated Key Trustee Server hosts and at least two dedicated Ranger KMS hosts, for a minimum of four separate hosts. Do not run multiple Key Trustee Server or Ranger KMS services on the same physical host, and do not run these services on hosts with other cluster services. Doing so causes resource contention with other important cluster services and defeats the purpose of high availability. See "Data at Rest Encryption Reference Architecture" for more information.

The Ranger KMS workload is CPU intensive. Cloudera recommends using machines with capabilities equivalent to your NameNode hosts, with Intel CPUs that support AES-NI for optimum performance.

Make sure that each host is secured and audited. Only authorized key administrators should have access to them. Red Hat provides security guides for RHEL:

- RHEL 7 Security Guide

For hardware sizing information, see "Data at Rest Encryption Requirements" for recommendations for each Cloudera Navigator encryption component.

For Cloudera Manager deployments, deploy Key Trustee Server in its own dedicated cluster. Deploy Ranger KMS in each cluster that uses Key Trustee Server. See "Data at Rest Encryption Reference Architecture" for more information.

For information about enabling Key Trustee Server high availability, refer to "Setting up Key Trustee Server High Availability".

For information about enabling Ranger KMS high availability, refer to "Installing Ranger KMS backed with a Key Trustee Server and HA".

Virtual Machine Considerations

If you are using virtual machines, make sure that the resources (such as virtual disks, CPU, and memory) for each Key Trustee Server and Ranger KMS host are allocated to separate physical hosts. Hosting multiple services on the same physical host defeats the purpose of high availability, because a single machine failure can take down multiple services.

Important:

Each and every HDFS operation, even for non-encrypted data, will contact KMS. This impacts CPU requirements and must be considered when planning Key Management System and Key Trustee Server implementations.

To maintain the security of the cryptographic keys, make sure that all copies of the virtual disk (including any back-end storage arrays, backups, snapshots, and so on) are secured and audited with the same standards you apply to the live data.

Related Information

Data at Rest Encryption Requirements

Data at Rest Encryption Reference Architecture

RHEL 7 Security Guide

Installing Ranger KMS backed with a Key Trustee Server and HA

Setting up Key Trustee Server High Availability

HDFS Transparent Encryption

Data encryption is mandatory for many government, financial, and regulatory entities, worldwide, to meet privacy and other security requirements. For example, the card payment industry has adopted the Payment Card Industry Data Security Standard (PCI DSS) for information security.

Other examples include requirements imposed by United States government's Federal Information Security Management Act (FISMA) and Health Insurance Portability and Accountability Act (HIPAA). Encrypting data stored in HDFS can help your organization comply with such regulations.

Transparent encryption for HDFS implements transparent, end-to-end encryption of data read from and written to HDFS blocks across your cluster. Transparent means that end-users are unaware of the encryption/decryption processes, and end-to-end means that data is encrypted at-rest and in-transit (see the Cloudera Engineering Blog post for complete details).

Note: HDFS Transparent Encryption is not the same as TLS encryption. Clusters configured TLS/SSL encrypt network communications throughout the cluster. Depending on the type of services your cluster supports, you may want to configure both HDFS Transparent Encryption and TLS/SSL for the cluster.

HDFS encryption has these capabilities:

- Only HDFS clients can encrypt or decrypt data.
- Key management is external to HDFS. HDFS cannot access unencrypted data or encryption keys. Administration of HDFS and administration of keys are separate duties encompassed by distinct user roles (HDFS administrator, Key Administrator), thus ensuring that no single user has unrestricted access to both data and keys.
- The operating system and HDFS interact using encrypted HDFS data only, mitigating threats at the OS- and file-system-level.
- HDFS uses the Advanced Encryption Standard-Counter mode (AES-CTR) encryption algorithm. AES-CTR supports a 128-bit encryption key (default), or can support a 256-bit encryption key when Java Cryptography Extension (JCE) unlimited strength JCE is installed.
- HDFS encryption has been designed to take advantage of the AES-NI instruction set, a hardware-based encryption acceleration technique, so your cluster performance should not adversely affected by configuring encryption. (The AES-NI instruction set can be an order of magnitude faster than software implementations of AES.) However, you may need to update cryptography libraries on your HDFS and MapReduce client hosts to use the acceleration mechanism.

Related Information

Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7 Download
Optimizing Performance for HDFS Transparent Encryption
Setting up Data at Rest Encryption for HDFS

Key Concepts and Architecture

HDFS must be integrated with an external enterprise-level keystore. The Hadoop Key Management server serves as a proxy between HDFS clients and the keystore. The keystore can be either the Cloudera Navigator Key Trustee Server or a support Hardware Security Module.

HDFS transparent encryption involves the creation of an encryption zone, which is a directory in HDFS whose contents will be automatically encrypted on write and decrypted on read. Each encryption zone is associated with a key (EZ Key) specified by the key administrator when the zone is created. The EZ keys are stored on the external keystore.

Keystores and the Key Management Server

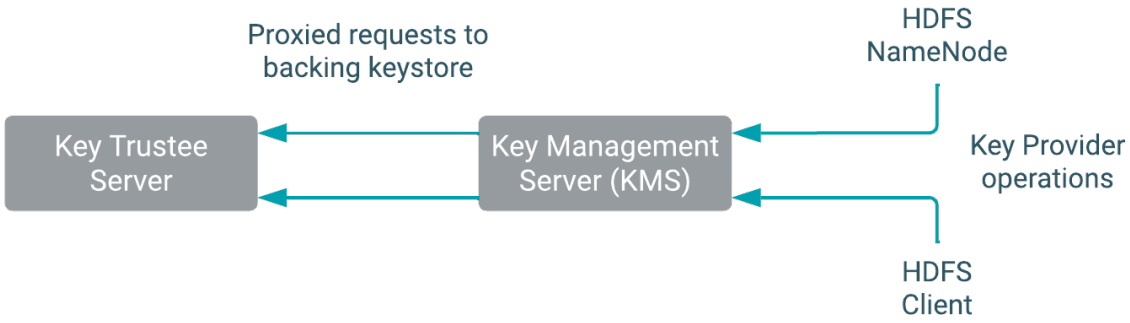
Integrating HDFS with an external, enterprise-level keystore is the first step to deploying transparent encryption. This is because separation of duties between a key administrator and an HDFS administrator is a very important aspect of this feature.

However, most keystores are not designed for the encrypt/decrypt request rates seen by Hadoop workloads. This led to the development of a new service, called the Key Management Server (KMS), which serves as a proxy between HDFS clients and the backing keystore. Both the keystore and KMS must use Hadoop's KeyProvider API to interact with each other and with HDFS clients.

While HDFS encryption can be used with a local Java KeyStore for key management, Cloudera does not recommend this for production environments where a more robust and secure key management solution should be used. Cloudera offers the following two options for enterprise-grade key management:

- Key Trustee Server is a key store for managing encryption keys. To integrate with the Key Trustee Server, Cloudera provides Ranger KMS.
- Hardware security modules (HSM) are third-party appliances that provide the highest level of security for keys.

The diagram below illustrates how HDFS clients and the NameNode interact with an enterprise keystore through the Key Management Server. The keystore can be either the Key Trustee Server or a support HSM.



Data Encryption Components and Solutions

Cloudera supports four encryption components which may be combined as unique solutions. When selecting a Key Management System (KMS), you must decide which components meet the key management and encryption requirements for your enterprise.

Cloudera Encryption components

Descriptions of Cloudera components for encrypting data at rest follow:

Ranger Key Management System (KMS)

Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database or you can use the secure key store like Key Trustee Server. Cryptographic key management service supporting HDFS TDE. Not a general purpose Key Management System, as opposed to Hadoop KMS which stores keys in file based Java Keystore, can be accessed only through KeyProvider API.

Key Trustee Server (KTS)

Key Manager that stores and manages cryptographic keys and other security artifacts

Key HSM

Allows Ranger Key Trustee Server to seamlessly integrate with the following hardware security modules (HSM)

- Luna 6 & 7
- CipherTrust
- GCP Cloud HSM
- Azure Key Vault

Navigator Encrypt

Transparently encrypts and secures data at rest without requiring changes to your applications

Cloudera Encryption solutions

You can deploy encryption components as any of the following solutions for encrypting data at rest:
Ranger KMS Only

- Consists of ONLY Ranger KMS with a backend database that provides key storage
- Ranger KMS provides enterprise-grade key management

Ranger KMS + HSM

- Consists of Ranger KMS with database + integration with a backend hardware security module (HSM)
- Ranger KMS provides enterprise-grade key management
- HSM provides encryption zone key protection
- HSM stores only the encryption master key

Ranger KMS + Key Trustee Server (KTS)

- Ranger KMS provides enterprise-grade key management
- KTS provides the key store that stores and manages cryptographic keys and other security artifacts

Ranger KMS + KTS + Key HSM

Allows Cloudera Key Trustee Server to seamlessly integrate with a HSM in addition to above items

Encryption Zones and Keys

HDFS transparent encryption introduces the concept of an *encryption zone* (EZ), which is a directory in HDFS whose contents will be automatically encrypted on write and decrypted on read.

Encryption zones always start off as empty directories, and tools such as distcp with the -skipcrccheck -update flags can be used to add data to a zone. (These flags are required because encryption zones are being used.) Every file and subdirectory copied to an encryption zone will be encrypted.

Note: An encryption zone cannot be created on top of an existing directory.

Each encryption zone is associated with a key (EZ Key) specified by the key administrator when the zone is created. EZ keys are stored on a backing keystore external to HDFS. Each file within an encryption zone has its own encryption key, called the Data Encryption Key (DEK). These DEKs are encrypted with their respective encryption zone's EZ key, to form an Encrypted Data Encryption Key (EDEK).

The following diagram illustrates how encryption zone keys (EZ keys), data encryption keys (DEKs), and encrypted data encryption keys (EDEKs) are used to encrypt and decrypt files.

EDEKs are stored persistently on the NameNode as part of each file's metadata, using HDFS extended attributes. EDEKs can be safely stored and handled by the NameNode because the hdfs user does not have access to the EDEK's encryption keys (EZ keys). Even if HDFS is compromised (for example, by gaining unauthorized access to a superuser account), a malicious user only gains access to the encrypted text and EDEKs. EZ keys are controlled by a separate set of permissions on the KMS and the keystore.

An EZ key can have multiple key versions, where each key version has its own distinct key material (that is, the portion of the key used during encryption and decryption). Key rotation is achieved by bumping up the version for an EZ key. Per-file key rotation is then achieved by re-encrypting the file's DEK with the new version of the EZ key to create new EDEKs. HDFS clients can identify an encryption key either by its key name, which returns the latest version of the key, or by a specific key version.

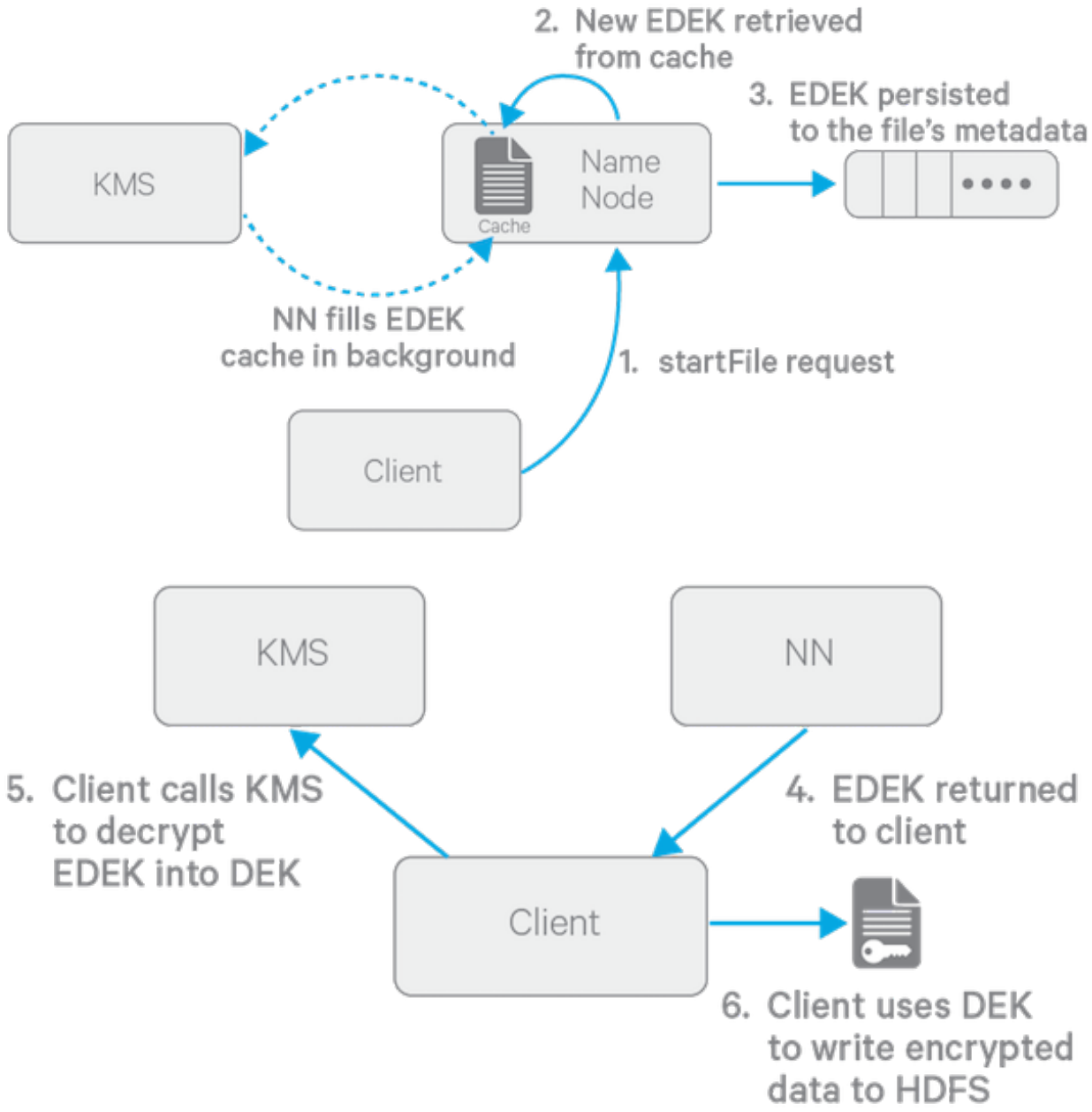
Related Information

Managing Encryption Keys and Zones

Extended Attributes in HDFS

Accessing Files Within an Encryption Zone

To encrypt a new file, the HDFS client requests a new EDEK from the NameNode. The HDFS client then asks the KMS to decrypt it with the encryption zone's EZ key. This decryption results in a DEK, which is used to encrypt the file.



The diagram above depicts the process of writing a new encrypted file. Note that the EDEK cache on the NameNode is populated in the background. Since it is the responsibility of KMS to create EDEKs, using a cache avoids having to call the KMS for each create request. The client can request new EDEKs directly from the NameNode.

To decrypt a file, the NameNode provides the HDFS client with the file's EDEK and the version number of the EZ key that was used to generate the EDEK. The HDFS client requests the KMS to decrypt the file's EDEK with the encryption zone's EZ key, which involves checking that the requesting client has permission to access that particular version of the EZ key. Assuming decryption of the EDEK is successful, the client then uses this DEK to decrypt the file.

Encryption and decryption of EDEKs takes place entirely on the KMS. More importantly, the client requesting creation or decryption of an EDEK never handles the EZ key. Only the KMS can use EZ keys to create and decrypt EDEKs as requested. It is important to note that the KMS does not store any keys, other than temporarily in its cache. It is up to the enterprise keystore to be the authoritative storage for keys, and to ensure that keys are never lost, as a lost key is equivalent to introducing a security hole. For production use, Cloudera recommends you deploy two or more redundant enterprise key stores.

Optimizing Performance for HDFS Transparent Encryption

CDP implements the *Advanced Encryption Standard New Instructions* (AES-NI), which provide substantial performance improvements. To get these improvements, you need a recent version of libcrypto.so on HDFS and MapReduce client hosts -- that is, any host from which you originate HDFS or MapReduce requests.

Many OS versions have an older version of the library that does not support AES-NI. The instructions that follow tell you what you need to do for each OS version that CDP supports.

Warning: To ensure that HDFS encryption functions as expected, the steps described in this section are mandatory for production use.

RHEL/CentOS 6.5 or later

The installed version of libcrypto.so supports AES-NI, but you need to install the openssl-devel package on all clients:

```
sudo yum install openssl-devel
```

RHEL/CentOS 6.4 or earlier 6.x versions, or SLES 11

Download and extract a newer version of libcrypto.so from a CentOS 6.5 repository and install it on all clients in /var /lib/hadoop/extra/native/:

1. Download the latest version of the openssl package. For example:

   ```
   wget http://mirror.centos.org/centos/6/os/x86_64/Packages/openssl-1.0.1e
   -30.el6.x86_64.rpm
   ```

   The libcrypto.so file in this package can be used on SLES 11 as well as RHEL/CentOS.
2. Decompress the files in the package, but do not install it:

   ```
   rpm2cpio openssl-1.0.1e-30.el6.x86_64.rpm | cpio -idmv
   ```
3. If you are using parcels, create the /var/lib/hadoop/extra/native/ directory:

   ```
   sudo mkdir -p /var/lib/hadoop/extra/native
   ```
4. Copy the shared library into /var/lib/hadoop/extra/native/. Name the target file libcrypto.so, with no suffix at the end, exactly as in the command that follows.

   ```
   sudo cp ./usr/lib64/libcrypto.so.1.0.1e /var/lib/hadoop/extra/native/lib
   crypto.so
   ```

Debian Wheezy

The installed version of libcrypto.so supports AES-NI, but you need to install the libssl-devel package on all clients:

```
sudo apt-get install libssl-dev
```

Ubuntu Precise and Ubuntu Trusty

Install the libssl-devel package on all clients:

```
sudo apt-get install libssl-dev
```

Testing if encryption optimization works

To verify that a client host is ready to use the AES-NI instruction set optimization for HDFS encryption at rest, use the following command:

```
hadoop checknative
```

You should see a response such as the following:

```
14/12/12 13:48:39 INFO bzip2.Bzip2Factory: Successfully loaded & initialized
 native-bzip2
library system-native14/12/12 13:48:39 INFO zlib.ZlibFactory: Successfully
 loaded & initialized native-zlib library
Native library checking:
hadoop:  true /usr/lib/hadoop/lib/native/libhadoop.so.1.0.0
zlib:    true /lib64/libz.so.1
snappy:  true /usr/lib64/libsnappy.so.1
lz4:     true revision:99
bzip2:   true /lib64/libbz2.so.1
openssl: true /usr/lib64/libcrypto.so
```

If you see true in the openssl row, Hadoop has detected the right version of libcrypto.so and optimization will work. If you see false in this row, you do not have the right version.

Managing Encryption Keys and Zones

Interacting with the KMS and creating encryption zones requires the use of two CLI commands: hadoop key and hdfs crypto. Before getting started with creating encryption keys and setting up encryption zones, make sure that your KMS ACLs have been set up according to best practices.

Validating Hadoop Key Operations

Use hadoop key create to create a test key, and then use hadoop key list to retrieve the key list.

```
hadoop key create KEYTRUSTEE_TEST
hadoop key list
```

Warning: If you are using or plan to use Cloudera Navigator Key HSM in conjunction with Cloudera Navigator Key Trustee Server, ensure that:

Key names begin with alphanumeric characters and do not use special characters other than hyphen (-), period (.), or underscore (_). Using other special characters can prevent you from migrating your keys to an HSM.

Creating Encryption Zones

Once a KMS has been set up and the NameNode and HDFS clients have been correctly configured, use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

Important: Cloudera does not currently support configuring the root directory as an encryption zone. Nested encryption zones are also not supported.

Important: The Java Keystore KMS default Truststore (for example, org.apache.hadoop.crypto.key.JavaKeyStoreProvider) does not support uppercase key names.

- Create an encryption key for your zone as keyadmin for the user/group (regardless of the application that will be using the encryption zone):

  ```
  hadoop key create <key_name>
  ```

- Create a new empty directory and make it an encryption zone using the key created above.

  ```
  hadoop fs -mkdir /ENCRYPTION_ZONE
  ```

```
hdfs crypto -createZone -keyName <key_name> -path /ENCRYPTION_ZONE
```

You can verify creation of the new encryption zone by running the -listZones command. You should see the encryption zone along with its key listed as follows:

```
$ hdfs crypto -listZones
/ENCRYPTION_ZONE    <key_name>
```

Warning: Do not delete an encryption key as long as it is still in use for an encryption zone. This results in loss of access to data in that zone. Also, do not delete the KMS service, as your encrypted HDFS data will be inaccessible. To prevent data loss, first copy the encrypted HDFS data to a non-encrypted zone using the distcp command.

Related Information

Configuring CDP Services for HDFS Encryption

Adding Files to an Encryption Zone

You can add files to an encryption zone by copying them to the encryption zone using distcp.

For example:

```
hadoop distcp /user/dir /ENCRYPTION_ZONE
```

Important: You can delete files or directories that are part of an HDFS encryption zone.

Copying data from encrypted locations

By default, distcp compares checksums provided by the filesystem to verify that data was successfully copied to the destination. When copying from an encrypted location, the file system checksums will not match because the underlying block data is different. This is true whether or not the destination location is encrypted or unencrypted.

In this case, you can specify the -skipcrccheck and -update flags to avoid verifying checksums. When you use -ski pcrccheck, distcp checks the file integrity by performing a file size comparison, right after the copy completes for each file.

Deleting Encryption Zones

To remove an encryption zone, delete the encrypted directory.

Warning: This command deletes the entire directory and all of its contents. Ensure that the data is no longer needed before running this command.

```
hadoop fs -rm -r -skipTrash /ENCRYPTION_ZONE
```

Important: The Key Trustee KMS does not directly run a key deletion (for example, it may perform a soft delete instead, or delay the actual deletion to prevent mistakes). In these cases, errors may occur when creating or deleting a key using the same name after it has already been deleted.

Backing Up Encryption Keys

It is very important that you regularly back up your encryption keys. Failure to do so can result in irretrievable loss of encrypted data.

If you are using the Java KeyStore KMS, make sure you regularly back up the Java KeyStore that stores the encryption keys.

Rolling Encryption Keys

When you roll an EZ key, you are essentially creating a new version of the key (ezKeyVersionName). Rolling EZ keys regularly helps enterprises minimize the risk of key exposure. If a malicious attacker were to obtain the EZ key and decrypt encrypted data encryption keys (EDEKs) into DEKs, they could gain the ability to decrypt HDFS files. Rolling an EZ key ensures that all DEKs for newly-created files will be encrypted with the new version of the EZ key. The older EZ key version that the attacker obtained cannot decrypt these EDEKs.

Before you begin

Before attempting to roll an encryption key (also known as an encryption zone key, or EZ key),you must be familiar with the concepts associated with Navigator Data Encryption and the HDFS Transparent Encryption.

About this task

You may want to roll the encryption key periodically, as part of your security policy or when an external security compromise is detected.

Procedure

1. Before rolling any keys, log in as HDFS Superuser and verify/identify the encryption zones to which the current key applies.

   This operation also helps clarify the relationship between the EZ key and encryption zones, and, if necessary, makes it easier to identify more important, high priority zones.

   ```
   $ hdfs crypto –listZones
   /ez key1
   /ez2 key2
   /user key1
   ```

   The first column identifies the encryption zone paths; the second column identifies the encryption key name.
2. You can verify that the files inside an encryption zone are encrypted using the hdfs crypto -getFileEncryptionInfo command.

   Note the EZ key version name and value, which you can use for comparison and verification after rolling the EZ key.

   ```
   $ hdfs crypto –getFileEncryptionInfo –path /ez/f
   {cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. crypto
   ProtocolVersion:
   CryptoProtocolVersion{description=’Encryption zones’, version=2, unknow
   nValue=null}, edek: 373c0c2e919c27e58c1c343f54233cbd,
   iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
    7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTXl0bXCpF}
   ```
3. Log off as HDFS Superuser and log in as Key Administrator.

   Because keys can be rolled, a key can have multiple key versions, where each key version has its own key material (the actual secret bytes used during DEK encryption and EDEK decryption). You can fetch an encryption key by either its key name, returning the latest version of the key, or by a specific key version.

   Roll the encryption key (previously identified/confirmed by the HDFS Superuser in step 1. Here, the <key name> is `key1`.

   ```
   hadoop key roll key1
   ```

   This operation contacts the KMS and rolls the keys there. Note that this can take a considerable amount of time, depending on the number of key versions residing in the KMS.

   ```
   Rolling key version from KeyProvider: org.apache.hadoop.crypto.key.kms.L
   oadBalancingKMSClientProvider@5ea434c8
   for keyName: key1
   ```

```
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8
 has been updated.
```

4. Log out as Key Administrator, and log in as HDFS Superuser. Verify that new files in the encryption zone have a new EZ key version.

Note: For performance reasons, the NameNode caches EDEKs, so after rolling an encryption key, you may not be able to see the new version encryption key immediately, or at least until after the EDEK cache is consumed. Of course, the file decryption and encryption still works with these EDEKs. If you require that all files' DEKs in an encryption zone are encrypted using the latest version encryption key, please re-encrypt the EDEKs.

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/new_file
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. cryptoP
rotocolVersion:
CryptoProtocolVersion{description=’Encryption zones’, version=2, unknown
Value=null}, edek: 9aa13ea4a700f96287cfe1349f6ff4f2,
iv: 465c878ad9325e42fa460d2a22d12a72, keyName: key1, ezKeyVersionName:
 4tuvorJ6Feeqk8WiCfdDs9K32KuEj7g2ydCAv0gNQbY}
```

Alternatively, you can use KMS Rest API to view key metadata and key versions. Elements appearing in brackets should be replaced with your actual values. So in this case, before rolling a key, you can view the key metadata and versions as follows:

```
$ curl -k --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-
name>/_metadata"
{
  "name" : "<key-name>",
  "cipher" : "<cipher>",
  "length" : <length>,
  "description" : "<decription>",
  "created" : <millis-epoc>,
  "versions" : <versions> (For example, 1)
}
$ curl -k  --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-
name>/_currentversion"
{
  "material" : "<material>",
  "name" : "<key-name>",
  "versionName" : "<versionName>" (For example, version 1)
}
```

Roll the key and compare the results:

```
$ hadoop key roll key1

Rolling key version from KeyProvider: KMSClientProvider[https://<KMS_HOS
TNAME>:16000/kms/v1/]

  for key name: <key-name>

key1 has been successfully rolled.

KMSClientProvider[https://<KMS_HOSTNAME>/kms/v1/] has been updated.

$ curl -k  --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-
name>/_currentversion"
{
  "material" : "<material>", (New material)
  "name" : "<key-name>",
```

```
  "versionName" : "<versionName>" (New version name. For example, version
 2)
}

$ curl -k --negotiate -u: "https://<KMS_HOSTNAME>:16000/kms/v1/key/<key-
name>/_metadata"
{
  "name" : "<key-name>",
  "cipher" : "<cipher>",
  "length" : <length>,
  "description" : "<decription>",
  "created" : <millis-epoc>,
  "versions" : <versions> (For example, version 2)
}
```

Deleting Encryption Zone Keys

For information on how to delete encryption zone keys, see Delete a Key

Re-encrypting Encrypted Data Encryption Keys (EDEKs)

When you re-encrypt an EDEK, you are essentially decrypting the original EDEK created by the DEK, and then re-encrypting it using the new (rolled) version of the EZ key. The file's metadata, which is stored in the NameNode, is then updated with this new EDEK. Re-encryption does not impact the data in the HDFS files or the DEK–the same DEK is still used to decrypt the file, so re-encryption is essentially transparent.

Related Information

Rolling Encryption Keys

Benefits and Capabilities

In addition to minimizing security risks, re-encrypting the EDEK offers other capabilities and benefits.

- Re-encrypting EDEKs does not require that the user explicitly re-encrypt HDFS files.
- In cases where there are several zones using the same key, the Key Administrator has the option of selecting which zone’s EDEKs are re-encrypted first.
- The HDFS Superuser can also monitor and cancel re-encryption operations.
- Re-encryption is restarted automatically in cases where you have a NameNode failure during the re-encryption operation.

Prerequisites and Assumptions

There are certain considerations that you must be aware of as you re-encrypt an EDEK.

- It is recommended that you perform EDEK re-encryption at the same time that you perform regular cluster maintenance because the operation can adversely impact CPU resources on the NameNode.
- In Cloudera Manager, review the cluster’s NameNode status, which must be in “Good Health”. If the cluster NameNode does not have a status of “Good Health”, then do not proceed with the re-encryption of the EDEK. In the Cloudera Manager WebUI menu, you can verify the status for the cluster NameNode, which must not be in Safe mode (in other words, the WebUI should indicate “Safemode is off”).

Running the re-encryption command without successfully verifying the preceding items will result in failures with errors.

Limitations

There are few limitations associated with the re-encryption of EDEKs.

Caution: You cannot perform any rename operations within the encryption zone during re-encryption. If you attempt to perform a rename operation during EDEK re-encryption, you will receive an IOException error.

EDEK re-encryption does not change EDEKs on snapshots, due to the immutable nature HDFS snapshots. Thus, you should be aware that after EZ key exposure, the Key Administrator must delete snapshots.

Re-encrypting an EDEK

This scenario operates on the assumption that an encryption zone has already been set up for this cluster.

Procedure

1. Navigate to the cluster in which you will be rolling keys and re-encrypting the EDEK.
2. Log in as HDFS Superuser.
3. View all of the options for the hdfs crypto command:

```
$ hdfs crypto
[-createZone –keyName <keyName> -path <path>]
[-listZones]
[-provisionTrash –path <path>]
[-getFileEncryptionInfo –path <path>]
[-reencryptZone <action> -path <zone>]
[-listReencryptionStatus]
[-help <command-name>]
```

4. Before rolling any keys, verify/identify the encryption zones to which the current key applies.

   This operation also helps clarify the relationship between the EZ key and encryption zones, and, if necessary, makes it easier to identify more important, high priority zones.

```
$ hdfs crypto –listZones
/ez     key1
```

   The first column identifies the encryption zone path (/ez); the second column identifies the encryption key name (key1).
5. Exit from the HDFS Superuser account and log in as Key Administrator.
6. Roll the encryption key (previously identified/confirmed by the HDFS Superuser in step 4).

   Here, the <key name> is `key1`

```
hadoop key roll key1
```

   This operation contacts the KMS and rolls the keys. Note that this can take a considerable amount of time, depending on the number of key versions.

```
Rolling key version from KeyProvider: org.apache.hadoop.crypto.key.kms.L
oadBalancingKMSClientProvider@5ea434c8
for keyName: key1
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSClientProvider@5ea434c8
 has been updated.
```

7. Log out as Key Administrator, and log in as HDFS Superuser.
8. Before performing the re-encryption, you can verify the status of the current key version being used (keyName).

   Then, after re-encrypting, you can confirm that the EZ key version (ezKeyVersionName) and EDEK have changed.

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/f
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. crypto
ProtocolVersion:
CryptoProtocolVersion{description='Encryption zones', version=2, unknow
nValue=null}, edek: 9aa13ea4a700f96287cfe1349f6ff4f2,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
 7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTXl0bXCpF}
```

9. After the EZ key has been rolled successfully, re-encrypt the EDEK by running the re-encryption command on the encryption zone:

```
$ hdfs crypto –reencryptZone –start –path /ez
```

The following information appears when the submission is complete. At this point, the NameNode is processing and re-encrypting all of the EDEKs under the /ez directory.

```
re-encrypt command successfully submitted for zone: /ez action:  START:
```

Depending on the number of files, the re-encryption operation can take a long time. Re-encrypting a 1M EDEK file typically takes between 2-6 minutes, depending on the NameNode hardware. To check the status of the re-encryption for the zone:

```
hdfs crypto –listReencryptionStatus
```

Column Name	Description	Sample Data
ZoneName	The encryption zone name	/ez
Status	• Submitted: the command is received, but not yet being processed by the NameNode. • Processing: the zone is being processed by the NameNode. • Completed: the NameNode has finished processing the zone, and every file in the zone has been re-encrypted.	Completed
EZKey Version Name	The encryption zone key version name, which used for re-encryption comparison. After re-encryption is complete, all files in the encryption zone are guaranteed to have an EDEK whose encryption zone key version is at least equal to this version.	ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz
Submission Time	The time at which the re-encryption operation commenced.	2017-09-07 10:01:09,262-0700
Is Canceled?	True: the encryption operation has been canceled. False: the encryption operation has not been canceled.	False
Completion Time	The time at which the re-encryption operation completed.	2017-09-07 10:01:10,441-0700
Number of files re-encrypted	The number that appears in this column reflects only the files whose EDEKs have been updated. If a file is created after the key is rolled, then it will already have an EDEK that has been encrypted by the new key version, so the re-encryption operation will skip that file. In other words, it's possible for a "Completed" re-encryption to reflect a number of re-encrypted files that is less than the number of files actually in the encryption zone. Note: In cases when you re-encrypt an EZ key that has already been re-encrypted and there are no new files, the number of files re-encrypted will be 0.	1

Column Name	Description	Sample Data
Number of failures	When 0, no errors occurred during the re-encryption operation. If larger than 0, then investigate the NameNode log, and re-encrypt.	0
Last file Checkpointed	Identifies the current position of the re-encryption process in the encryption zone--in other words, the file that was most recently re-encrypted.	0

10. After the re-encryption completes, you can confirm that the EDEK and EZ Key Version Name values have changed.

```
$ hdfs crypto –getFileEncryptionInfo –path /ez/f
{cipherSuite: {name: AES/CTR/NoPadding, algorithmBlockSize: 16}. crypto
ProtocolVersion:
CryptoProtocolVersion{description='Encryption zones', version=2, unknow
nValue=null}, edek: 373c0c2e919c27e58c1c343f54233cbd,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
 ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRTOHS3YjCz }
```

Managing Re-encryption Operations

There are various facets of the EDEK re-encryption process such as cancelling re-encryption, rolling keys during a re-encryption operation, and throttling re-encryption operations.

Cancelling Re-encryption

Only users with the HDFS Superuser privilege can cancel the EDEK re-encryption after the operation has started.

To cancel a re-encryption:

```
hadoop crypto -reencryptZone cancel -path <zone>
```

Rolling Keys During a Re-encryption Operation

While it is not recommended, it is possible to roll the encryption zone key version on the KMS while a re-encryption of that encryption zone is already in progress in the NameNode. The re-encryption is guaranteed to complete with all DEKs re-encrypted, with a key version equal to or later than the encryption zone key version when the re-encryption command was submitted. This means that, if initially the key version is rolled from v0 to v1, then a re-encryption command was submitted. If later on the KMS the key version is rolled again to v2, then all EDEKs will be at least re-encrypted to v1. To ensure that all EDEKs are re-encrypted to v2, submit another re-encryption command for the encryption zone.

Rolling keys during re-encryption is not recommended because of the potential negative impact on key management operations. Due to the asynchronous nature of re-encryption, there is no guarantee of when, exactly, the rolled encryption keys will take effect. Re-encryption can only guarantee that all EDEKs are re-encrypted at least on the EZ key version that existed when the re-encryption command is issued.

Throttling Re-encryption Operations

With the default operation settings, you will not typically need to throttle re-encryption operations. However, in cases of excessive performance impact due to the re-encryption of large numbers of files, advanced users have the option of throttling the operation so that the impact on the HDFS NameNode and KT KMS are minimized.

Specifically, you can throttle the operation to control the rate of the following:

- The number of EDEKs that the NameNode should send to the KMS to re-encrypt in a batch (dfs.namenode.reencrypt.batch.size)
- The number of threads in the NameNode that can run concurrently to contact the KMS. (dfs.namenode.reencrypt.edek.threads)

- Percentage of time the NameNode read-lock should be held by the re-encryption thread (dfs.namenode.reencrypt.throttle.limit.handler.ratio)
- Percentage of time the NameNode write-lock should be held by the re-encryption thread (dfs.namenode.reencrypt.throttle.limit.updater.ratio)

You can monitor the HDFS NameNode heap and CPU usage from Cloudera Manager.

Securing the Key Management System (KMS)

Cloudera provides the following Key Mangement System (KMS) implementations: Ranger KMS with database, Ranger KMS with HSM, Ranger KMS with Key Trustee Server, and Ranger KMS with Key Trustee Server and Key HSM. You can secure Ranger KMS using Kerberos, TLS/SSL communication, and access control lists (ACLs) for operations on encryption keys.

Cloudera Manager supports wizard-driven instructions for installing both Ranger KMS with a database and Ranger KMS with KTS.

Enabling Kerberos Authentication for the KMS

You can use Cloudera Manager to enable Kerberos authentication for the KMS.

About this task

Minimum Required Role: Full Administrator

Procedure

1. Open the Cloudera Manager Admin Console and go to the KMS service.
2. Click Configuration.
3. Set the Authentication Type property to kerberos.
4. Click Save Changes.
5. Because Cloudera Manager does not automatically create the principal and keytab file for the KMS, you must run the Generate Credentials command manually.

   On the top navigation bar, go to Administration Security Kerberos Credentials and click Generate Missing Credentials

   Note: This does not create a new Kerberos principal if an existing HTTP principal exists for the KMS host.
6. Return to the home page by clicking the Cloudera Manager logo.
7. Click the ⏻ icon that is next to any stale services to invoke the cluster restart wizard.
8. Click Restart Stale Services.
9. Click Restart Now.
10. Click Finish.

Configuring TLS/SSL for the KMS

You must configure specific TLS/SSL properties associated with the KMS.

About this task

Minimum Required Role: Configurator (also provided by Cluster Administrator, Full Administrator)

Procedure

1. Go to the KMS service.
2. Click Configuration.

3. In the Search field, type TLS/SSL to show the KMS TLS/SSL properties (in the Key Management Server Default Group Security category).
4. Edit the following TLS/SSL properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Key Management Server	Encrypt communication between clients and Key Management Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (TLS/SSL)).
Key Management Server TLS/SSL Server JKS Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Key Management Server is acting as a TLS/SSL server. The keystore must be in JKS format.
Key Management Server TLS/SSL Server JKS Keystore File Password	The password for the Key Management Server JKS keystore file.
Key Management Server Proxy TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Key Management Server Proxy might connect to. This is used when Key Management Server Proxy is the client in a TLS/SSL connection. This truststore must contain the certificates used to sign the services connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Key Management Server Proxy TLS/SSL Certificate Trust Store Password	The password for the Key Management Server Proxy TLS/SSL Certificate Trust Store File. This password is not required to access the truststore; this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

5. Click Save Changes.
6. Return to the home page by clicking the Cloudera Manager logo.
7. Click the ⏻ icon that is next to any stale services to invoke the cluster restart wizard.
8. Click Restart Stale Services.
9. Click Restart Now.
10. Click Finish.

Migrating Keys from a Java KeyStore to Cloudera Navigator Key Trustee Server

You can migrate keys from an existing Java KeyStore (JKS) to Key Trustee Server to improve security, durability, and scalability.

Before you begin

This procedure assumes that the Java KeyStore (JKS) is on the same host as the new Key Trustee KMS service.

Procedure

1. Stop the Java KeyStore KMS service.
2. Add and configure the Key Trustee KMS service, and configure HDFS to use it for its KMS Service setting.
3. Restart the HDFS service and redeploy client configuration for this to take effect.
   a. Home Cluster-wide Deploy Client Configuration
4. Add the following to the Key Management Server Proxy Advanced Configuration Snippet (Safety Valve) for kms-site.xml (Key Trustee KMS Service Configuration Category Advanced):

```
<property>
  <name>hadoop.kms.key.provider.uri</name>
  <value>keytrustee://file@/var/lib/kms-keytrustee/keytrustee/.keytrustee/
,jceks://file@/PATH/TO/KMS.KEYSTORE</value>
  <description>URI of the backing KeyProvider for the KMS</description>
</property>
```

```
<property>
  <name>hadoop.security.keystore.java-keystore-provider.password-file</
name>
  <value>/TMP/PASSWORD.TXT</value>
  <description>Java KeyStore password file</description>
</property>
```

If the Java KeyStore is not password protected, omit the hadoop.security.keystore.java-keystore-provider.password-file property.

5. Click Save Changes and restart the Key Trustee KMS service.

   If the Java KeyStore is not password protected, skip to step 7.

6. Create the file /var/lib/keytrustee-kms/jetty-deployment/webapps/kms/WEB-INF/classes/*TMP/PASSWORD.TXT* and add the Java KeyStore password to it.

7. Change the ownership of /var/lib/keytrustee-kms/jetty-deployment/webapps/kms/WEB-INF/classes/*TMP/PASSWORD.TXT* to kms:kms.

   ```
   sudo chown kms:kms /var/lib/keytrustee-kms/jetty-deployment/webapps/kms/
   WEB-INF/classes/TMP/PASSWORD.TXT
   ```

8. From the host running the Key Trustee KMS service, if you have not configured Kerberos and TLS/SSL, run the following command:

   ```
   curl -L -d "trusteeOp=migrate"
   "http://KMS01.EXAMPLE.COM:16000/kms/v1/trustee/key/migrate?user.n
   ame=USERNAME&trusteeOp=migrate"
   ```

   If you have configured Kerberos and TLS/SSL, use the following command instead:

   ```
   curl --negotiate -u : -L -d "trusteeOp=migrate"
   "https://KMS01.EXAMPLE.COM:16000/kms/v1/trustee/key/migrate?user.n
   ame=USERNAME&trusteeOp=migrate" --cacert /PATH/TO/KMS/CERT
   ```

9. Monitor /var/log/kms-keytrustee/kms.log and /var/log/kms-keytrustee/kms-catalina.<*DATE*>.log to verify that the migration is successful.

   You can also run sudo -u <key_admin> hadoop key list to verify that the keys are listed.

10. After you have verified that the migration is successful, remove the safety valve entry used in step 3 and restart the Key Trustee KMS service.

Related Information

Role Instances

Migrating Ranger Key Management Server Role Instances to a New Host

You can move the Ranger Admin, Ranger KMS db and Ranger KMS KTS role instances for an existing Ranger KMS service from one host to another, using Cloudera Manager.

Note: This procedure applies only to the Ranger Key Management Server role instances. Do not attempt to move the Key Trustee Server.

In some cases–for example, after upgrading your servers–you may want to migrate a Ranger KMS Server role instance to a new host. This procedure describes how to move a Ranger KMS role instance from an existing cluster host to another cluster host.

Migrate the Ranger Admin role instance to a new host

To migrate the Ranger KMS role instances to a new host, first migrate the Ranger Admin role instance.

Procedure

1. Add a new Ranger Admin role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger Admin role instance.
3. Stop the initial Ranger Admin instance.
4. Delete the initial Ranger Admin instance.
5. Restart the cluster.

   Restarting the cluster removes the "stale" changes.

Migrate the Ranger KMS db role instance to a new host

After migrating the Ranger Admin role instance to a new host, migrate the Ranger KMS db role instance.

About this task

Only if Ranger KMS has a backend database for key storage, should you migrate the Ranger KMS db role instance.

Procedure

1. Add a new Ranger KMS db role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger KMS db role instance.
3. Stop the initial Ranger KMS db instance.
4. Delete the initial Ranger KMS db instance.
5. Restart the cluster.
6. Login to Ranger Admin UI using keyadmin credentials.
7. Update the cm_kms service to use the kms url that refers to the new hostname.

Related Information

Managing Hosts

Migrate the Ranger KMS KTS role instance to a new host

After migrating the Ranger Admin, Ramger KMS db role instances to a new host, migrate the Ranger KMS KTS role instance.

About this task

Only if Ranger KMS is backed by Key Trustee Server for key storage, should you migrate the Ranger KMS KTS role instance.

Procedure

1. Add a new Ranger KMS KTS role instance on another node.

   Note: If you enabled manual SSL on this cluster, you must update the SSL configs when adding a new role.

2. Start the new Ranger KMS KTS role instance.
3. Stop the initial Ranger KTS service.
4. Delete the older Ranger KTS instance.

5. Restart the cluster.
6. Login to Ranger Admin UI using keyadmin credentials.
7. Update the cm_kms service to use the kms url that refers to the new hostname.
8. Copy or rsync conf and gpg files such as keytrustee.conf, pubring.gpp and secring.gpg present at /var/lib/kms-keytrustee/keytrustee/.keytrustee/ from older host to new host. For example:

```
[root@mm-ktslog-1 ~]# ll /var/lib/kms-keytrustee/keytrustee/.keytrustee/
total 20
-rw------- 1 kms kms  715 Oct  7 10:59 keytrustee.conf
-rw------- 1 kms kms 5026 Oct  7 10:59 pubring.gpg
-rw------- 1 kms kms 4885 Oct  7 10:59 secring.gpg
```

9. Restart the Ranger KMS KTS service.

Related Information

Managing Hosts

Migrating ACLs from Key Trustee KMS to Ranger KMS

You must perform the following procedures to migrate ACLs from Key Trustee Key Management Server (KMS) to Ranger KMS.

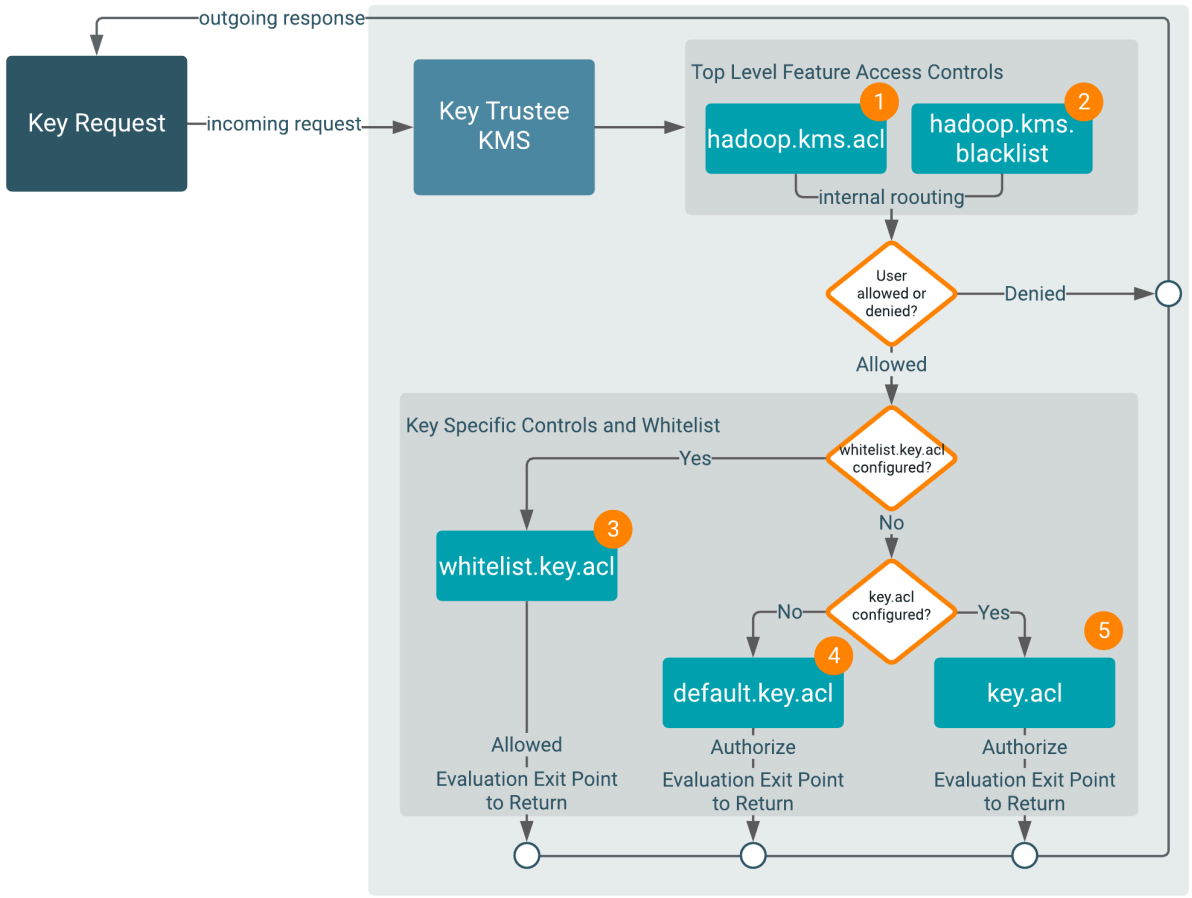
Key Trustee ACL evaluation

Before going into the details of how Key Trustee ACLs are evaluated, it is critical that you understand the key rules that the Key Trustee Key Management Server uses in performing this evaluation.

KMS ACL Flow Rules:

- The whitelist class bypasses key.acl and default.key.acl controls.
- The key.acl definitions override all default definitions.

Encryption key access is evaluated as follows:



1 and 2

The KMS evaluates the hadoop.kms.acl.<OPERATION> and hadoop.kms.blacklist.<OPERATION> classes to determine whether or not access to a specific KMS feature or function is authorized.

In other words, a user must be allowed by hadoop.kms.acl.<OPERATION>, and not be disallowed by hadoop.kms.blacklist.<OPERATION>.

If a user is denied access to a KMS-wide operation, then the flow halts and returns the result Denied.

If a user is allowed access to a KMS-wide operation, then the evaluation flow proceeds.

3

The KMS evaluates the whitelist.key.acl class.

The KMS ACL workflow evaluates the whitelist.key.acl.<OPERATION>, and if the user is allowed access, then it is granted (Allowed) . If not, then the flow continues with the evaluation.

4 and 5

The KMS evaluates the default.key.acl.<OPERATION> and key.acl.<OPERATION> classes.

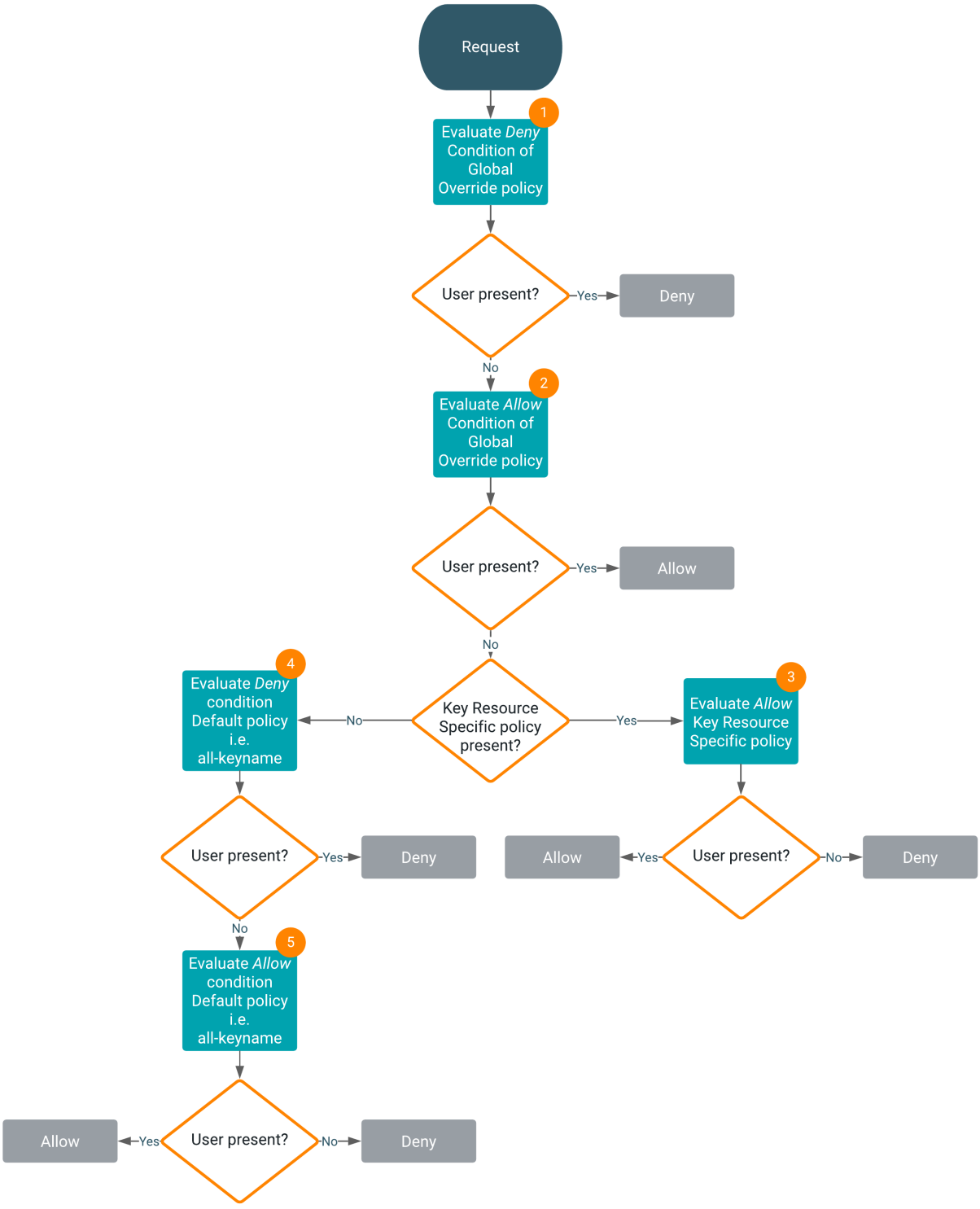
The KMS evaluates whether or not there is a key.acl.KEY.<OPERATION> class that matches the action the user is attempting to perform. If there is, it then evaluates that value to determine whether or not the user can perform the requested operation.

Note: Before evaulating the default.key.acl.<OPERATION> and key.acl.<OPERATION> classes, the flow logic determines which classes exist. Only one of these can exist and be used at any time (for example, key.acl.prodkey.READ overrides default.key.acl.READ for prodkey, so the flow logic is configured with it's own READ ACLs)

Depending on the result of the Key Trustee ACL evaluation, controls are applied to the key and results (Allowed or Denied).

Access evaluation with Ranger KMS policies

Access is evaluated with Ranger KMS policies as follows:



1

After the request is received, the Deny condition of the Global Override policy is evaluated. If the user is present, the flow halts and returns the result Deny. If the user is not present, the evaluation flow proceeds.

2

Now, the Allow condition of the Global Override policy is evaluated. If the user is present, the flow halts and returns the result Allow. If the user is not present, the evaluation flow proceeds.

3

If the Key Resource Specific policy is present, the Allow condition of the Key Resource Specific policy is evaluated. If the user is not present, the flow halts and returns the result Deny. If the user is present, the flow is complete and returns the result Allow.

4

If the Key Resource Specific policy is not present, the Deny condition of the Default policy, all-keyname, is evaluated. If the user is present, the flow halts and returns the result Deny. If the user is not present, the evaluation flow proceeds.

5

Now, the Allow condition of the Default policy, all-keyname, is evaluated. If the user is not present, the flow halts and returns the result Deny. If the user is present, the flow is complete and returns the result Allow.

Key Trustee KMS operations not supported by Ranger KMS

The following Key Trustee KMS operations are not supported by Ranger KMS.

- hadoop.kms.acl.<OPERATION>

  The ACLs mentioned below are ignored by Ranger KMS because these ACLs are not migrated to the Ranger KMS policy.

  ```
  hadoop.kms.acl.CREATE
  hadoop.kms.acl.DELETE
  hadoop.kms.acl.ROLLOVER
  hadoop.kms.acl.GET
  hadoop.kms.acl.GET_KEYS
  hadoop.kms.acl.GET_METADATA
  hadoop.kms.acl.SET_KEY_MATERIAL
  hadoop.kms.acl.GENERATE_EEK
  hadoop.kms.acl.DECRYPT_EEK
  ```

- keytrustee.kms.acl.<OPERATION>

  The ACLs mentioned below are Key Trustee-specific ACLs. These ACLs are ignored by Ranger KMS because they are not migrated to the Ranger KMS policy. Also, these ACLs are not supported by Hadoop KMS.

  ```
  keytrustee.kms.acl.UNDELETE
  keytrustee.kms.acl.PURGE
  ```

Note: The KTS to Ranger KMS migration utility may exit with output similar to:

```
Following users do not exist in Ranger DB: [csso_xxxx, user01, csso_xxx2,
 ser2_old]
Following groups do not exist in Ranger DB: [usergroup1, usergroup2,
 unknown_test_group, usergroup_old]
To fix this problem, either add the users/groups to your user management
 system and re-sync the users/groups,
or, create the listed users/groups in Ranger, using the Ranger Admin Web
 UI: https://<servername>.root.hwx.site:6182/
```

The workaround is to add the required users and groups in the ranger database, as internal users and groups, using the Ranger Admin Web UI, then resume the upgrade process.

ACLs supported by Ranger KMS and Ranger KMS Mapping

The following ACLs are supported by Ranger KMS and Ranger KMS mapping.

- whitelist.key.acl.<operation> and hadoop.kms.blacklist.<Operation>

  In this case, you create a Global Override policy under the service cm_kms.

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Global Override Policy	*	Override	whitelist.key.acl.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			whitelist.key.acl.GENERATE_EEK	ALLOW	GENERATE_EEK
			whitelist.key.acl.DECRYPT_EEK	ALLOW	DECRYPT_EEK
			whitelist.key.acl.READ	ALLOW	GET, GET KEYS, GET METADATA
			hadoop.kms.blacklist.CREATE	DENY	CREATE
			hadoop.kms.blacklist.DELETE	DENY	DELETE
			hadoop.kms.blacklist.ROLLOVER	DENY	ROLLOVER
			hadoop.kms.blacklist.GET	DENY	GET
			hadoop.kms.blacklist.GET_KEYS	DENY	GET KEYS
			hadoop.kms.blacklist.GET_METADATA	DENY	GET METADATA
			hadoop.kms.blacklist.SET_KEY_MATERIAL	DENY	SET KEY MATERIAL
			hadoop.kms.blacklist.GENERATE_EEK	DENY	GENERATE_EEK
			hadoop.kms.blacklist.DECRYPT_EEK	DENY	DECRYPT_EEK

- default.key.acl.<operation>

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Default Policy all-keyname	*	Normal	default.key.acl.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			default.key.acl.GENERATE_EEK	ALLOW	GENERATE_EEK
			default.key.acl.DECRYPT_EEK	ALLOW	DECRYPT_EEK
			default.key.acl.READ	ALLOW	GET, GET KEYS, GET METADATA

- key.acl.<key-name>.<OPERATION> Key Specific ACL

  In this case, you create a Key Resource Specific policy under the service cm_kms.

  Service : cm_kms

Policy	Key-resource	Priority	Key Trustee ACL	Ranger Policy Condition	Ranger Policy Permission
Key Resource Specific policy <keyname>	<keyname>	Normal	key.acl.<key-name>.MANAGEMENT	ALLOW	CREATE, DELETE, ROLLOVER
			key.acl.<key-name>.GENERATE_EEK	ALLOW	GENERATE_EEK
			key.acl.<key-name>.DECRYPT_EEK	ALLOW	DECRYPT_EEK
			key.acl.<key-name>.READ	ALLOW	GET, GET KEYS, GET METADATA
			key.acl.<key-name>.ALL	ALLOW	SELECT ALL

Note: In Key Resource Specific policies, DENY ALL OTHER ACCESS flags are set to true.

Configuring CDP Services for HDFS Encryption

There are recommendations that you must consider for setting up HDFS Transparent Encryption with various CDP services.

Important: Encrypting /tmp using HDFS encryption is not supported.

Note: Ranger audit directories in HDFS are owned by specific service users. In case of HDFS, it will be the HDFS service user. Typically, HDFS service user is blacklisted for DECRYPT_EEK operation. Hence, enabling encryption of ranger audit directories in HDFS is not recommended.

Transparent Encryption Recommendations for HBase

Make /hbase an encryption zone. Do not create encryption zones as subdirectories under /hbase, because HBase may need to rename files across those subdirectories. When you create the encryption zone, name the key hbase-key to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without HBase currently installed, create the /hbase directory and make that an encryption zone.

On a cluster with HBase already installed, perform the following steps:

1. Stop the HBase service.
2. Move data from the /hbase directory to /hbase-tmp.
3. Create an empty /hbase directory and make it an encryption zone.
4. Distcp all data from /hbase-tmp to /hbase, preserving user-group permissions and extended attributes.
5. Start the HBase service and verify that it is working as expected.
6. Remove the /hbase-tmp directory.

KMS ACL Configuration for HBase

In the KMS ACL, grant the hbase user and group DECRYPT_EEK permission for the HBase key:

```
<property>
  <name>key.acl.hbase-key.DECRYPT_EEK</name>
  <value>hbase hbase</value>
  </description>
</property>
```

Transparent Encryption Recommendations for Hive

HDFS encryption has been designed so that files cannot be moved from one encryption zone to another or from encryption zones to unencrypted directories. Therefore, the landing zone for data when using the LOAD DATA IN PATH command must always be inside the destination encryption zone.

To use HDFS encryption with Hive, ensure you are using one of the following configurations:

Single Encryption Zone

With this configuration, you can use HDFS encryption by having all Hive data inside the same encryption zone. In Cloudera Manager, configure the Hive Scratch Directory (hive.exec.scratchdir) to be inside the encryption zone.

Recommended HDFS Path: /warehouse/tablespace/

To use the auto-generated KMS ACL, make sure you name the encryption key hive-key.

For example, to configure a single encryption zone for the entire Hive warehouse, you can rename /warehouse/table space/ to /warehouse/tablespace-old, create an encryption zone at /warehouse/tablespace/, and then distcp all the data from /warehouse/tablespace-old to /warehouse/tablespace/.

In Cloudera Manager, configure the Hive Scratch Directory (hive.exec.scratchdir) to be inside the encryption zone by setting it to /warehouse/tablespace/tmp, ensuring that permissions are 1777 on /warehouse/tablespace/tmp.

Multiple Encryption Zones

With this configuration, you can use encrypted databases or tables with different encryption keys. To read data from read-only encrypted tables, users must have access to a temporary directory that is encrypted at least as strongly as the table.

For example:

1. Configure two encrypted tables, ezTbl1 and ezTbl2.
2. Create two new encryption zones, /data/ezTbl1 and /data/ezTbl2.
3. Load data to the tables in Hive using LOAD statements.

Other Encrypted Directories

- LOCALSCRATCHDIR: The MapJoin optimization in Hive writes HDFS tables to a local directory and then uploads them to the distributed cache. To ensure these files are encrypted, either disable MapJoin by setting hive .auto.convert.join to false, or encrypt the local Hive Scratch directory (hive.exec.local.scratchdir) using Cloudera Navigator Encrypt.
- DOWNLOADED_RESOURCES_DIR: JARs that are added to a user session and stored in HDFS are downloaded to hive.downloaded.resources.dir on the HiveServer2 local filesystem. To encrypt these JAR files, configure Cloudera Navigator Encrypt to encrypt the directory specified by hive.downloaded.resources.dir.
- NodeManager Local Directory List: Hive stores JARs and MapJoin files in the distributed cache. To use MapJoin or encrypt JARs and other resource files, the yarn.nodemanager.local-dirs YARN configuration property must be configured to a set of encrypted local directories on all nodes.

Changed Behavior after HDFS Encryption is Enabled

You must consider various factors when working with Hive tables after enabling HDFS transparent encryption for Hive.

- Loading data from one encryption zone to another results in a copy of the data. Distcp is used to speed up the process if the size of the files being copied is higher than the value specified by HIVE_EXEC_COPYFILE_MAXSIZE. The minimum size limit for HIVE_EXEC_COPYFILE_MAXSIZE is 32 MB, which you can modify by changing the value for the hive.exec.copyfile.maxsize configuration property.
- When loading data to encrypted tables, Cloudera strongly recommends using a landing zone inside the same encryption zone as the table.
  - Example 1: Loading unencrypted data to an encrypted table - Use one of the following methods:
    - If you are loading new unencrypted data to an encrypted table, use the LOAD DATA ... statement. Because the source data is not inside the encryption zone, the LOAD statement results in a copy. For this reason, Cloudera recommends landing data that you need to encrypt inside the destination encryption zone. You can use distcp to speed up the copying process if your data is inside HDFS.
    - If the data to be loaded is already inside a Hive table, you can create a new table with a LOCATION inside an encryption zone as follows:

      ```
      CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FR
      OM <unencrypted_table>
      ```

      The location specified in the CREATE TABLE statement must be inside an encryption zone. Creating a table pointing LOCATION to an unencrypted directory does not encrypt your source data. You must copy your data to an encryption zone, and then point LOCATION to that zone.
  - Example 2: Loading encrypted data to an encrypted table - If the data is already encrypted, use the CREATE TABLE statement pointing LOCATION to the encrypted source directory containing the data. This is the fastest way to create encrypted tables.

    ```
    CREATE TABLE encrypted_table [STORED AS] LOCATION ... AS SELECT * FROM <
    encrypted_source_directory>
    ```
- Users reading data from encrypted tables that are read-only must have access to a temporary directory which is encrypted with at least as strong encryption as the table.
- Temporary data is now written to a directory named .hive-staging in each table or partition
- Previously, an INSERT OVERWRITE on a partitioned table inherited permissions for new data from the existing partition directory. With encryption enabled, permissions are inherited from the table.

KMS ACL Configuration for Hive

When Hive joins tables, it compares the encryption key strength for each table. For this operation to succeed, you must configure the KMS ACL to allow the hive user and group READ access to the Hive key.

```
<property>
  <name>key.acl.hive-key.READ</name>
  <value>hive hive</value>
</property>
```

If you have restricted access to the GET_METADATA operation, you must grant permission for it to the hive user or group:

```
<property>
  <name>hadoop.kms.acl.GET_METADATA</name>
  <value>hive hive</value>
</property>
```

If you have disabled HiveServer2 Impersonation, you must configure the KMS ACLs to grant DECRYPT_EEK permissions to the hive user, as well as any user accessing data in the Hive warehouse.

Cloudera recommends creating a group containing all Hive users, and granting DECRYPT_EEK access to that group.

For example, suppose user jdoe (home directory /user/jdoe) is a Hive user and a member of the group hive-users. The encryption zone (EZ) key for /user/jdoe is named jdoe-key, and the EZ key for /user/hive is hive-key. The following ACL example demonstrates the required permissions:

```
<property>
  <name>key.acl.hive-key.DECRYPT_EEK</name>
  <value>hive hive-users</value>
</property>
<property>
  <name>key.acl.jdoe-key.DECRYPT_EEK</name>
  <value>jdoe,hive</value>
</property>
```

If you have enabled HiveServer2 impersonation, data is accessed by the user submitting the query or job, and the user account (jdoe in this example) may still need to access data in their home directory. In this scenario, the required permissions are as follows:

```
<property>
  <name>key.acl.hive-key.DECRYPT_EEK</name>
  <value>nobody hive-users</value>
</property>

<property>
  <name>key.acl.jdoe-key.DECRYPT_EEK</name>
  <value>jdoe</value>
</property>
```

Transparent Encryption Recommendations for Hue

Make /user/hue an encryption zone because Oozie workflows and other Hue-specific data are stored there by default. When you create the encryption zone, name the key hue-key to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without Hue currently installed, create the /user/hue directory and make it an encryption zone.

On a cluster with Hue already installed:

1. Create an empty /user/hue-tmp directory.
2. Make /user/hue-tmp an encryption zone.
3. DistCp all data from /user/hue into /user/hue-tmp.
4. Remove /user/hue and rename /user/hue-tmp to /user/hue.

KMS ACL Configuration for Hue

In the KMS ACLs, grant the hue and oozie users and groups DECRYPT_EEK permission for the Hue key:

```
<property>
  <name>key.acl.hue-key.DECRYPT_EEK</name>
  <value>oozie,hue oozie,hue</value>
</property>
```

Transparent Encryption Recommendations for Impala

There are various recommendations to consider when configuring HDFS Transparent Encryption for Impala.

Recommendations

- If HDFS encryption is enabled, configure Impala to encrypt data spilled to local disk.

- Limit the rename operations for internal tables once encryption zones are set up. Impala cannot do an ALTER TABLE RENAME operation to move an internal table from one database to another, if the root directories for those databases are in different encryption zones. If the encryption zone covers a table directory but not the parent directory associated with the database, Impala cannot do an ALTER TABLE RENAME operation to rename an internal table, even within the same database.
- Avoid structuring partitioned tables where different partitions reside in different encryption zones, or where any partitions reside in an encryption zone that is different from the root directory for the table. Impala cannot do an INSERT operation into any partition that is not in the same encryption zone as the root directory of the overall table.
- If the data files for a table or partition are in a different encryption zone than the HDFS trashcan, use the PURGE keyword at the end of the DROP TABLE or ALTER TABLE DROP PARTITION statement to delete the HDFS data files immediately. Otherwise, the data files are left behind if they cannot be moved to the trashcan because of differing encryption zones. This syntax is available in Impala 2.3 and higher.

Steps

Start every impalad process with the --disk_spill_encryption=true flag set. This encrypts all spilled data using AES-256-CFB. Set this flag by selecting the Disk Spill Encryption checkbox in the Impala configuration (Impala serviceConfigurationCategorySecurity).

Important: Impala does not selectively encrypt data based on whether the source data is already encrypted in HDFS. This results in at most 15 percent performance degradation when data is spilled.

KMS ACL Configuration for Impala

Cloudera recommends making the impala user a member of the hive group, and following the ACL recommendations in KMS ACL Configuration for Hive.

Related Information

KMS ACL Configuration for Hive

Transparent Encryption Recommendations for MapReduce and YARN

MapReduce v1 stores both history and logs on local disks by default. Even if you do configure history to be stored on HDFS, the files are not renamed. Hence, no special configuration is required.

Recommendations for MapReduce v2 (YARN)

Make /user/history a single encryption zone, because history files are moved between the intermediate and done directories, and HDFS encryption does not allow moving encrypted files across encryption zones. When you create the encryption zone, name the key mapred-key to take advantage of auto-generated KMS ACLs.

Steps

On a cluster with MRv2 (YARN) installed, create the /user/history directory and make that an encryption zone.

If /user/history already exists and is not empty:

1. Create an empty /user/history-tmp directory.
2. Make /user/history-tmp an encryption zone.
3. DistCp all data from /user/history into /user/history-tmp.
4. Remove /user/history and rename /user/history-tmp to /user/history.

Transparent Encryption Recommendations for Search

Make /solr an encryption zone. When you create the encryption zone, name the key solr-key to take advantage of auto-generated KMS ACLs.

Steps

On a cluster without Solr currently installed, create the /solr directory and make that an encryption zone.

On a cluster with Solr already installed:

1. Create an empty /solr-tmp directory.
2. Make /solr-tmp an encryption zone.
3. DistCp all data from /solr into /solr-tmp.
4. Remove /solr, and rename /solr-tmp to /solr.

KMS ACL Configuration for Search

In the KMS ACL, grant the solr user and group DECRYPT_EEK permission for the Solr key:

```
<property>
  <name>key.acl.solr-key.DECRYPT_EEK</name>
  <value>solr solr</value>
  </description>
</property>
```

Transparent Encryption Recommendations for Spark

There are various recommendations to consider when configuring HDFS Transparent Encryption for Spark.

Recommendations

- By default, application event logs are stored at /user/spark/applicationHistory, which can be made into an encryption zone.
- Spark also optionally caches its JAR file at /user/spark/share/lib (by default), but encrypting this directory is not required.

KMS ACL Configuration for Spark

In the KMS ACL, grant DECRYPT_EEK permission for the Spark key to the spark user and any groups that can submit Spark jobs:

```
<property>
  <name>key.acl.spark-key.DECRYPT_EEK</name>
  <value>spark spark-users</value>
</property>
```

Transparent Encryption Recommendations for Sqoop

There are various recommendations to consider when configuring HDFS Transparent Encryption for Sqoop.

Recommendations

- For Hive support: Ensure that you are using Sqoop with the --target-dir parameter set to a directory that is inside the Hive encryption zone.
- For append/incremental support: Make sure that the sqoop.test.import.rootDir property points to the same encryption zone as the --target-dir argument.
- For HCatalog support: No special configuration is required.

Integrating Components for Encrypting Data at Rest

How to integrate Cloudera Data Encryption components to provide enterprise data encryption solutions.

Ranger Key Mangement System (KMS)

Consists of Ranger KMS Ranger KMS providing enterprise-grade key management with a backend database that provides key storage.

1. Install Ranger KMS using CM Administration Security HDFS Encryption Wizard .
2. Install a seperate database to store keys.

   For more information, see related links.

Ranger KMS and HSM

Consists of Ranger KMS and database integrated with a backend hardware security module (HSM). In this solution, Ranger KMS provides enterprise-grade key management, HSM provides encryption zone key protection. HSM stores only the encryption master key.

1. Install Ranger KMS using CM Administration Security HDFS Encryption Wizard .
2. Install a seperate database to store keys.
3. Obtain and Integrate one of the following hardware security modules (HSM) supplied by a vendor.

   - Luna 6 or 7
   - CipherTrust
   - GCP Cloud HSM
   - Azure Key Vault

   For more information, see related links.

Ranger KMS and Key Trustee Server (KTS)

Consists of Ranger KMS providing enterprise-grade key management and the Key Trustee Server key store that stores and manages cryptographic keys and other security artifacts.

1. Install Ranger KMS backed by KTS using CM Administration Security HDFS Encryption Wizard .

Ranger KMS, KTS, and Key HSM

Consists of Ranger KMS, KTS and Key HSM which provides seamless integration of all Cloudera encryption components with a HSM added.

1. Install Ranger KMS backed by KTS using CM Administration Security HDFS Encryption Wizard .
2. Obtain and Integrate one of the following hardware security modules (HSM) supplied by a vendor.

   - Luna 6 or 7
   - CipherTrust
   - GCP Cloud HSM
   - Azure Key Vault

For more information, see

Related Information

Configuring a database for Ranger or Ranger KMS

Set up Luna 7 HSM for Ranger KMS w/database

How to integrate Cloudera Ranger Key Management System (KMS) software with the Luna 7 HSM appliance supplied by SafeNet.

About this task

This task describes how to set up the Luna 7 hardware security moudule (HSM) supplied by SafeNet. The process inlcudes setting up Luna 7 HSM on a client (host) and using Cloudera Manager to add configuration properties that enable Ranger KMS and Luna 7 HSM to interact.

Before you begin

You must:

- Acquire the Luna 7 HSM from SafeNet.
- Have both Ranger KMS and a backend database to store keys installed in your environment.

See related topics for more information about installing Ranger KMS and a database to store keys.

Procedure

Set Up the Luna 7 Client

1. Download Luna 7 client on the host where Ranger KMS service resides.

   ```
   610-013144-006_SW_Client_SDK_SafeNet_HSM_7.3.0_Linux_RevA.tar
   ```

2. Untar the Luna 7 client.

   ```
   tar -xf 610-013144-006_SW_Client_SDK_SafeNet_HSM_7.3.0_Linux_RevA.tar
   ```

   the LunaClient_7.3.0-165_Linux/ folder gets created.

3. Navigate to the Luna client folder.

   ```
   cd LunaClient_7.3.0-165_Linux/64/
   ```

4. In the Luna client folder, install Luna products and components.

   ```
   bash install.sh
   ```

   a) At the (y/n) prompt, choose y.

   If you select no or n, this product will not be installed.

   b) At the Products prompt, choose Luna products to be installed:

   - [1]: Luna Network HSM
   - [2]: Luna PCIe HSM
   - [3]: Luna USB HSM
   - [4]: Luna Backup HSM
   - [N|n]: Next
   - [Q|q]: Quit

   Enter selection: 1, then enter selection n.

   c) At the Components prompt, choose Luna Components to be installed

   - [1]: Luna SDK
   - [2]: Luna JSP (Java)
   - [3]: Luna JCProv (Java)
   - [B|b]: Back to Products selection
   - [I|i]: Install
   - [Q|q]: Quit

   Enter selection: i, then enter selection Q.
   Enter selection: 1,2,and 3 then type i.

5. Navigate to the Luna SA command directory.

```
cd /usr/safenet/lunaclient/bin
```

You should see the following:

ls

ckdemo cmu common configurator lunacm multitoken openssl.cnf plink pscp

salogin uninstall.sh vtl

6. Add a user to the hsmusers group.

```
sudo gpasswd --add kms hsmusers
```

7. Copy the Luna appliance server certificate to the client.

```
scp admin@<LunaBoxHostname>:server.pem
```

```
 scp e02paruser115@elab2.safenet-inc.com:server.pem .
(grant permission chmod 777 and chown kms:kms)
The authenticity of host 'elab2.safenet-inc.com (192.43.161.62)' can't be
established.
ECDSA key fingerprint is SHA256:Lz36zjWHh3BMtI9TVHUBGoHffxgA6azFtPSGRBC
kiYU.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'elab2.safenet-inc.com,192.43.161.62' (ECDSA) t
o the list of known hosts.
e02paruser115@elab2.safenet-inc.com's password:  SafeNetPSG95 (given by
 the luna hsm team)
  press enter
server.pem                                     100% 1155      1.1KB/s
           00:00
```

8. Confirm that server.pem is added to the client.

```
ls
```

```
ckdemo cmu common configurator lunacm multitoken openssl.cnf plink pscp
 salogin server.pem uninstall.sh vtl
```

server.pem is added

9. As the KMS user, register the server with the client.

```
su -l kms
./vtl addServer -n <LunaBoxHostname> -c server.pem
```

```
 ./vtl addserver -n elab2.safenet-inc.com -c server.pem
```

vtl (64-bit) v7.3.0-165. Copyright (c) 2018 SafeNet. All rights reserved.

New server elab2.safenet-inc.com successfully added to server list.

10. Generate a client certificate.

```
./vtl  createCert -n  <ClientHostname>
```

```
 ./vtl createcert -n e02paruser115
```

vtl (64-bit) v7.3.0-165. Copyright (c) 2018 SafeNet. All rights reserved.

Private Key created and written to: /usr/safenet/lunaclient/cert/client/e02paruser115Key.pem. Certificate created and written to: /usr/safenet/lunaclient/cert/client/e02paruser115.pem .

(grant permission chmod 777 and chown kms:kms)

11. Copy the client certificate to the server.

```
scp /usr/safenet/lunaclient/cert/client/<ClientHostname>.pem admin@<Luna
BoxHostname>:
```

```
scp /usr/safenet/lunaclient/cert/client/e02paruser115.pem e02paruser115@
elab2.safenet-inc.com:
e02paruser115@elab2.safenet-inc.com's password: SafeNetPSG95
e02paruser115.pem                                             100%
1172   201.7KB/s   00:00
```

12. Login to luna hsm.

```
ssh admin@<lunaboxhostname>
```

```
ssh e02paruser115@elab2.safenet-inc.com
e02paruser115@elab2.safenet-inc.com's password: SafeNetPSG95
Last login: Fri Jul 19 03:59:38 2019 from 114.143.87.94
Luna Network HSM Command Line Shell v7.3.0-165.
Copyright (c) 2018 SafeNet. All rights reserved.
[elab2] lunash:>
```

13. Register the client with the server, then assign the client to a server partition.

```
lunash:> client register -client <ClientHostname> -hostname <ClientHostn
ame>
```

```
client register -client e02paruser115 -hostname e02paruser115
```

14. Check the existing partitions.

```
lunash:> partition list
```

```
lunash:> partition list
                        Storage (bytes)
                        ---------------------------
                        Partition            Name                    Obj
ects   Total    Used    Free
                        ============================================
============================
                        1254277068838        elab2par058
 0  325896       0  325896
```

15. Assign client to the partition.

```
lunash:> client assignPartition -client <ClientHostname> -partition <Gat
ewayPartition>
```

```
lunash:> client assignPartition -client e02paruser115 -partition elab2pa
r058
```

16. client show -client e02paruser115

```
ClientID:     e02paruser115
Hostname:     e02paruser115
Partitions:   "elab2par058"
```

17. Log out from the Luna HSM.

```
lunash:> exit
```

18. Set the read permissions for the certificate files in the following directories.

Note: Make sure to log in as root user.

```
chmod a+r /usr/safenet/lunaclient/cert/server/*.pem
chmod a+r /usr/safenet/lunaclient/cert/client/*.pem
(grant permission chmod 777 and chown kms:kms to above .pem files)
```

19. Verify that the client is connected to its assigned partition.

Note: Make sure to log in as kms user.

```
cd /usr/safenet/lunaclient/bin/
./vtl verify
```

```
[root@os-mv-711-1 bin]# ./vtl verify
vtl (64-bit) v7.3.0-165. Copyright (c) 2018 SafeNet. All rights reserved.

The following Luna SA Slots/Partitions were found:

Slot            Serial #                   Label
==========================================================================
===
0               1254277068842              elab2par115
```

20. ./lunacm

```
./lunacm
```

```
[root@os-mv-711-1 bin]# ./lunacm
lunacm (64-bit) v7.3.0-165. Copyright (c) 2018 SafeNet. All rights rese
rved.

Available HSMs:

Slot ID ->           0
Label ->             elab2par115
Serial # ->          1254277068842
Model ->             LunaSA 7.3.0
```

```
Firmware version ->  7.3.0
Configuration ->     Luna User Partition with SD (PW) Key Export with Cl
eaning Mode
Slot Description ->  Net Token Slot

Current Slot ID:  0
```

21. role login -n co

```
enter password: hanuman123
```

22. par con

If Master Key RangerKMSKey exists, then the following will be visible:

```
lunacm:>par con
    The 'Crypto Officer' is currently logged in.
    Looking for objects accessible to the 'Crypto Officer'.

    Object List:

    Label:             RangerKMSKey
    Handle:            131
    Object Type:       Symmentric Key
    Object UID:        ba8e00002e00000554380800
    Number of Objects: 1

Command Result: No Error
Else
lunacm:>par con

    The 'Crypto Officer' is currently logged in.
    Looking for objects accessible to the 'Crypto Officer'.

    No objects viewable to 'Crypto Officer' are currently stored in the
 partition.
Command Result: No Error
```

23. Navigate to the following directory on the Gateway.

```
# cd /usr/safenet/lunaclient/jsp/lib/
(grant permission chmod 777 and chown kms:kms to all the at this location)
```

24. Copy the Luna .JAR files over to the Gateway.

```
cp libLunaAPI.so Luna*.jar {JAVA_HOME}/jre/lib/ext/
```

```
cp libLunaAPI.so Luna*.jar /usr/java/jdk1.8.0_232-cloudera/jre/lib/ext
```

25. Set the file permissions for the JDK library as follows:

```
chmod a+r {JAVA_HOME}/jre/lib/
```

```
chmod a+r /usr/java/jdk1.8.0_232-cloudera/jre/lib/
```

26. Open the following file in a text editor:

```
vim {JAVA_HOME}/jre/lib/security/java.security
```

```
vim /usr/java/jdk1.8.0_232-cloudera/jre/lib/security/java.security
```

a) Add these two lines:

```
security.provider.6=com.safenetinc.luna.provider.LunaProvider
com.safenetinc.luna.provider.createExtractableKeys=true
```

replacing the line highlighted below:

```
Java SDK/JRE 1.6.x or 1.7.x installation to read as follows:
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=sun.security.ec.SunEC
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=com.safenetinc.luna.provider.LunaProvider
security.provider.7=sun.security.jgss.SunProvider
security.provider.8=com.sun.security.sasl.Provider
security.provider.9=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.10=sun.security.smartcardio.SunPCSC
```

27. Set the file permissions for the Luna client as follows:

```
chmod -R 777 /usr/safenet
chown kms:kms
```

Set KMS Configuration Properties.

28. In Cloudera Manager Ranger KMS Configs edit the following properties:

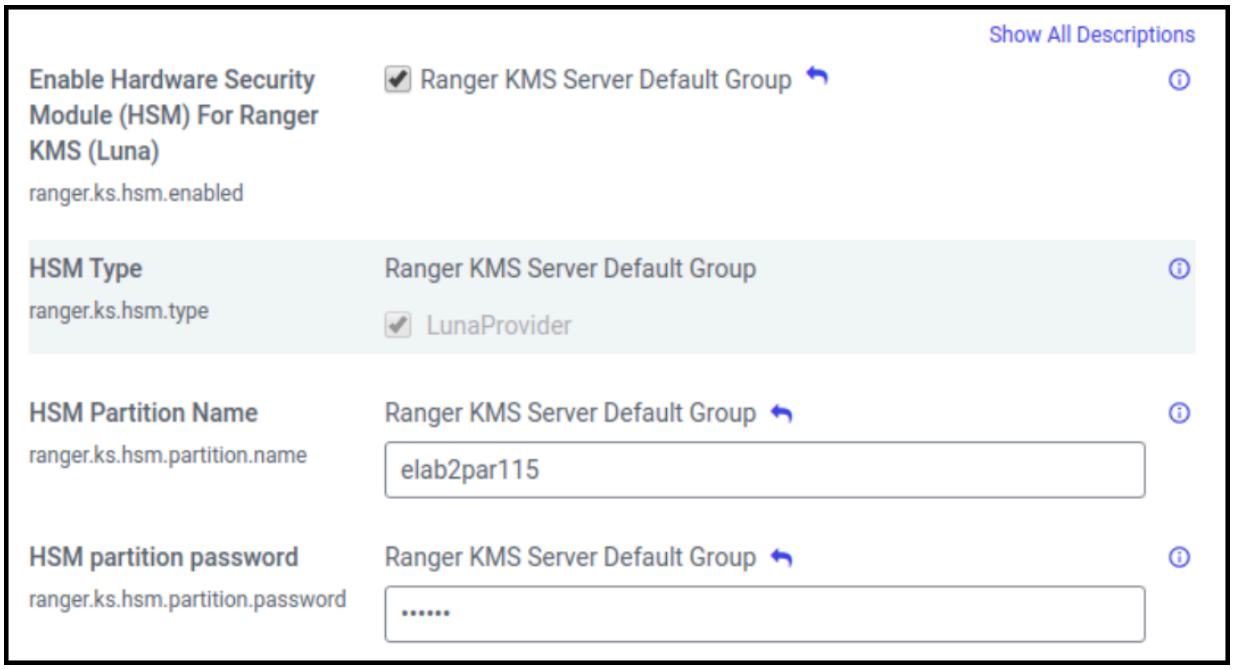
Note: For CM-7.1.1 and CM-7.1.2 you must add properties to the dbks-site.xml, also known as Ranger KMS Server Advanced Configuration Snippet (Safety Valve) for conf/dbks-site.xml .

```
ranger.ks.hsm.type = LunaProvider
ranger.ks.hsm.enabled = true
ranger.ks.hsm.partition.name=elab2par115
ranger.ks.hsm.partition.password=hanuman123
```

```
(CM-7.1.1 & CM-7.1.2 password will be in plain text)
```

Note: For CM-7.1.3 and higher, update the configuration as shown in:

Figure 1: Adding Ranger KMS Configuration for Luna 7 HSM



29. Restart Ranger KMS from Cloudera Manager.
30. Login to Luna client and validate whether the master key is successfully created.

```
cd /usr/safenet/lunaclient/bin/
./lunacm
role login -n co

enter password: hanuman123

par con
```

```
lunacm:>par con
    The 'Crypto Officer' is currently logged in.
    Looking for objects accessible to the 'Crypto Officer'.

    Object List:

    Label:              RangerKMSKey
    Handle:             131
    Object Type:        Symmentric Key
    Object UID:         ba8e00002e00000554380800
    Number of Objects: 1

Command Result: No Error
```

Results

Ranger KMS is successfully started.

What to do next

You can now create Encryption zone keys using hadoop command or from Ranger UI using credentials of keyadmin user.

Set up Luna 6 HSM for Ranger KMS, KTS, and KeyHSM

How to integrate Ranger KMS and KTS with with the Luna 6 HSM appliance supplied by SafeNet.

About this task

This task describes how to set up the Luna 7 hardware security moudule (HSM) supplied by SafeNet. The process inlcudes setting up Luna 6 HSM on a client (host), intalling KeyHSM and using Luna 7 HSM to validate keys.

Before you begin

You must:

- Acquire the Luna 6 HSM from SafeNet.
- Have both Ranger Key Management System and Key Trustee Server installed in your environment.
- Get KeyHSM software.

See related topics for more information about installing Ranger KMS and KTS to store keys.

Procedure

Set Up the Luna 6 Client

Note: Perform the following steps on both active and passive KTS nodes.

1. SSH to (active or passive) KTS node.

```
alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_232-cloudera/
bin/java 1
```

2. Untar the Luna 7 client.

```
tar -xvf safenet-linux-64bit-client-6.2.2.tar
```

   the LunaClient_6.2.2-x_Linux/ folder gets created.

3. Navigate to the Luna client folder.

```
cd LunaClient_6.2.2-x_Linux/64/
```

4. In the Luna clent folder, install Luna products and components.

   Note: Before the install begins, you must select network HSM and JSP as shown in the example.

   ```
   yes | ./linux/64/install.sh -p sa
   ```

   Example:

   a) At the (y/n) prompt, choose y.

      If you select no or n, this product will not be installed.

   b) At the Products prompt, choose Luna products to be installed:

      - [1]: Luna Network HSM
      - [2]: Luna PCIe HSM
      - [3]: Luna USB HSM
      - [4]: Luna Backup HSM
      - [N|n]: Next
      - [Q|q]: Quit

      Enter selection: 1, then enter selection n.

   c) At the Components prompt, choose Luna Components to be installed

      - [1]: Luna SDK
      - [2]: Luna JSP (Java)
      - [3]: Luna JCProv (Java)
      - [B|b]: Back to Products selection
      - [I|i]: Install
      - [Q|q]: Quit

      Enter selection: i, then enter selection Q.
      Enter selection: 1,2,and 3 then type i.

5. Register the HSM on this client.

   a) Retrieve the HSM's public key.

      ```
      $ scp admin@luna-2.atx.cloudera.com:server.pem .
      ```

   b) Register the HSM on the client machine.

      ```
      $ /usr/safenet/lunaclient/bin/vtl addServer -n luna-2.atx.cloudera.com -
      c server.pem
      ```

   c) Confirm the HSM has been added.

      ```
      $ /usr/safenet/lunaclient/bin/vtl list
      ```

   you should see the following:

   ls

   new server <luna.server.name> successfully added to server list

6. Create client certificate.

   ```
   $ /usr/safenet/lunaclient/bin/vtl createCert -n $(hostname -f)
   where  $(hostname -f) is the ip address if running on a virtual machine.
   ```

   Note: The Luna appliance must be able to connect to the hostname across your network.

7. Send the client's public key created in the step above to the HSM.

```
$ scp /usr/safenet/lunaclient/cert/client/$(hostname -f).pem
```

```
$ scp /usr/safenet/lunaclient/cert/client/$(hostname -f).pem admin@luna-
2.atx.cloudera.com.
```

8. Register the client on the HSM.
   a) SSH to the HSM.

```
$ ssh admin@luna-2.atx.cloudera.com
```

   b) Register the client with a friendly name on the HSM.

```
lunaclient> client register -client <friendly.name> -h <hostname.from.st
ep 5.a>
```

```
[luna-2] lunash:> client register -client
 dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.site -h
 dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.site
```

   c) Assign a partition to the client.

```
lunaclient> client assignpartition -client <friendly name> -partition pa
r1
```

Note: This example assumes only acceess to partitions 1 and 2.

```
[luna-2] lunash:>client assignpartition -client dsranktkmslunahsm-4.dsra
nktkmslunahsm.root.hwx.site -partition par1
```

```
[luna-2] lunash:> client register -client dsranktkmslunahsm-4.dsranktkms
lunahsm.root.hwx.site -h dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.
site
'client register successful.
Command result : 0 (Success)
luna-2] lunash:>client assignpartition -client dsranktkmslunahsm-4.dsrank
tkmslunahsm.root.hwx.site -partition par1
'client assignPartition successful.
Command result : 0 (Success)
```

9. Verify registration on the client.

```
$ /usr/safenet/lunaclient/bin/vtl verify
```

```
root@dsranktkmslunahsm-4 /usr/safenet/lunaclient/bin/vtl verify
```

```
The following Luna SA Slots/Partitions were found:
Slot        Serial #         Label
====        =============    ===========
  0             462309014          par1
```

Install and Configure HSM

Note: Perform the following steps on both active and passive KTS nodes.

10. SSH to active/passive KTS node.

11. Obtain Key HSM software.

12. Install Key HSM software.

```
# rpm -ivh keytrustee-keyhsm-*.rpm
```

13. Move the Key Trustee Server and Key HSM installation directory.

```
cd /usr/share/keytrustee-server-keyhsm/
```

14. Configure Key HSM to use SafeNet Luna client.

a) Run # keyhsm setup luna.

```
# keyhsm setup luna
```

b) Use the hostname and any port above 1024.

The recommended port is 9090.

c) Provide data about the HSM slot.

```
# service keyhsm setup luna
-- Configuring keyHsm General Setup --
Please enter keyHsm SSL listener IP address: oks-hsm.vpc.cloudera.com
Please enter keyHsm SSL listener PORT number: 9090
validate Port:                                  :[ Successful ]

-- Configuring SafeNet Luna HSM --
Please enter SafeNetHSM Slot Number: 0
Please enter SafeNet HSM password (input suppressed):
Configuration stored in: 'application.properties'. (Note: You can also
use service keyHsm settings to quickly view your current configuration)
Configuration saved in 'application.properties' file
```

15. Validate the Key HSM service.

```
$ service keyhsm validate
```

```
Check Key HSM is stopped            :[Successful]
Configuration Available             :[Successful]
Port 127.0.0.1:9090 available       :[Successful]
Unlimited-Strength JCE              :[Successful]
Validate cipher list                :[Successful]
HSM availability                    :[Successful]
All services available:             :[Successful]
```

16. Start the Key HSM service.

```
$ service keyhsm start
```

17. Configure Key HSM to trust KTS.

Note: Must be the full path to the file.

```
$ keyhsm trust /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.
pem
```

18. Configure KTS to trust the Key HSM server.

```
$ ktadmin keyhsm --server http://$(hostname -f):<port configured in step
 14.b> --trust
```

```
$ktadmin keyhsm --server http://127.0.0.1:9090 --trust
```

19. Restart Key HSM.

```
$ service keyhsm restart
```

20. Restart the KTS from Cloudera Manager UI.
21. Test the HSM.

```
curl -k https://$(hostname -f):11371/test_hsm
```

22. Login Ranger UI using keyadmin user role for creating an encryption zone key.

Validating Keys in Luna HSM

23. Login to Luna HSM machine .

```
ssh admin@luna-2.atx.cloudera.com
```

24. [luna-2] lunash:>partition showContents -par par1
25. Enter the password for the partition and the Keys will be visible as partition objects.

Results

Ranger KMS is successfully started.

What to do next

We can now create Encryption zone keys using hadoop command or from Ranger UI using credentials of keyadmin user.

Set up Luna 7 HSM for Ranger KMS, KTS, and KeyHSM

How to integrate Ranger KMS, KTS, and KeyHSM with with the Luna 7 HSM appliance supplied by SafeNet.

About this task

The task described in this procdural section guides you through setting up the Luna 7 hardware security moudule (HSM) supplied by SafeNet, for use with Ranger components supplied by Cloudera. The process inlcudes setting up Luna 7 HSM on a client (host), intalling KeyHSM and using Luna 7 HSM to validate keys.

Before you begin

You must:

- Acquire the Luna 7 HSM from SafeNet.
- Have both Ranger Key Management System and Key Trustee Server installed in your environment.
- Get KeyHSM software.

See related topics for more information about installing Ranger KMS and KTS to store keys.

Procedure

Set Up the Luna 7 Client

Note: Perform the following steps on both active and passive KTS nodes.

1. SSH to (active or passive) KTS node.

```
alternatives --install /usr/bin/java java /usr/java/jdk1.8.0_232-cloudera/
bin/java 1
```

2. Untar the Luna 7 client.

```
tar -xvf safenet-linux-64bit-client-7.3.2.tar
```

   the LunaClient_7.3.0-x_Linux/ folder gets created.

3. Navigate to the Luna client folder.

```
cd LunaClient_7.3.0-x_Linux/64/
```

4. In the Luna clent folder, install Luna products and components.

   Note: Before the install begins, you must select network HSM and JSP as shown in the example.

```
<LunaHome>/64/install.sh
```

   Example:

   a) At the (y/n) prompt, choose y.

      If you select no or n, this product will not be installed.

   b) At the Products prompt, choose Luna products to be installed:

      - [1]: Luna Network HSM
      - [2]: Luna PCIe HSM
      - [3]: Luna USB HSM
      - [4]: Luna Backup HSM
      - [N|n]: Next
      - [Q|q]: Quit

      Enter selection: 1, then enter selection n.

   c) At the Components prompt, choose Luna Components to be installed

      - [1]: Luna SDK
      - [2]: Luna JSP (Java)
      - [3]: Luna JCProv (Java)
      - [B|b]: Back to Products selection
      - [I|i]: Install
      - [Q|q]: Quit

      Enter selection: i, then enter selection Q.
      Enter selection: 1,2,and 3 then type i.

5. Register the HSM on this client.
   a) Retrieve the HSM's public key:

   ```
   $ scp admin@luna-2.atx.cloudera.com:server.pem .
   ```

   b) Register the HSM on the client machine.

   ```
   $ /usr/safenet/lunaclient/bin/vtl addServer -n luna-2.atx.cloudera.com -
   c server.pem
   ```

   c) Confirm the HSM has been added.

   ```
   $ /usr/safenet/lunaclient/bin/vtl list
   ```

   you should see the following:

   ls

   new server <luna.server.name> successfully added to server list
6. Create client certificate.

   ```
   $ /usr/safenet/lunaclient/bin/vtl createCert -n $(hostname -f)
   where  $(hostname -f) is the ip address if running on a virtual machine.
   ```

   Note: The Luna appliance must be able to connect to the hostname across your network.
7. Send the client's public key created in the previous step to the HSM.

   ```
   $ scp /usr/safenet/lunaclient/cert/client/$(hostname -f).pem
   ```

   ```
   $ scp /usr/safenet/lunaclient/cert/client/$(hostname -f).pem admin@luna-
   2.atx.cloudera.com.
   ```

8. Register the client on the HSM.
   a) SSH to the HSM.

```
$ ssh admin@luna-2.atx.cloudera.com
```

   b) Register the client with a friendly name on the HSM.

```
lunaclient> client register -client <friendly.name> -h <hostname.from.st
ep 5.a>
```

```
[luna-2] lunash:> client register -client
 dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.site -h
 dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.site
```

   c) Assign a partition to the client.

```
lunaclient> client assignpartition -client <friendly name> -partition pa
r1
```

Note: This example assumes only acceess to partitions 1 and 2.

```
[luna-2] lunash:>client assignpartition -client dsranktkmslunahsm-4.dsra
nktkmslunahsm.root.hwx.site -partition par1
```

```
[luna-2] lunash:> client register -client dsranktkmslunahsm-4.dsranktkms
lunahsm.root.hwx.site -h dsranktkmslunahsm-4.dsranktkmslunahsm.root.hwx.
site
'client register successful.
Command result : 0 (Success)
luna-2] lunash:>client assignpartition -client dsranktkmslunahsm-4.dsrank
tkmslunahsm.root.hwx.site -partition par1
'client assignPartition successful.
Command result : 0 (Success)
```

9. Verify registration on the client.

```
$ /usr/safenet/lunaclient/bin/vtl verify
```

```
root@dsranktkmslunahsm-4 /usr/safenet/lunaclient/bin/vtl verify
```

```
The following Luna SA Slots/Partitions were found:
Slot        Serial #         Label
====        ============     ===========
  0             462309014          par1
```

Install and Configure HSM

Note: Perform the following steps on both active and passive KTS node.

10. SSH to active/passive KTS node.
11. Obtain Key HSM software.
12. Install Key HSM software.

```
# rpm -ivh keytrustee-keyhsm-*.rpm
```

13. Move the Key Trustee Server and Key HSM installation directory.

```
cd /usr/share/keytrustee-server-keyhsm/
```

Note: For Luna 7, you must add the keyhsm user to the hsmgroup group to provide that user access to certificates used by the Luna client software.

```
usermod -G keytrustee,hsmusers keyhsm
```

14. Configure Key HSM to use SafeNet Luna client.

a) Run # keyhsm setup luna

```
# keyhsm setup luna
```

b) Use the hostname and any port above 1024)

The recommended port is 9090.

c) Provide data about the HSM slot.

```
# service keyhsm setup luna
-- Configuring keyHsm General Setup --
Please enter keyHsm SSL listener IP address: oks-hsm.vpc.cloudera.com
Please enter keyHsm SSL listener PORT number: 9090
validate Port:                                  :[ Successful ]

-- Configuring SafeNet Luna HSM --
Please enter SafeNetHSM Slot Number: 0
Please enter SafeNet HSM password (input suppressed):
Configuration stored in: 'application.properties'. (Note: You can also
use service keyHsm settings to quickly view your current configuration)
Configuration saved in 'application.properties' file
```

15. Validate the Key HSM service.

```
$ service keyhsm validate
```

```
Check Key HSM is stopped             :[Successful]
Configuration Available              :[Successful]
Port 127.0.0.1:9090 available        :[Successful]
Unlimited-Strength JCE               :[Successful]
Validate cipher list                 :[Successful]
HSM availability                     :[Successful]
All services available:              :[Successful]
```

16. Start the Key HSM service.

```
$ service keyhsm start
```

17. Configure Key HSM to trust KTS.

Note: Must be the full path to the file.

```
$ keyhsm trust /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.
pem
```

18. Configure KTS to trust the Key HSM server.

```
$ ktadmin keyhsm --server http://$(hostname -f):<port configured in step
 14.b> --trust
```

```
$ktadmin keyhsm --server http://127.0.0.1:9090 --trust
```

19. Restart Key HSM.

```
$ service keyhsm restart
```

20. Restart the KTS from Cloudera Manager UI.
21. Test the HSM.

```
curl -k https://$(hostname -f):11371/test_hsm
```

22. Login Ranger UI using keyadmin user role for creating an encryption zone key.

Validating Keys in Luna HSM

23. Login to Luna HSM machine.

```
ssh admin@luna-2.atx.cloudera.com
```

24. [luna-2] lunash:>partition showContents -par par1
25. Enter the password for the partition and the Keys will be visible as partition objects.

Results

Ranger KMS is successfully started.

What to do next

You can now create Encryption zone keys using hadoop command or from Ranger UI using credentials of keyadmin user. Optionally, you can change the default encryption algoritm for KeyHSM and Luna 7.

Configuring encryption algorithms for Luna 7

How to change the default encryption algorithm for KeyHSM and Luna 7.7.

About this task

KeyHSM supports configuring the specific encryption algorithm used by the Luna 7.7 HSM. This section describes how to configure which specific encryption algorithm Luna 7 uses, by replacing the KeyHSM default algorithm with one of the optional supported algorithms.

Note: Do not change the algorithm value after creating encryption zone keys and encryption keys. Changing the algorithm value will impact the key retrieval and encryption zone operations.

Procedure

1. Stop the KeyHSM service.
2. Navigate to the KeyHSM root directory.
3. In the KeyHSM root dir, open the application.properties file.
4. Find the hsm.luna.encryption.algorithm property, with default value=RSA/ECB/PKCS1Padding.

5. Edit the application properties file to replace the default value with one of the following ones:

   Encryption algorithms supported by KeyHSM/Luna 7.7 HSM:

   - RSA/ECB/PKCS1Padding (default)
   - RSA
   - RSA_ECB_OAEPSHA_224ANDMGF1Padding
   - AES_RSA_NONE_OAEPSHA_512ANDMGF1Padding
   - AES_CBC_PKCS5Padding
   - RSA_NONE_OAEP_WITH_SHA224AndMGF1Padding
   - RSA_NONE_OAEP_WITH_SHA256AndMGF1Padding
   - RSA_NONE_OAEP_WITH_SHA384AndMGF1Padding
   - RSA_NONE_OAEP_WITH_SHA1AndMGF1Padding

   Upgrade Scenario:

   Note: When upgrading KeyHSM from any version < 7.1.7.1 to 7.1.7.1, zone keys and hdfs encryption zone(s) will exist. Do not change the encryption algorithms without first creating new encryption keys, using the following steps:

   a. Unlock the encryption zones with existing keys.
   b. Backup the zone data.
   c. Stop the KeyHSM.
   d. Change the algorithm value as described previously.
   e. Start the KeyHSM.
   f. Create new keys using the new algorithm value.
   g. Lock the encryption zone with new keys.

Set up GCP Cloud HSM for Ranger KMS, KTS, and KeyHSM

How to integrate Ranger KMS and KTS with with the Google Cloud Platform (GCP) HSM service.

About this task

This task describes how to set up the Google Cloud Platform (GCP) hardware security moudule (HSM) service provided by Google. The process inlcudes setting up the GCP HSM service on a client (host), setting up KeyHSM and using the GCP HSM to validate keys.

Before you begin

You must:

- Log in to the Google cloud console using your accout. (Requires Google account access).
- Have Ranger Key Management System, Key Trustee Server and Key HSM installed in your environment.
- Have Java (jdk1.8.0.232) installed.

See related topics for more information about installing Ranger KMS, KTS and KeyHSM.

Procedure

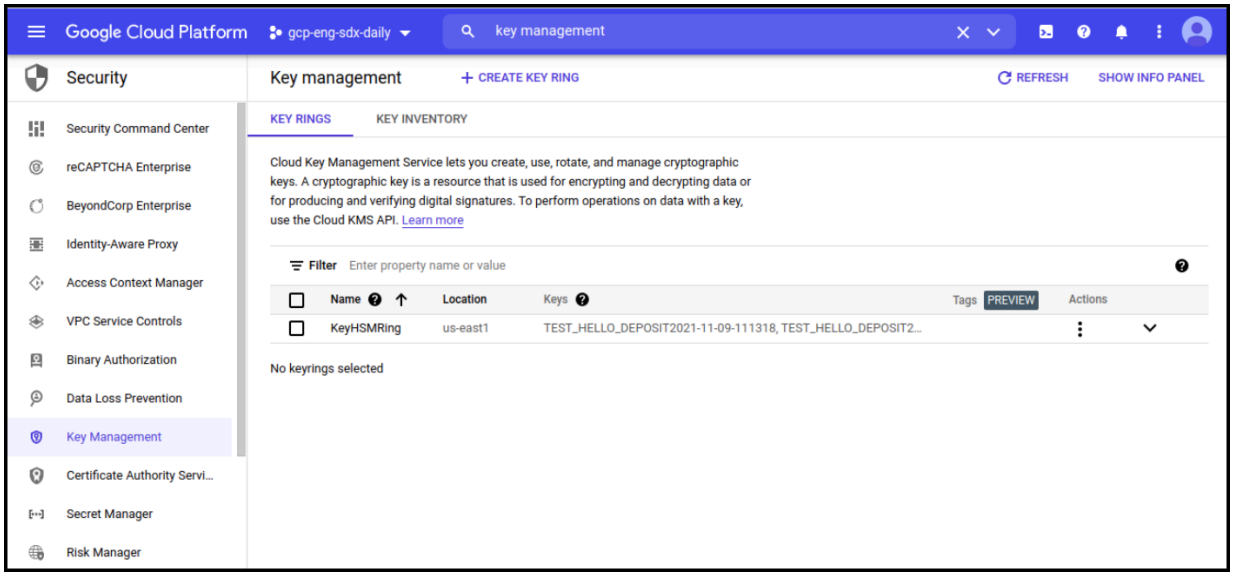
Set Up Google Cloud HSM

1. Login to Google Cloud console using Cloudera account.
2. Create the service account by selecting or creating the Project.
3. Create the key.
4. Download and save the Key in JSON format.

Note: Record the project ID, Location ID and JSON file.

5. In GCP Console Key Management create the key ring.

Figure 2: Creating a key ring in Google Cloud Platform



This example shows a project gcp-eng-sdx-daily, service account keyhsm, and key ring KeyHSMRing.

Integrate GCP with KeyHSM

Note: Perform the following steps on both active and passive KTS nodes.

6. In your Key HSM root directory, copy the autthentication key (json file) you created in the setup process, and provided the appropriate access.

```
# rpm -ivh keytrustee-keyhsm-*.rpm
```

```
cd /usr/share/keytrustee-server-keyhsm/
chown keyhsm:keytrustee <key.filename>.json
```

7. Set up the GCP HSM.

```
keyhsm setup googlecloudhsm
```

```
keyhsm.management.address=127.0.0.1
keyhsm.server.port=9090
google.cloud.hsm.key.ring=KeyHSMRing

# Google App Credential File
google.cloud.hsm.app.cred.file=<authentication file>.json

# Google HSM Project Id
google.cloud.hsm.project.id=<project ID>

# Google HSM Location Id
google.cloud.hsm.location.id=<location ID>
```

Note: Use the key ring, authentication file, project ID, and location ID strings you created during setup.

8. Validate the Key HSM service.

```
$ service keyhsm validate
```

```
Check Key HSM is stopped             :[Successful]
Configuration Available              :[Successful]
Port 127.0.0.1:9090 available        :[Successful]
Unlimited-Strength JCE               :[Successful]
Validate cipher list                 :[Successful]
HSM availability                     :[Successful]
All services available:              :[Successful]
```

9. Start the Key HSM service.

```
$ service keyhsm start
```

10. Configure KTS to trust the Key HSM server.

```
$ ktadmin keyhsm --server http://$(hostname -f):<port configured in setup>
 --trust
```

```
$ktadmin keyhsm --server http://127.0.0.1:9090 --trust
```

11. Restart Key HSM.

```
$ service keyhsm restart
```

12. Restart the KTS from Cloudera Manager UI.
13. Test the HSM.

```
curl -k https://$(hostname -f):11371/test_hsm
```

14. Login to the Ranger UI using keyadmin user role for creating an encryption zone key.

Results

Keys will be created in the Key ring on GCP.

What to do next

Further keys for zone operation can be created using Ranger UI with keyadmin role credentials and also using hadoop commands.

Setting up CipherTrust HSM for Ranger KMS, KTS, and KeyHSM

How to integrate Ranger KMS, KTS, and KeyHSM with the CipherTrust HSM appliance.

About this task

This task describes how to set up the CipherTrust hardware security moudule (HSM) appliance provided by Thales. The process describes configuring the NAE port using CipherTrust Manager, setting up and configuring KeyHSM in your cluster, and validating keys using CipherTrust Manager.

Before you begin

You must have installed the following in your environment:

- Thales CipherTrust Manger .
- Ranger Key Management System, Key Trustee Server and Key HSM
- Java (jdk1.8.0.232)

See related topics for more information about installing Ranger KMS, KTS and KeyHSM.

Procedure

Configuring NAE port in Thales CipherTrust Manager

1. Log in to Thales CipherTrust Manager.
2. In CipherTrust Manager Admin Settings , select Add Interface.
3. In Type, Select NAE (default).
4. In Network Interface, selectAll.
5. In Port, type a value for the port number.
   9000
6. In Mode, select one of the following options to match your environment:
   - No TLS, user must supply password.
   - TLS, Ignore client cert. user must supply password.
7. Click Add.
8. Create a user.
   a) In Access Management Users , click Create New User .
   b) In Create a New User, provide a username, password, and other required information.
   c) Click Create.

Setting up a cluster and configuring KeyHSM

Note: Perform the following steps on both active and passive KTS nodes.

9. In your Key HSM root directory, make sure that appropriate versions of KeyHSM files are available with proper permissions.

```
cd /usr/share/keytrustee-server-keyhsm/
```

Note:

Cipher Trust HSM supports two KeyHSM versions.

   a. If using Ingrian v6.x, then copy all JAR files into this folder and make sure you have provided the required permissions.
   b. If using Ingrian v8.12.x, then copy all the JAR files except gson-2.1.jar into this folder and make sure you have provided the required permissions.

10. If SSL is enabled on CipherTrust Manager run the following command:

```
echo "thales_machine_ip  nae.keysecure.local" >> /etc/hosts
```

11. Setup KeyHSM service.

```
keyhsm setup keysecure
```

```
-- Configuring keyHsm General Setup --
Cloudera Recommends to use 127.0.0.1 as the listener port for Key HSM
Please enter Key HSM SSL listener IP address: [127.0.0.1] Hit Enter
Will attempt to setup listener on 127.0.0.1
Please enter Key HSM SSL listener PORT number: 9090

validate Port:                                    :[ Successful ]

-- Ingrian HSM Credential Configuration --
Please enter HSM login USERNAME: username
```

```
Please enter HSM login PASSWORD: password

Please enter HSM IP Address or Hostname: 18.218.251.172
Please enter HSM Port number: 9000
Valid address:                                    :[ Successful ]

Use SSL? [Y/n] Y (If TSL is enabled on NAE port then press Y else type n
 and hit enter and act accordingly)
org.bouncycastle.cert.X509CertificateHolder@f20f09ff
org.bouncycastle.cert.X509CertificateHolder@ebb30faf
Trust this server? [y/N] y

Trusted server:                                   :[ Successful ]
```

12. Validate the KeyHSM service.

```
$ service keyhsm validate
```

13. Start the KeyHSM service.

```
$ service keyhsm start
```

14. Configure KeyHSM to trust KTS.

Note: Must be the full path to the file.

```
$ keyhsm trust /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.
pem
```

15. Configure KTS to trust the KeyHSM server.

```
$ ktadmin keyhsm --server http://127.0.0.1:9090 --trust
```

16. Restart the KeyHSM service.

```
$ service keyhsm restart
```

17. Restart the KTS from Cloudera Manager UI.

18. Test the HSM.

```
curl -k https://$(hostname -f):11371/test_hsm
```

19. Login to the Ranger UI using keyadmin user role for creating an encryption zone key and do further validation.

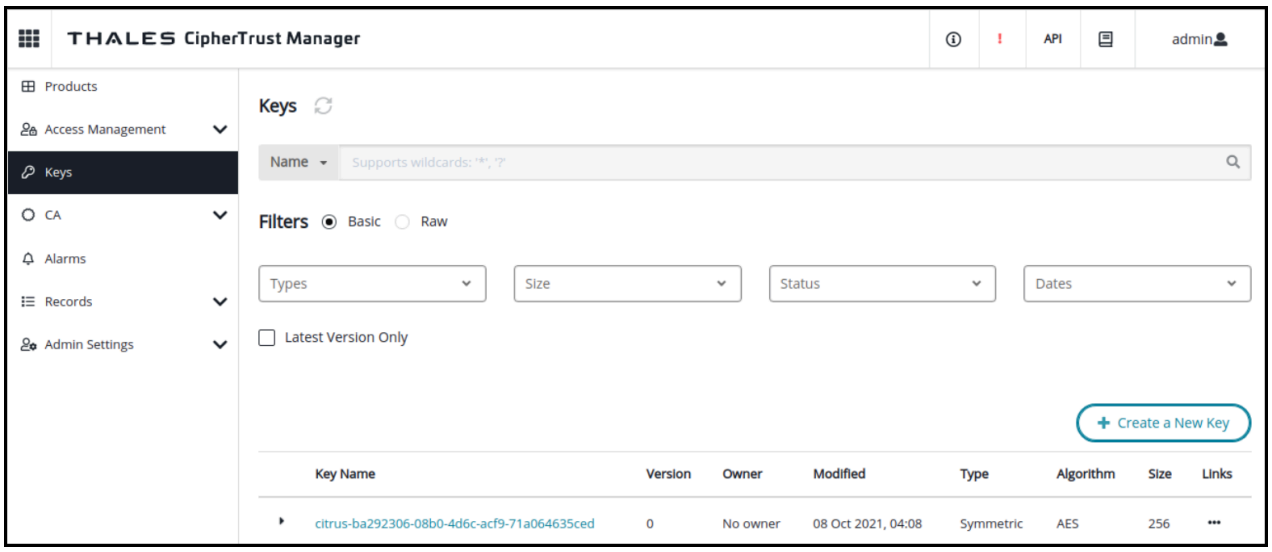
Validating Keys in Cipher Trust HSM

20. In Thales Cipher Trust Manager Left Navigation Panel , click Keys.

Results

Keys created in the second to last step should be present, as shown in:

Figure 3: Validating Keys in CipherTrust Manager



What to do next

Further keys for zone operation can be created using Ranger UI with keyadmin role credentials and also using hadoop commands.

Integrating Ranger KMS DB with Google Cloud HSM

How to integrate Ranger KMS DB with Google Cloud HSM

About this task

This task describes how to integrate Ranger KMS DB with Google Cloud Platform (GCP) Hardware Security Module (HSM). This process includes setting up the GCP HSM service on a client (host), configuring Ranger KMS with GCP, or migrating the Master Key storage from the KMS database to the Google Cloud HSM.

Before you begin

- Ensure you can log in to the Google cloud console using your accout. (Requires Google account access).
- Ensure you have Java (jdk1.8.0.232) installed.

Procedure

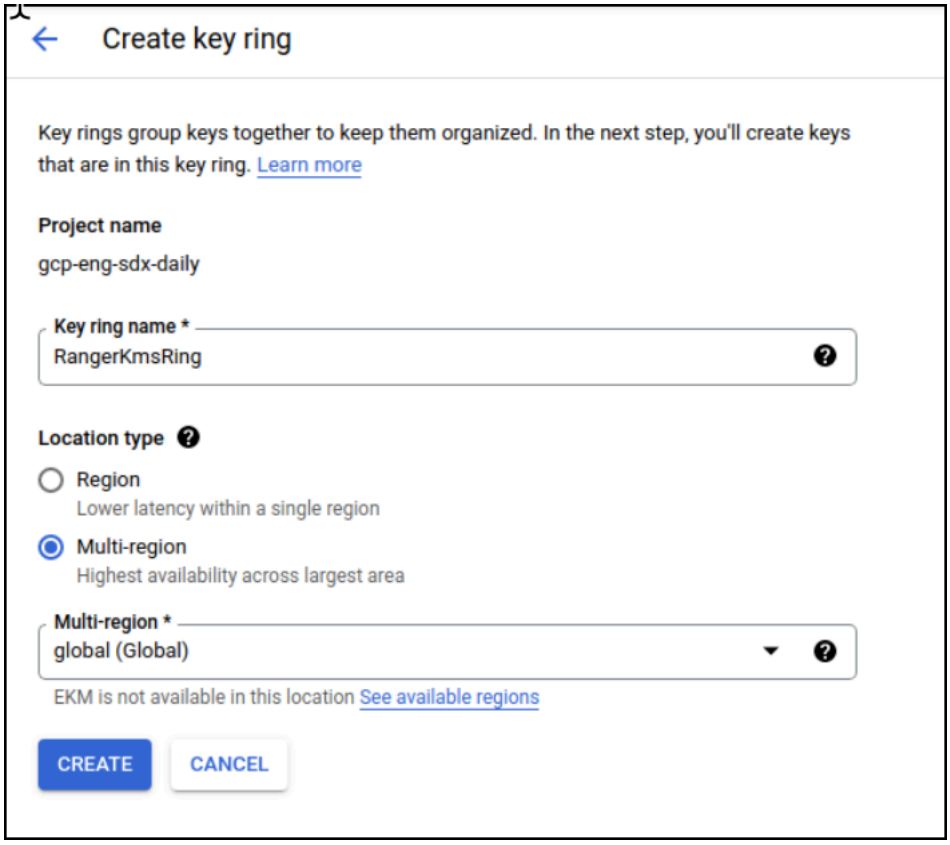
Set Up Google Cloud HSM

1. Log in to Google Cloud console using Cloudera account.
2. Create the service account by selecting or creating the Project.
3. Create the key.
4. Download and save the key in JSON format.

Note: Record the project ID, Location ID and save the JSON file.

5. In GCP Console Key Management create the key ring.

Figure 4: Creating a key ring in Google Cloud Platform

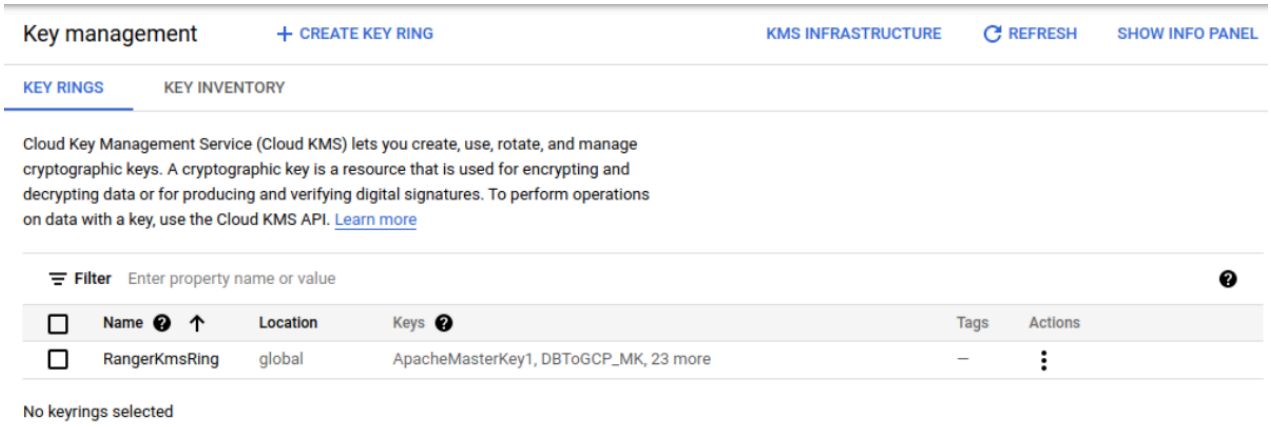


This example shows a project gcp-eng-sdx-daily, region Global, and key ring RangerKMSRing.

Results

The key ring is created.

Figure 5: RangerKMSRing created



Fresh Install - Steps to Configure Ranger KMS with GCP

These are the steps to configure Ranger KMS with the Google Cloud Platform (GCP). These steps need to be performed only when the cluster is ready with all the required services and no encryption zone keys are present.

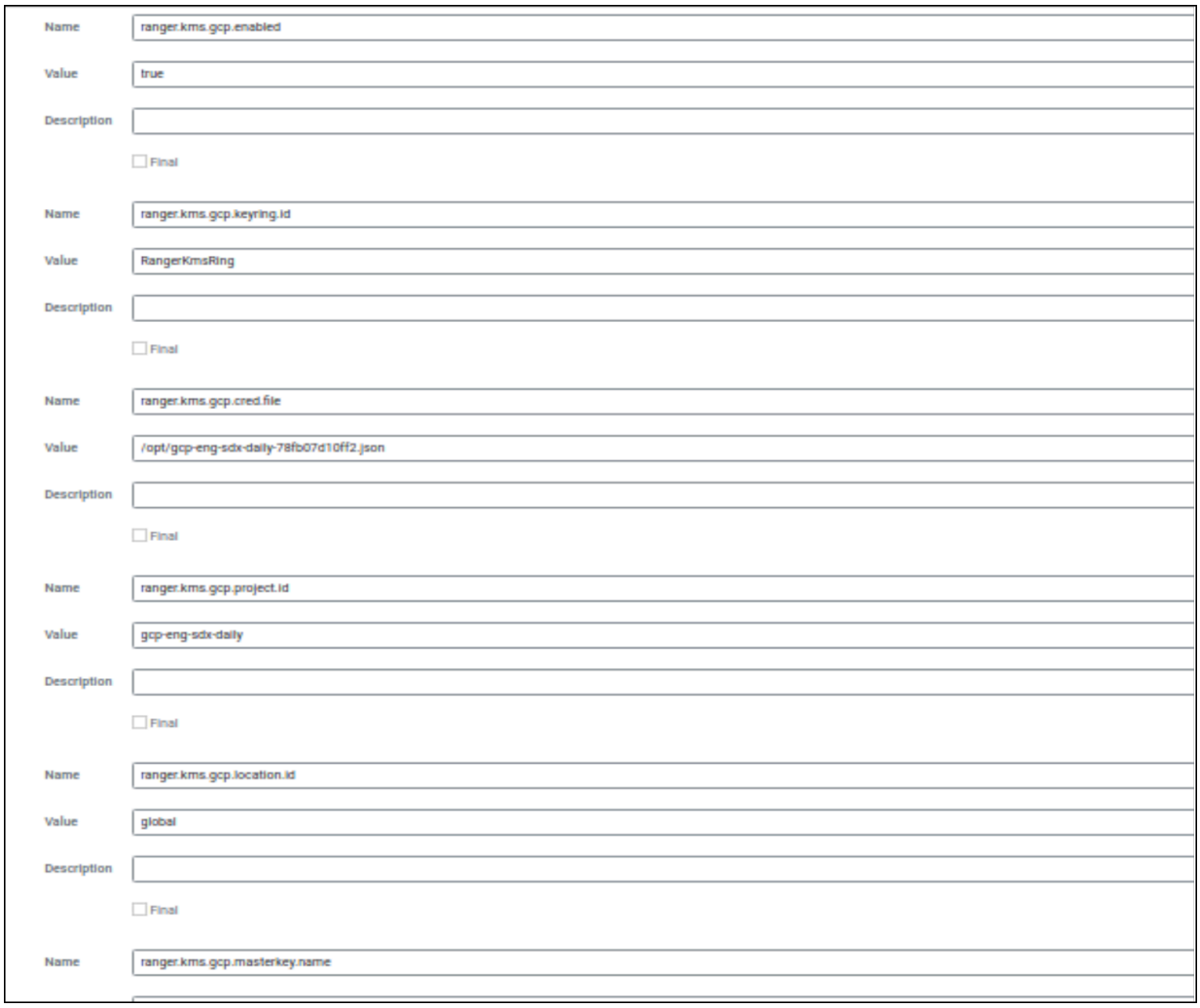
Procedure

1. Stop the KMS service if running.
2. Go to CM Ranger KMS Configuration Search for Ranger KMS Server Advanced Configuration Snippet (Safety Valve) for conf/dbks-site.xml and click on + icon to add the properties.
3. Add the following properties :

Name	Value
ranger.kms.gcp.enabled	true
ranger.kms.gcp.keyring.id	<Key Ring Name>
ranger.kms.gcp.cred.file	/path/to/downloadedCredFile.json
ranger.kms.gcp.project.id	your-project-id
ranger.kms.gcp.location.id	project-location-id

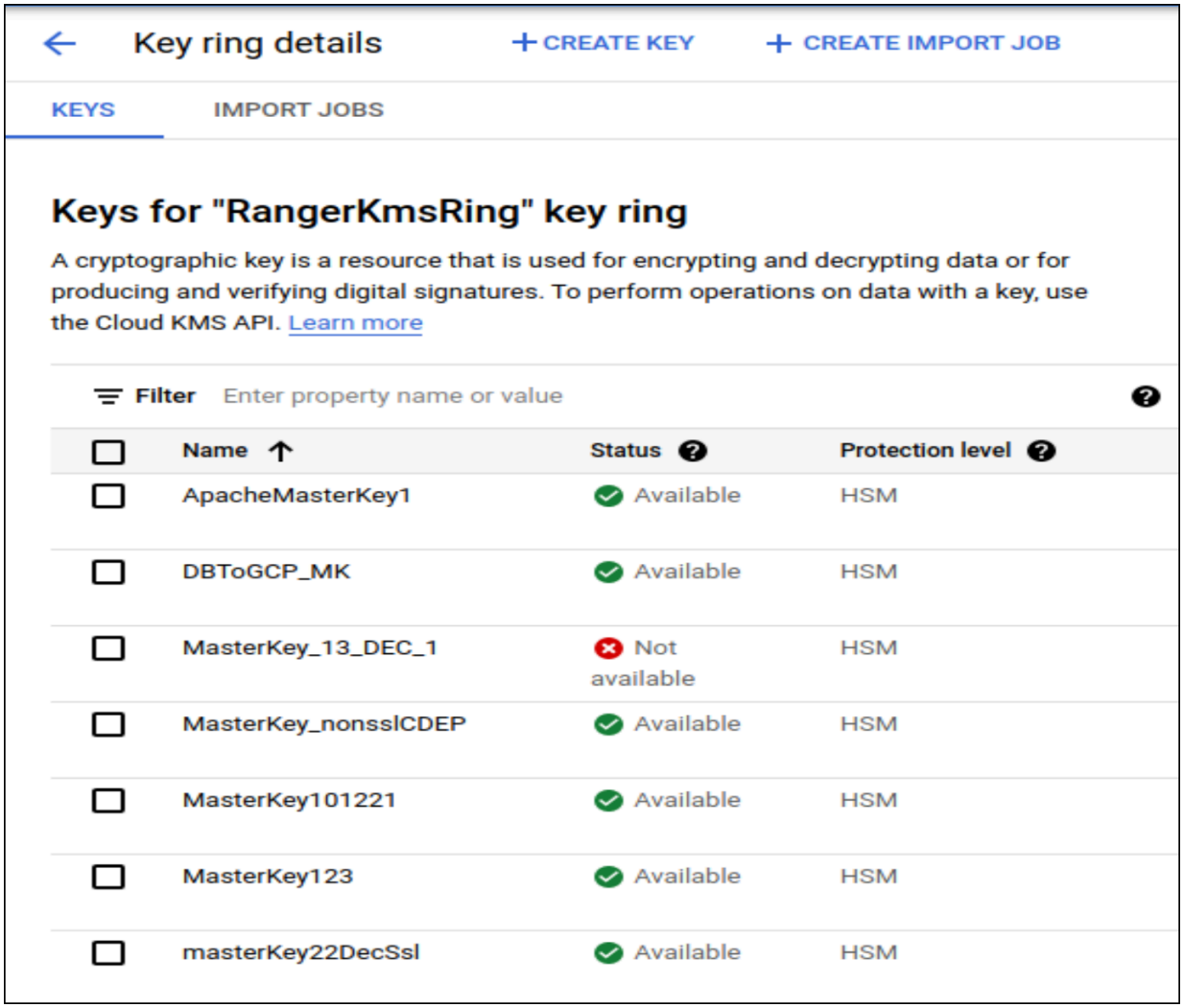
Name	Value
ranger.kms.gcp.masterkey.name	YourMasterKeyName

Figure 6: Adding properties to safety valve



4. Save changes. Restart Ranger KMS

Figure 7: A master key will be created in your keyring on GCP



On restart,a master key will be created in your keyring on GCP.

What to do next

Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Migrating the Master Key From Ranger KMS Database To Google Cloud HSM

These are the steps required to migrate the Master Key Storage from the KMS database to Google Cloud HSM. These steps need to be performed when the cluster is ready with all the required services and encryption zone keys are present in the Ranger KMS DB.

Procedure

Note: These steps are not required if Ranger KMS with GCP is being installed for the first time.

1. Export the following as environment variables JAVA_HOME, RANGER_KMS_HOME, RANGER_KMS_CONF, SQL_CONNECTOR_JAR and HADOOP_CREDSTORE_PASSWORD.You can find the values of the environment variables as follows:
   a) SSH into one of the KMS hosts.
   b) Run the command to find the required environment variable value

   ```
   ps -ef | grep proc_rangerkms
   ```

   c) For HADOOP_CREDSTORE_PASSWORD move to

   ```
   cd /var/run/cloudera-scm-agent/process/xxx-ranger_kms-RANGER_KMS_SERVER
   ```

   d) Open the proc.json file and search for HADOOP_CREDSTORE_PASSWORD.

   ```
   export JAVA_HOME = /usr/java/jdk1.8.0_232-cloudera/
   export RANGER_KMS_HOME = /opt/cloudera/parcels/CDH-7.2.15-1.cdh7.2.15.p
   0.19846155/lib/ranger-kms
   export RANGER_KMS_CONF = /var/run/cloudera-scm-agent/process/51-ranger_km
   s-RANGER_KMS_SERVER/conf/
   export SQL_CONNECTOR_JAR = /usr/share/java/mysql-connector-java.jar
   export HADOOP_CREDSTORE_PASSWORD= hadoop_credstore_pwd
   ```

2. Stop the Ranger KMS service.
3. Once the environment variables have been exported and Ranger KMS service has been stopped, use the MigrateMKeyStorageDbToGCP.sh utility script to migrate the master key.

   ```
   ${RANGER_KMS_HOME}/MigrateMKeyStorageDbToGCP.sh ApacheMasterKey1
   my_gcpProjectName my_gcpKeyRingName  my_gcpKeyRingLocationName my_pathO
   fJsonCredFile.json
   ```

   Note: Migration of Master Key Storage from Google Cloud HSM To Ranger KMS DB is not supported as after creation, a key cannot be moved to another location or exported.

Results

Master Key from Ranger KMS DB has been successfully migrated to GCP.

Integrating Ranger KMS DB with CipherTrust Manager HSM

How to integrate Ranger KMS DB with CipherTrust Manager HSM.

About this task

This task describes how to integrate Ranger KMS DB with CipherTrust Manager Hardware Security Module (HSM). This process includes configuring the NAE port in Thales Cipher Trust Manager, configuring Ranger DB KMS to interact with Thales CipherTrust HSM, or, migrating Ranger KMS DB Master Key To CipherTrust Manager HSM, and migrating the master key from CipherTrust Manager HSM to Ranger KMS DB.

Before you begin

- Ensure you have Thales CipherTrust Manger installed in your enivronment.
- Ensure you have Java (jdk1.8.0.232) installed.

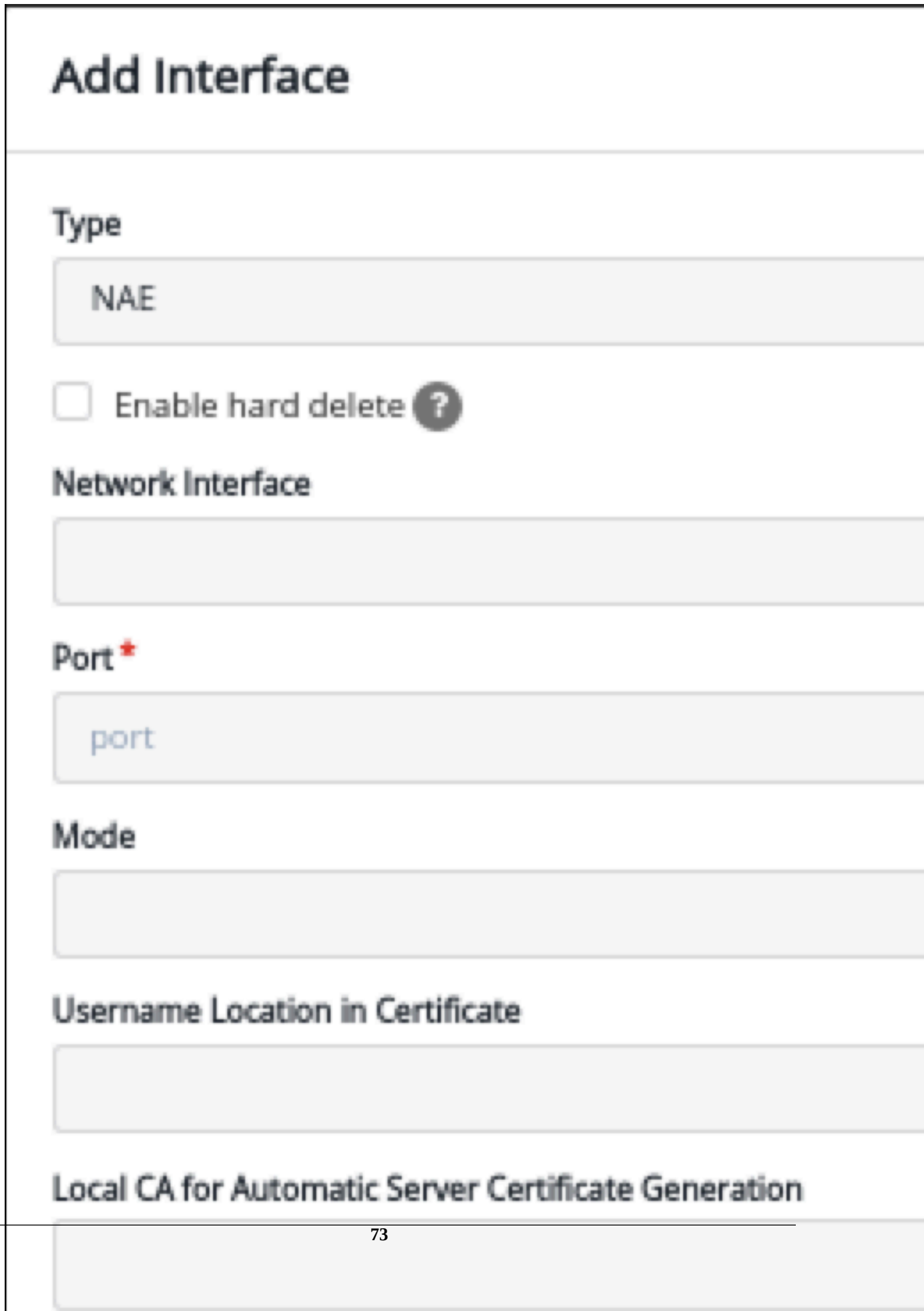
Procedure

Configure NAE port in Thales CipherTrust Manager

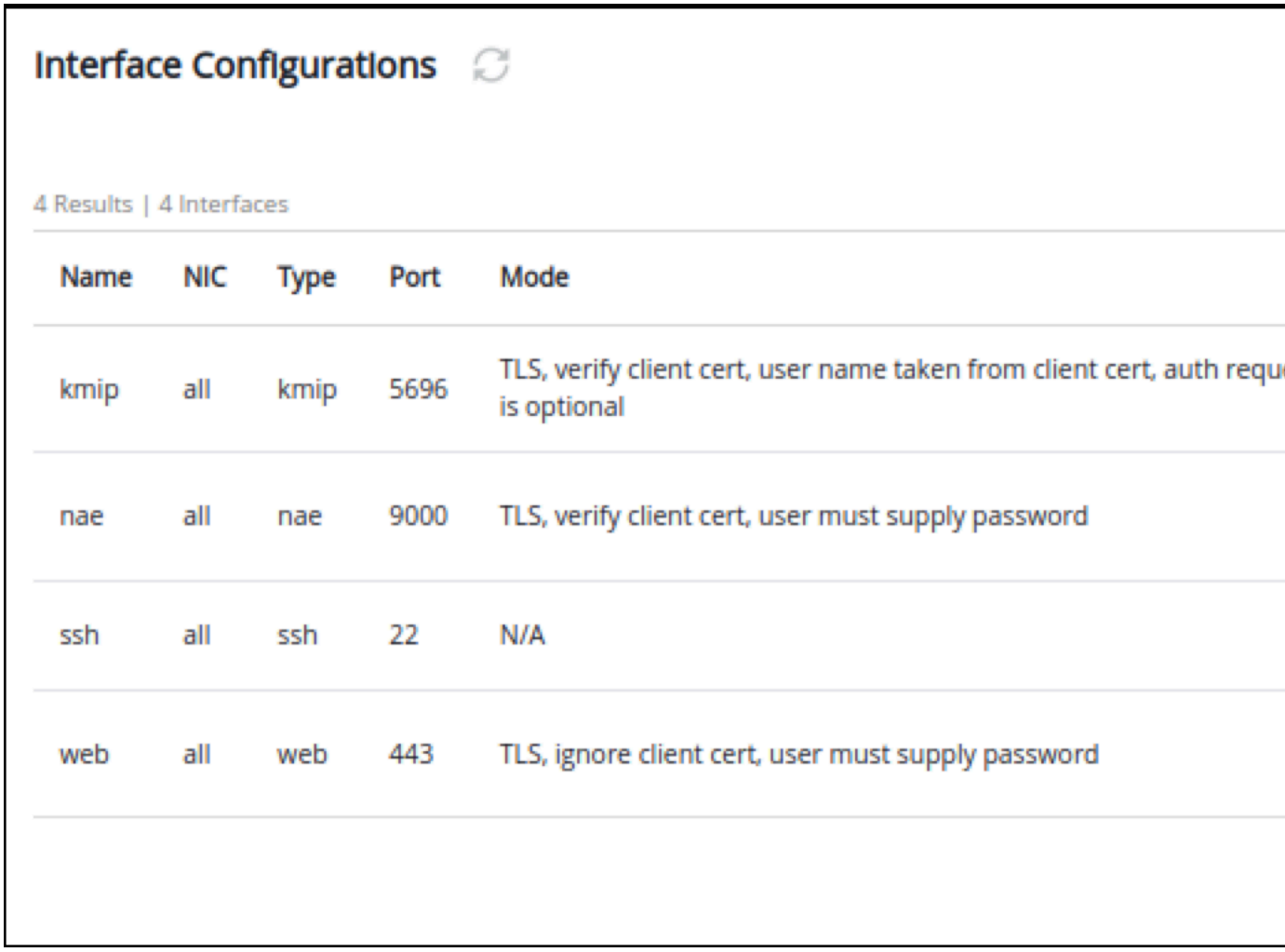
1. Log in to Thales CipherTrust Manager.
2. In CipherTrust Manager Admin Settings , select Add Interface.

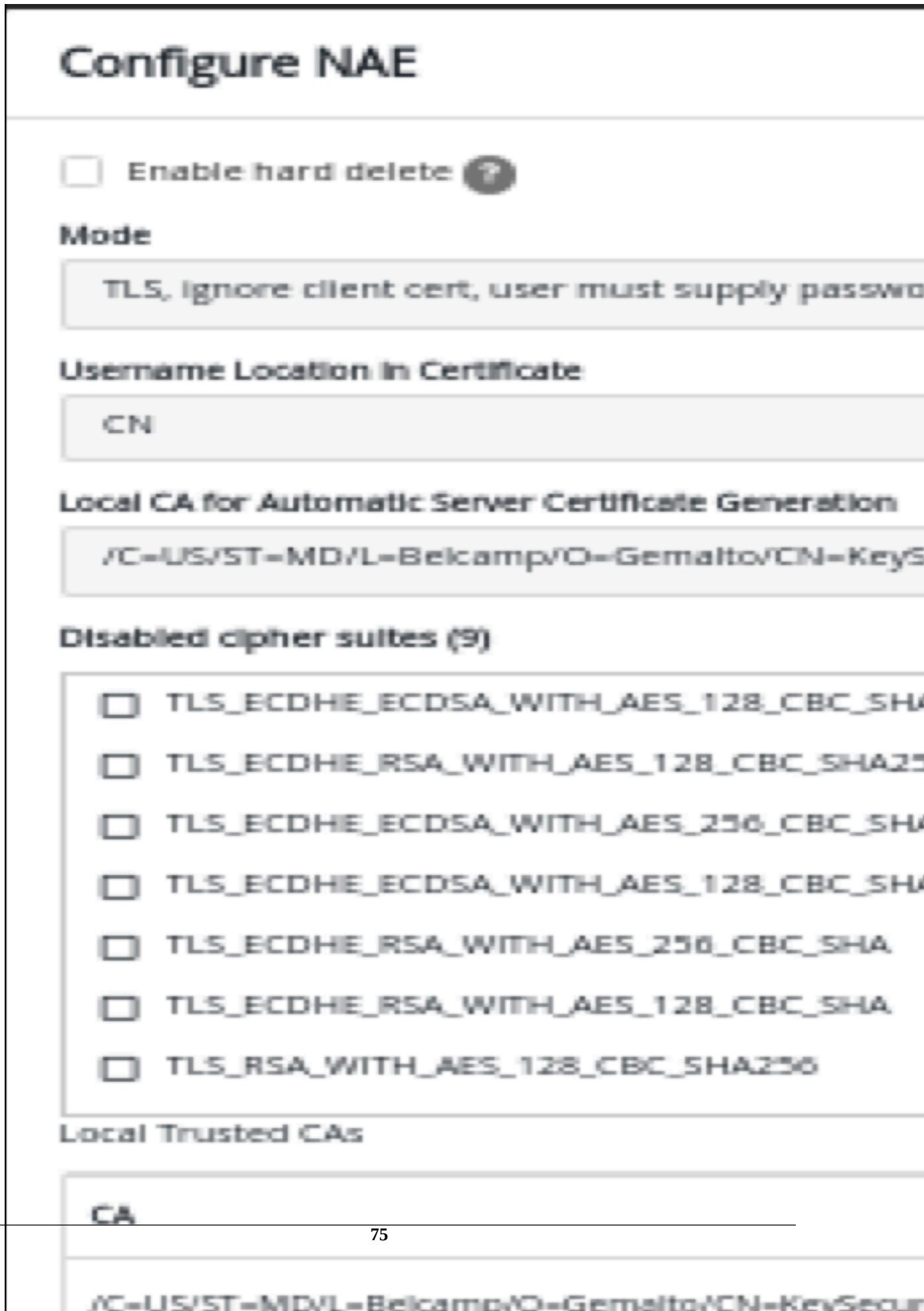
3. In Type, Select NAE (default).
4. In Network Interface, selectAll.
5. In Port, type a value for the port number.
   9000
6. In Mode, select one of the following options to match your environment:
   - No TLS, user must supply password.
   - TLS, Ignore client cert. user must supply password.

7. Click Add.


Add Interface
Type
NAE
Enable hard delete
Network Interface
Port *
port
Mode
Username Location in Certificate
Local CA for Automatic Server Certificate Generation

8. If selected mode is TLS, ignore client cert, user must supply password while adding interface, then click Edit and Download Current Certificate as shown in the images below. Else, skip this step.




Configure NAE
Enable hard delete
Mode
TLS, ignore client cert, user must supply passwo
Username Location in Certificate
CN
Local CA for Automatic Server Certificate Generation
/C=US/ST=MD/L=Belcamp/O=Gemalto/CN=KeyS
Disabled cipher suites (9)
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA25
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
Local Trusted CAs
CA
/C=US/ST=MD/L=Belcamp/O=Gemalto/CN=KeySecur

9. After the certificate is downloaded (e.g -Certificate_nae.txt) copy it to Ranger KMS server
   Create a directory on Ranger KMS serverhost under /etc/security.

   ```
   mkdir etc/security/serverKeys
   ```

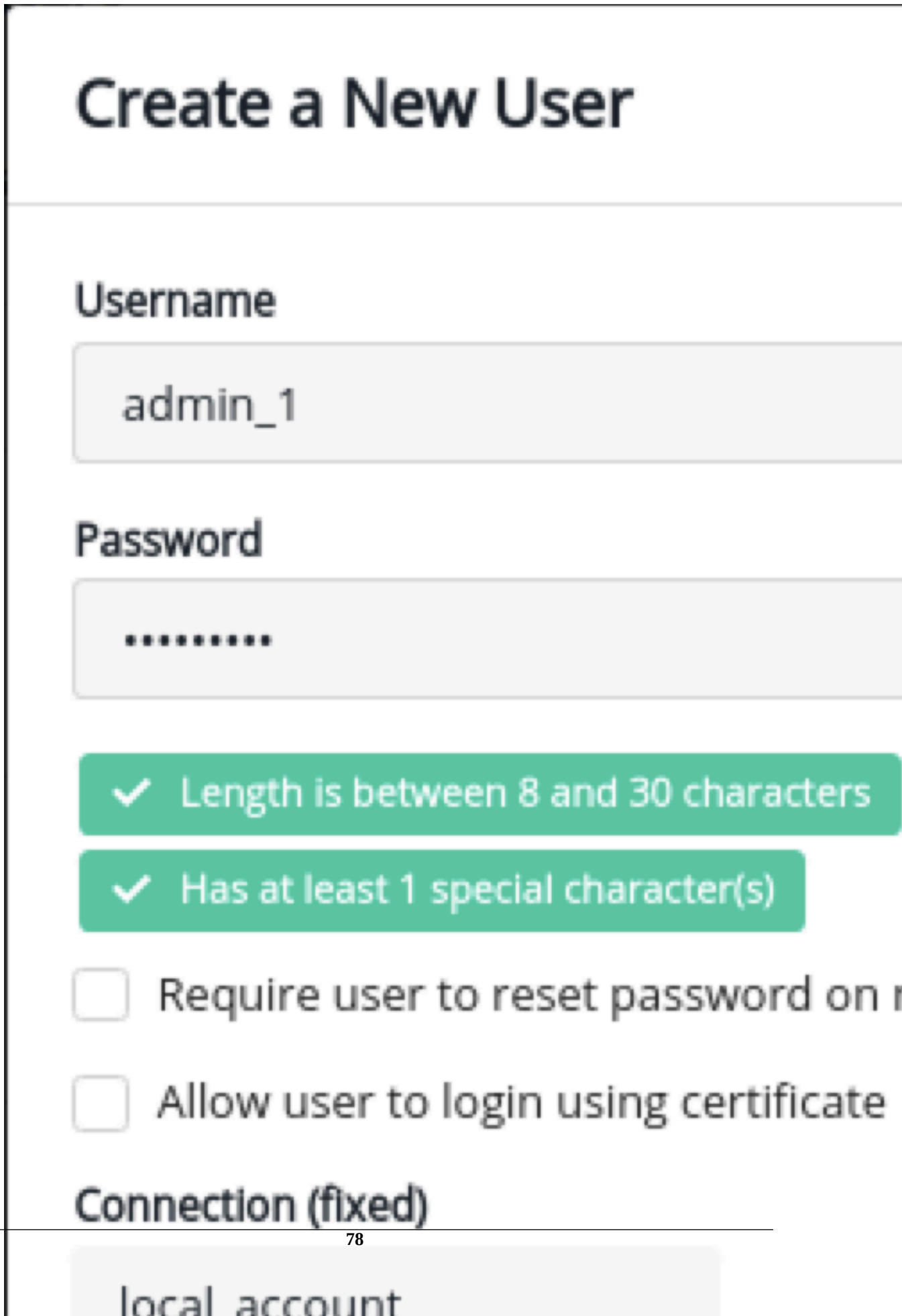
   and scp the downloaded certificate to this directory. Ensure that the user has required access to the file

   ```
   chown kms:kms etc/security/serverKeys/Certificate_nae.txt
   ```

   ```
   chmod 755 etc/security/serverKeysCertificate_nae.txt
   ```

10. Create a user.

a) Go to Access Management Users , click Create New User .
b) In Create a New User, provide a username, password, and any required information.
c) Click Create.


Create a New User
Username
admin_1
Password
Length is between 8 and 30 characters
Has at least 1 special character(s)
Require user to reset password on
Allow user to login using certificate
Connection (fixed)
local_account

Fresh installation - Configuring Ranger KMS DB to interact with Thales CipherTrust HSM.

These are the steps required to configure Ranger KMS DB to interact with Thales CipherTrust HSM. These steps need to be performed only when the cluster is ready with all the required services and no encryption zone keys are present.

Procedure

1. SSH into the Ranger KMS host.
2. Create a directory and copy the files IngrianNAE.properties, libIngPKCS11.so and sunpkcs11.cfg from Thales CipherTrust Manager to the directory.

   ```
   mkdir -p /opt/safenetConf/64/8.3.1
   ```

   and then copy the above mentioned files under this location.

   Note: Make sure you have provided read-write access (chmod 775) to Ranger KMS service users (chown kms:kms) to all the 3 files.
3. Update the IngrianNAE.properties file and initialize the below mentioned properties.

   ```
   NAE_IP = Hostname / CipherTrust Manager IP address
   ```

   ```
   NAE_Port=9000 (should be the same port provided on CipherTrust instance
   under Device tab)
   ```

   ```
   Protocol=tcp (valid values ssl or tcp, should be the same protocol provi
   ded on CipherTrust instance)
   ```
4. Initialize the following property only when the protocol of the CipherTrust Manager instance is SSL. This is the location of the certificate downloaded from CipherTrust Manager and copied on Ranger KMS host.

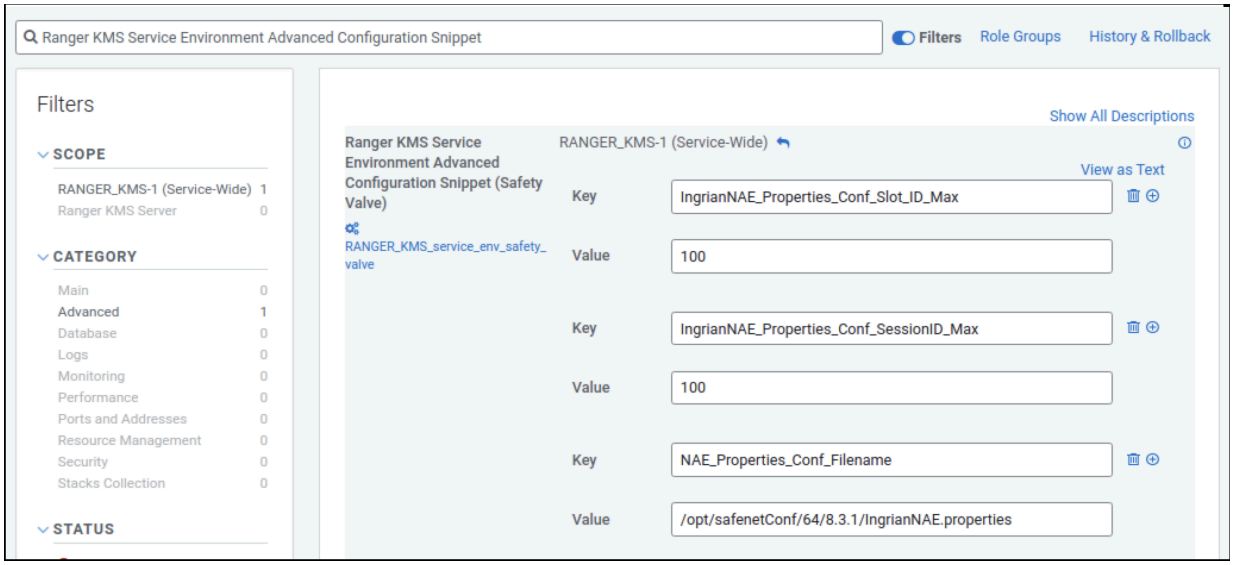
   ```
   CA_File=/etc/security/serverKeys/Certificate_nae.txt
   ```
5. Go to CM Ranger KMS Configuration Search for Ranger KMS Server Advanced Configuration Snippet and click on + icon to add the properties.

6. Add the following properties:

```
IngrianNAE_Properties_Conf_Slot_ID_Max=100
```

```
IngrianNAE_Properties_Conf_SessionID_Max=100
```

```
NAE_Properties_Conf_Filename=/opt/safenetConf/64/8.3.1/IngrianNAE.properties
```



7. Save changes.
8. Search for the following properties and set the values as follows:

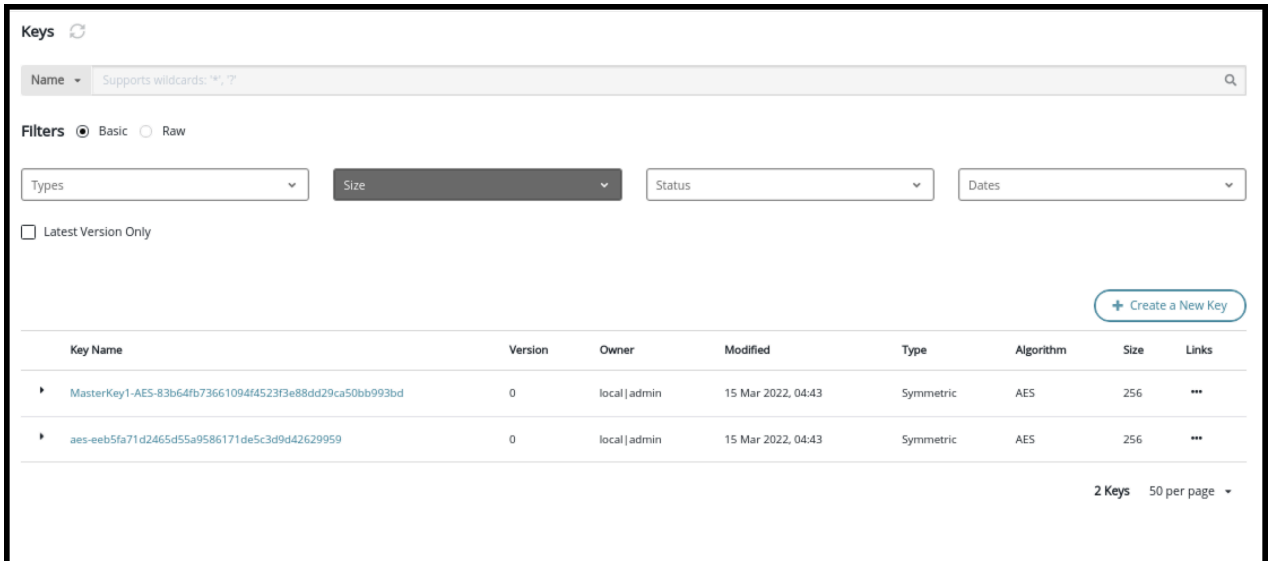
```
ranger.kms.keysecure.enabled = true

ranger.kms.keysecure.UserPassword.Authentication = true

ranger.kms.keysecure.masterkey.name = MasterKey1

ranger.kms.keysecure.login.username = keysecure-username

ranger.kms.keysecure.login.password = keysecure-password

ranger.kms.keysecure.masterkey.size = 256

ranger.kms.keysecure.sunpkcs11.cfg.filepath = /opt/safenetConf/64/8.3.1/sunpkcs11.cfg
```

9. Save changes.
10. Restart Ranger KMS.

Results

The master key with alias MasterKey1 is created in CipherTrust Manager.

Example



What to do next

Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Migrating Ranger KMS DB Master Key To CipherTrust Manager HSM

How to migrate the Ranger KMS DB Master Key to CipherTrust Manager HSM. These steps need to be performed when the cluster is ready with all the required services and encryption zone keys are present in the Ranger KMS DB.

Before you begin

Ensure you have confiured CipherTrust Manager.

Procedure

1. Stop Ranger KMS service.
2. SSH into the Ranger KMS host.
3. Create a directory and copy the files IngrianNAE.properties, libIngPKCS11.so and sunpkcs11.cfg from Thales Cipher Trust Manager to the directory.

   ```
   mkdir -p /opt/safenetConf/64/8.3.1
   ```

   and then copy the above mentioned files under this location.

   Note: Make sure you have provided read-write access (chmod 775) to Ranger KMS service users (chown kms:kms) to all the 3 files.

4. Update the IngrianNAE.properties file and initialize the below mentioned properties.

   ```
   NAE_IP = Hostname / CipherTrust Manager IP address
   ```

   ```
   NAE_Port=9000 (should be the same port provided on CipherTrust instance
   under Device tab)
   ```

   ```
   Protocol=tcp (valid values ssl or tcp, should be the same protocol provi
   ded on CipherTrust instance)
   ```

5. Initialize the following property only when the protocol of the CipherTrust Manager instance is SSL. This is the location of the certificate downloaded from Cipher Trust Manager and copied on Ranger KMS host.

```
CA_File=/etc/security/serverKeys/Certificate_nae.txt
```

6. Go to the Ranger KMS home directory.

```
cd /opt/cloudera/parcels/CDH/lib/ranger-kms
```

7. Export the following variables and ensure the variables are exported successfully using the command env|grep NAE

```
export IngrianNAE_Properties_Conf_Slot_ID_Max=100
export IngrianNAE_Properties_Conf_SessionID_Max=100
export NAE_Properties_Conf_Filename=/opt/safenetConf/64/8.3.1/IngrianNAE.p
roperties
```

8. Go to Ranger KMS home directory and export the environment variables.

```
export JAVA_HOME = /usr/java/jdk1.8.0_232-cloudera
export RANGER_KMS_HOME = /opt/cloudera/parcels/CDH/lib/ranger-kms
export RANGER_KMS_CONF = /var/run/cloudera-scm-agent/process/current_pr
ocsess_dir-RANGER_KMS_SERVER/conf
export SQL_CONNECTOR_JAR = /path/to/SQL-Connector.jar
export HADOOP_CREDSTORE_PASSWORD = hadoop_cred_store_password
```

9. Run the DBMKTOKEYSECURE.sh script.

```
./DBMKTOKEYSECURE.sh masterKey-name keysecure-UserName keysecure-passwor
d /path/to/sunpkcs/config/file/sunpkcs11.cfg
```

Master Key from Ranger KMS DB has been successfully imported into CipherTrust Manager.

10. Go to CM Ranger KMS Configuration. Search for the following properties and set the values.

```
ranger.kms.keysecure.enabled = true
ranger.kms.keysecure.UserPassword.Authentication = true
ranger.kms.keysecure.masterkey.name = masterkeyname (must be same name u
sed above with migration utility ./DBMKTOKEYSECURE.sh )
ranger.kms.keysecure.login.username = keysecure-username
ranger.kms.keysecure.login.password = keysecure-password  (User details
are craeted on KeySecure while adding the user)
ranger.kms.keysecure.masterkey.size = 256
ranger.kms.keysecure.sunpkcs11.cfg.filepath = /opt/safenetConf/64/8.3.1/su
npkcs11.cfg
```

11. Search for Ranger KMS Service Environment Advanced Configuration Snippet. Click on the + icon and add the following environment variables.

```
IngrianNAE_Properties_Conf_Slot_ID_Max=100

IngrianNAE_Properties_Conf_SessionID_Max=100

NAE_Properties_Conf_Filename=/opt/safenetConf/64/8.3.1/IngrianNAE.properti
es
```

12. Save changes.
13. Restart Ranger KMS.

Results
Master Key from Ranger KMS DB has been successfully migrated to CipherTrust Manager.

What to do next
Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Migrating the Master key from CipherTrust Manger HSM to Ranger KMS DB
How to migrate the Master key from CipherTrust HSM to Ranger KMS DB. These steps need to be performed when the cluster is ready with all the required services and encryption zone keys are present in the Ranger KMS DB.

Before you begin
Ensure Ranger KMS service is running.

Procedure
1. Find the KMS current process directory.

```
ps -ef | grep proc_rangerkms
```

2. Stop Ranger KMS service from CM.
3. Go to the Ranger KMS home directory and export the following environment variables.

```
export JAVA_HOME = /usr/java/jdk1.8.0_232-cloudera
export RANGER_KMS_HOME = /opt/cloudera/parcels/CDH/lib/ranger-kms
export RANGER_KMS_CONF = /var/run/cloudera-scm-agent/process/current_pr
ocsess_dir-RANGER_KMS_SERVER/conf
export SQL_CONNECTOR_JAR = /path/to/SQL-Connector.jar
export HADOOP_CREDSTORE_PASSWORD = hadoop_cred_store_password

export IngrianNAE_Properties_Conf_Slot_ID_Max = 100
export IngrianNAE_Properties_Conf_SessionID_Max = 100
export NAE_Properties_Conf_Filename = /path/to/ingrian_props_file/Ingri
anNAE.properties
```

4. If the Ranger KMS DB table contains any old master key, then delete the entry from ranger_masterkey table manually and proceed to the next step.
5. Run the ./KEYSECUREMKTOKMSDB script by passing the master key password.

```
./KEYSECUREMKTOKMSDB masterKeyPassword
```

6. Once the script runs successfully, go to CM Ranger KMS Configuration. Search for dbks-site.xml and update the value of ranger.kms.keysecure.enabled to false.
7. Search for ranger.db.encrypt.key.password and set its value to the master-key password which you used above while invoking the migration script.
8. Restart Ranger KMS.

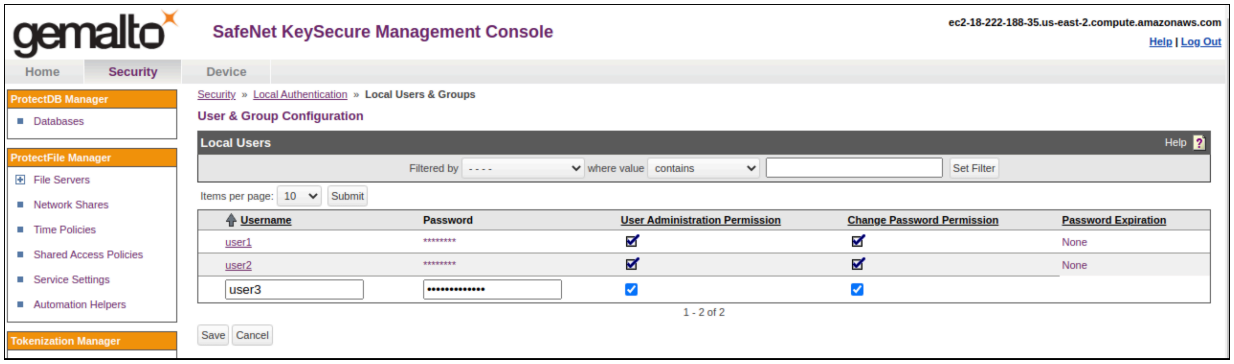
What to do next
Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Integrating Ranger KMS DB with SafeNet Keysecure HSM
How to integrate Ranger KMS DB with SafeNet Keysecure HSM.

About this task
This task describes how to integrate Ranger KMS DB with Safenet Keysecure Hardware Security Module (HSM). This process includes setting up the SafeNet KeySecure Management Console, and configuring Ranger KMS to communicate with the Keysecure instance.

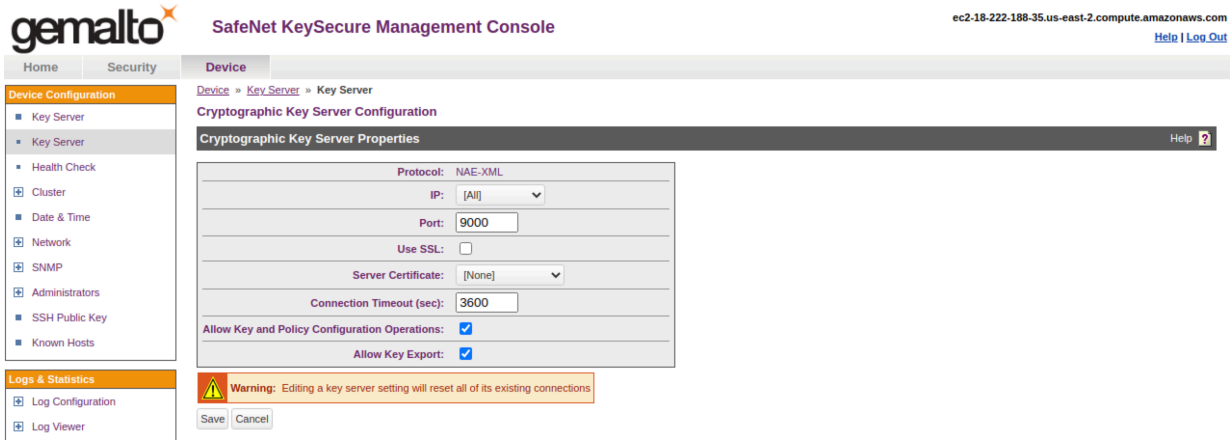
Creating the user on SafeNet keysecure

1. Log in to keysecure as an user with admin privileges.
2. Go to the Security tab.
3. Go to the Users & Groups section.
4. Click Local Authentication, and click Add to add a new user.
5. Check both ‘User Administration Permission’ and ‘Change Password Permission’ when adding the new user.
6. Save changes.



Creating device on SafeNet KeySecure

1. Log in to Keysecure with user having admin privilges.
2. Go to Device NAE-XML protocol.
3. Click Properties Edit.
4. Select Allow Key and Policy Configuration Operations and Allow Key Export .



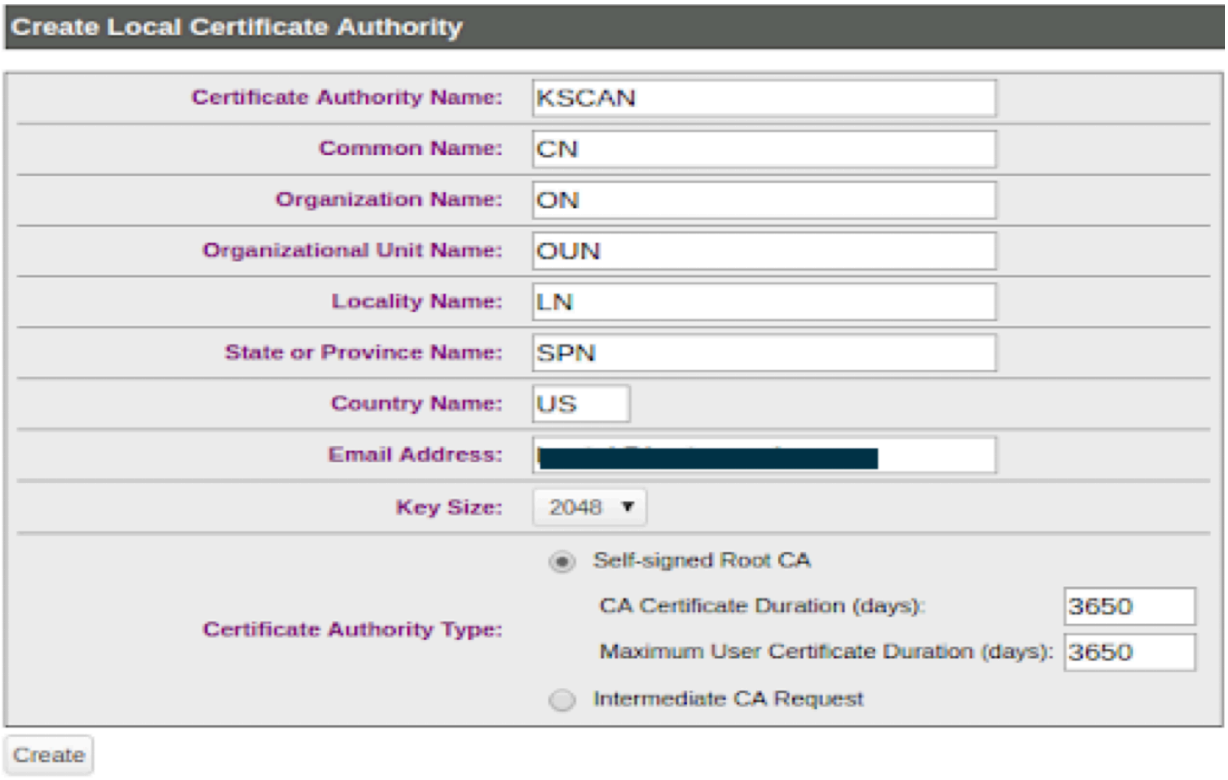
5. Save changes.

Configure SSL on Safenet Keysecure (NAE-XML)

Creating a local CA

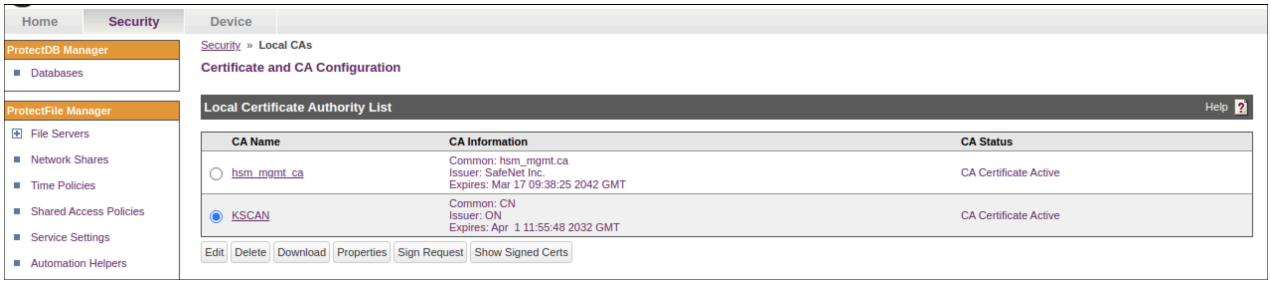
1. Log in to the Management Console as an administrator with Certificate Authority (CA) access control.
2. Navigate to the Security, CAs & SSL Certificates section and click o Local CA's.

3. Enter the required details and select Self-signed Root CA as the Certificate Authority Type.



4. Click Create.

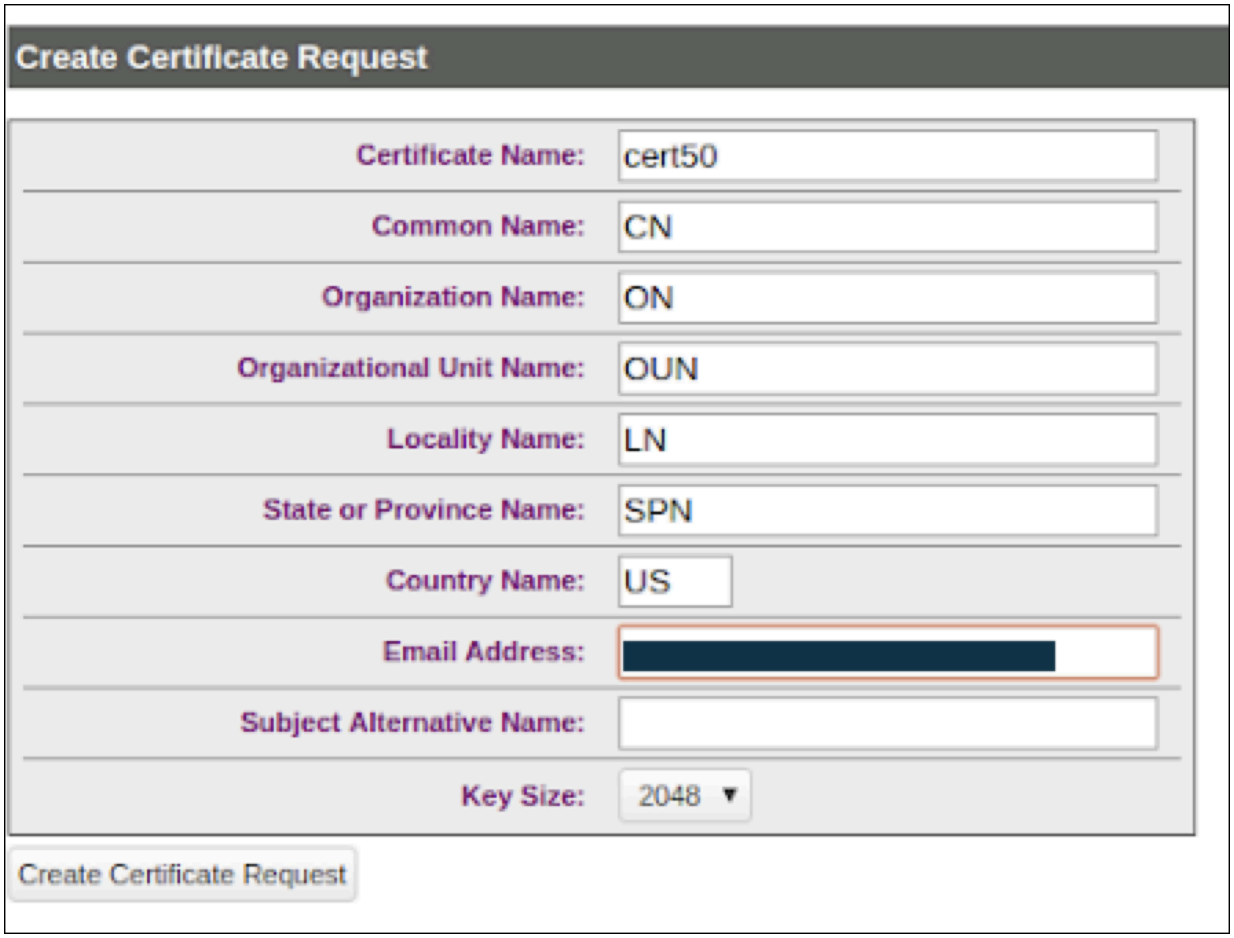
The Local CA is visble.



Creating a Server Certificate Request on the Management Console

1. Log on to the Management Console as an administrator with Certificate Authority (CA) access control.
2. Go to the Security tab and on the left side panel .
3. Navigate to the Device CAs & SSL Certificates section.

4. Click SSL certificates and modify the fields as needed.



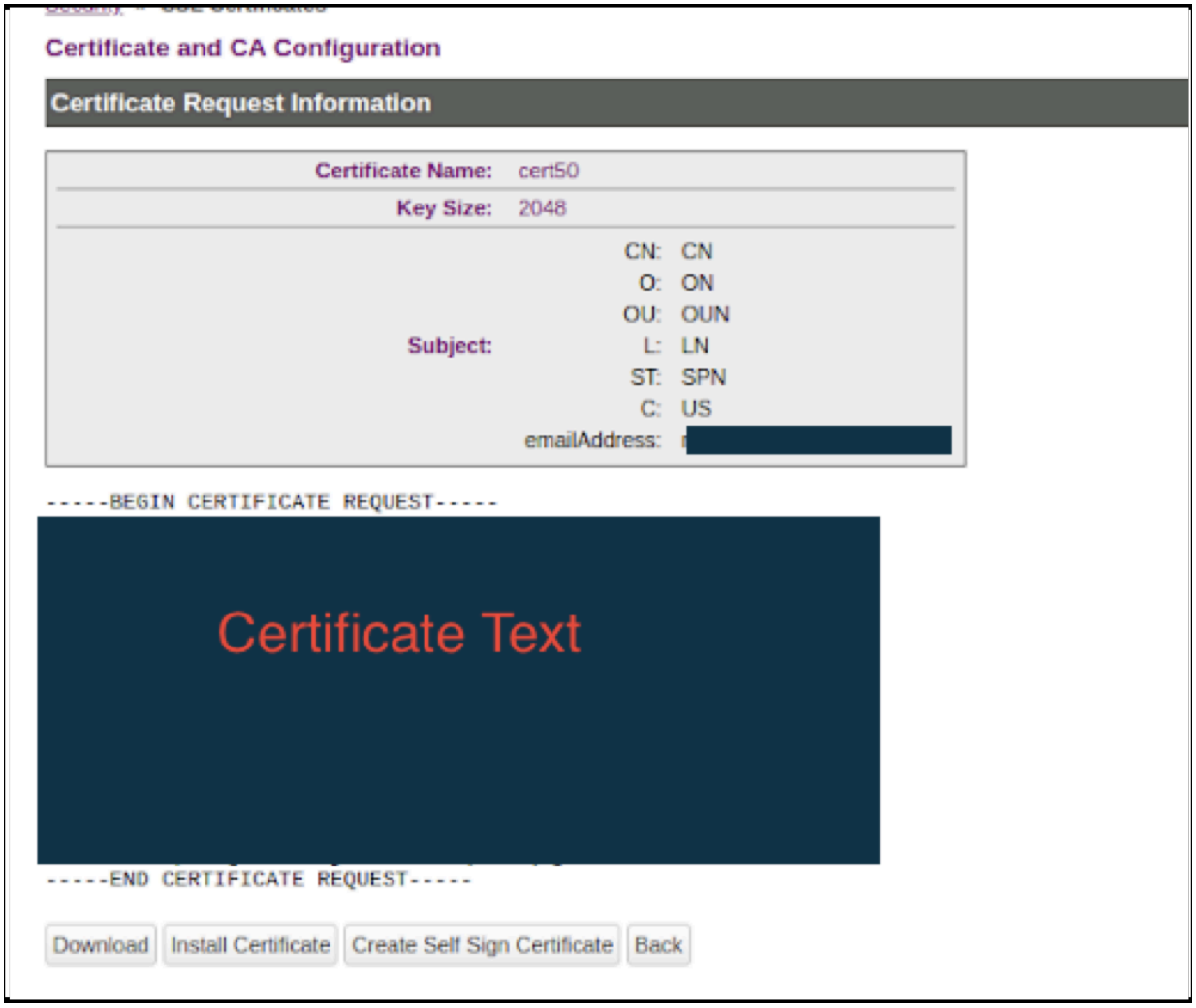
5. Click Create Certificate Request.

   This creates the certificate request and places it in the Certificate List section of the Certificate and CA Configuration page. The new entry shows that the Certificate Purpose is Certificate Request and that the Certificate Status is Request Pending.

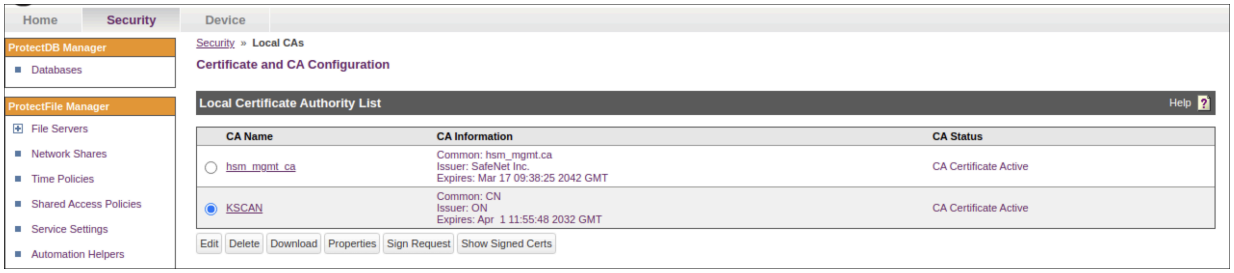
Signing a Server Certificate Request with a Local CA

1. Log on to the Management Console as an Administrator with Certificates and Certificate Authorities (CA) access controls.
2. Navigate to the Security Tab -> Device, CAs and SSL Certificates section.
3. Click SSL Certificates .

4. Select the certificate request (cert50) and click Properties.

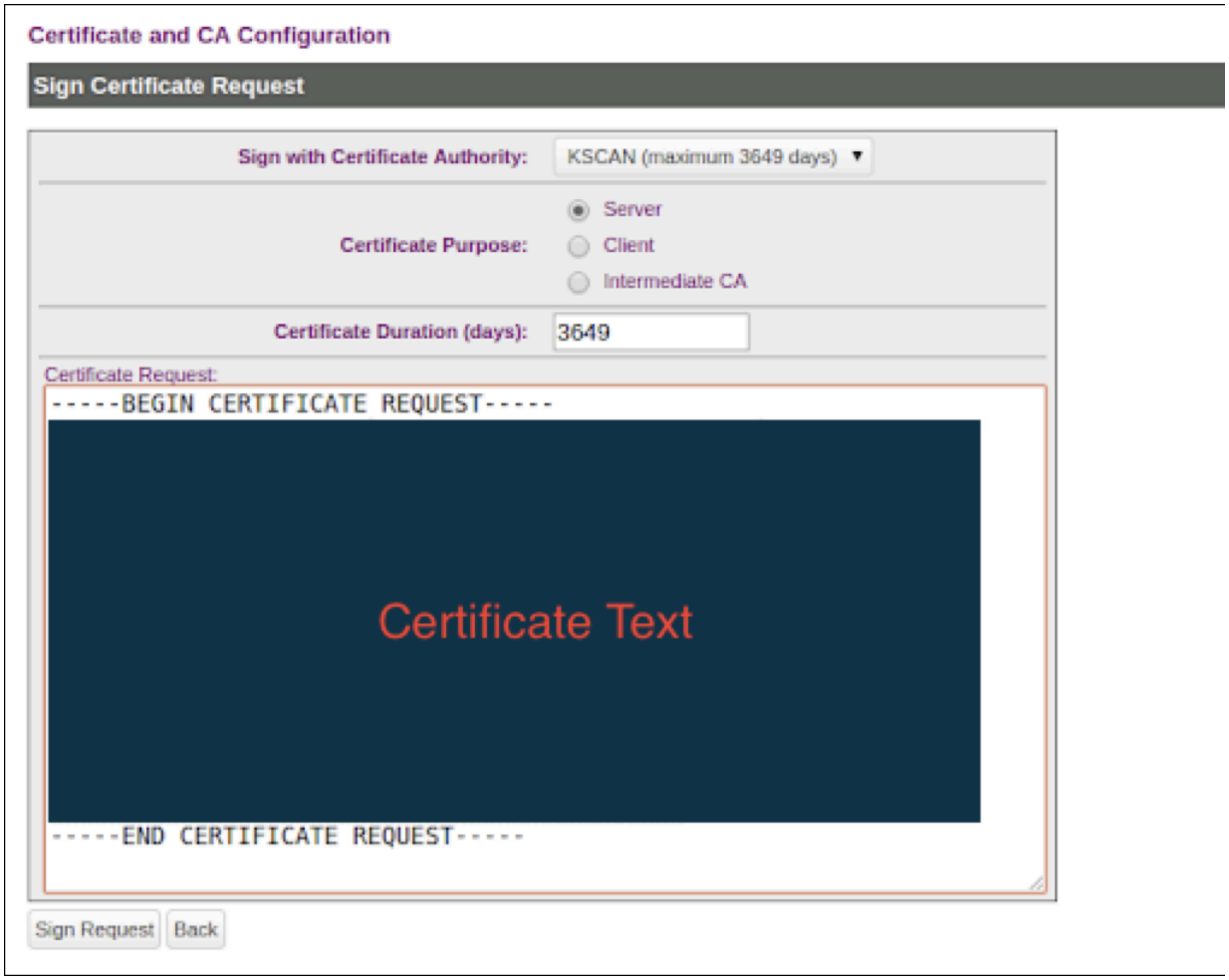


5. Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and footer (-----END CERTIFICATE REQUEST-----).
6. Navigate to the Security Tab -> Device, CAs & SSL Certificates section.
7. Click Local CAs and select the CA name from the list.
8. Click Sign Request to access the Sign Certificate Request section.



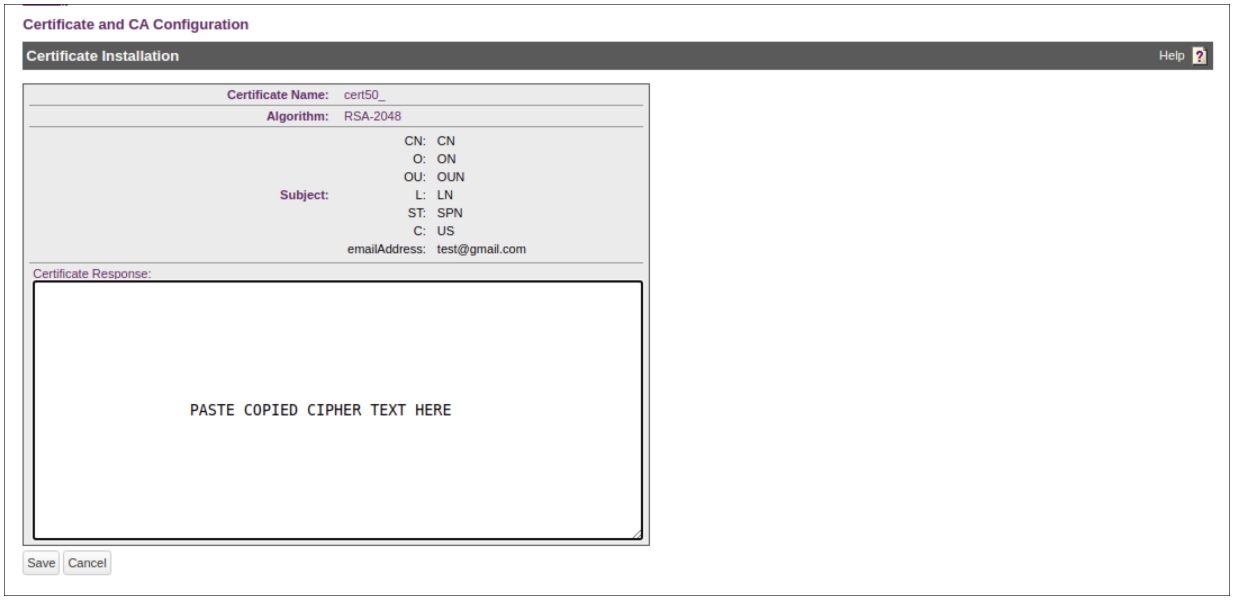
9. On the Sign Certificate Request screen, select Server as certificate Purpose.
10. Enter the validity of the certificate for Certificate Duration (days).

11. Paste the copied text from the server certificate request, including the header and footer in Certificate Request.



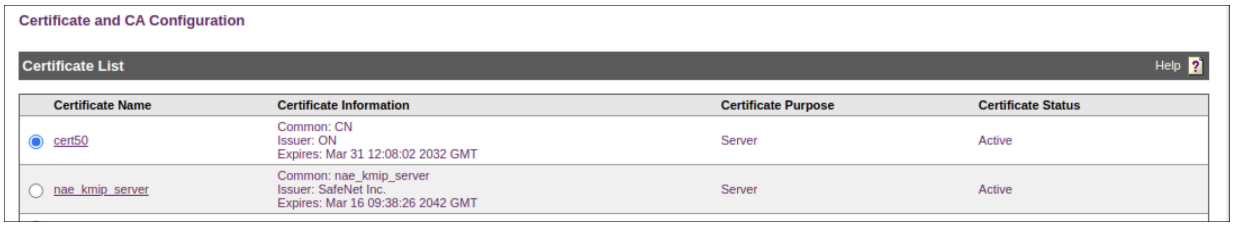
12. Click Sign Request. This takes you to the CA Certificate Information section.
13. Copy the actual (for eaxample,. KSCAN) certificate text. The copied text must include the header (-----BEGIN CERTIFICATE-----) and footer (-----END CERTIFICATE-----).
14. Navigate back to the Certificate List section (Device, CAs & SSL Certificates) and click SSL Certificates.
15. Select your certificate request and click properties.
16. Click Install Certificate.

17. Paste the certificate as the Certificate Response.



18. Click Save.

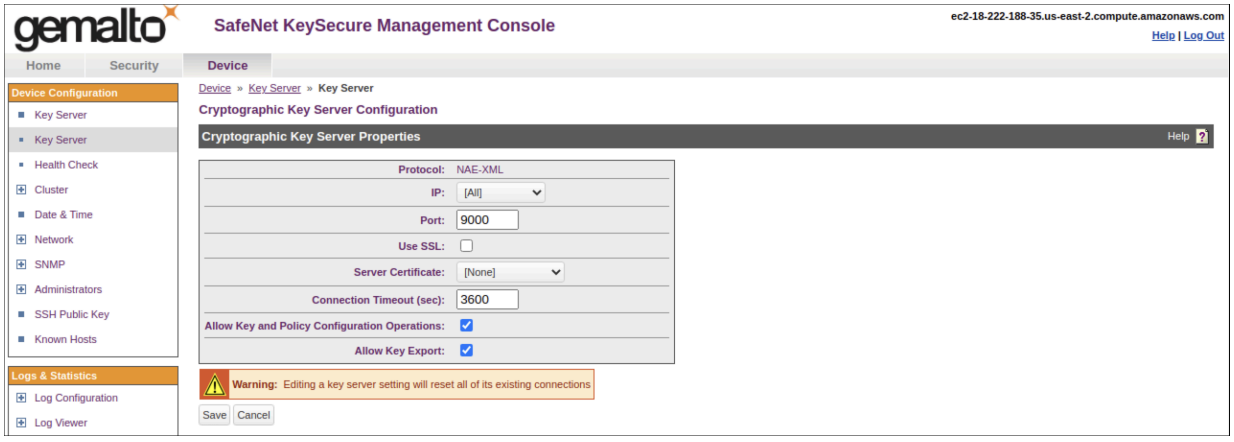
    The Management Console takes you to the Certificate List section. The section shows that the Certificate Purpose is Server and that the Certificate Status is Active.



Enable SSL on Keysecure (NAE-XML)

After SSL has been configured in Safenet KeySecure, perform the following steps.

1. Log in to keysecure with admin privileges.
2. Go to the Device tab and click NAE-XML -> properties -> edit.



3. Select Use SSL.
4. Select the Server Certificate from the given drop-down list (for example, cert50).
5. Save changes.

Fresh Installation Of Ranger KMS with SafeNet KeySecure (NAE-XML)

These are the steps required to configure Ranger KMS DB to interact with Safenet KeySecure HSM. These steps need to be performed only when the cluster is ready with all the required services and no encryption zone keys are created.

Procedure

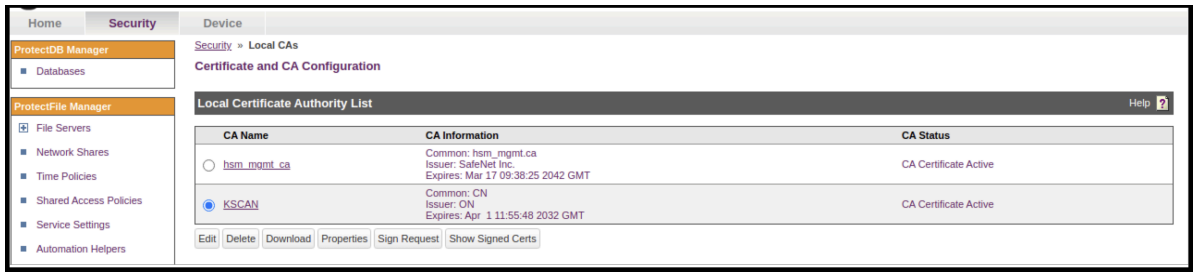
1. SSH into the Ranger KMS host.
2. Create a directory and copy the files IngrianNAE.properties, libIngPKCS11.so and sunpkcs11.cfg from Gemalto SafeNet KeySecure to the directory.

```
mkdir -p /opt/safenetConf/64/8.3.1
```

and then copy the above mentioned files under this location.

Note: Make sure you have provided read-write access (chmod 775) to Ranger KMS service users (chown kms:kms) to all the 3 files.

3. If SSL is enabled on KeySecure ,then download the CA certificate file (e.g. KSCAN.crt) created on key secure cluster while configuring SSL on keysecure instance.
   a) Log in to KeySecure.
   b) Click on Security tab and then go to Device CAs and SSL Certificates section
   c) Click on Local CA's link
   d) Select the appropriate certificate and click on download as shown in the below image.



   e) Once the certificate is downloaded, copy/scp it to the Ranger KMS host at location "/etc/security/serverKeys/". Make sure the Ranger KMS user has read-write access to the file.

```
chown  kms:kms /etc/security/serverKeys/KSCAN.crt
```

```
chmod  755 /etc/security/serverKeys/KSCAN.crt
```

4. Update the IngrianNAE.properties file and initialize the below mentioned properties.

```
NAE_IP = Your SafeNet KeySecure IP address (Provide public IP if its on a
 different network.)
```

```
NAE_Port=9000 (should be the same port provided on KeySecure instance un
der Device tab)
```

```
Protocol=tcp (valid values ssl or tcp, should be the same protocol provi
ded on KeySecure instance)
```

5. Initialize the following property only when the protocol of the KeySecure instance is SSL. This is the location of the certificate downloaded from KeySecure and copied on Ranger KMS host.

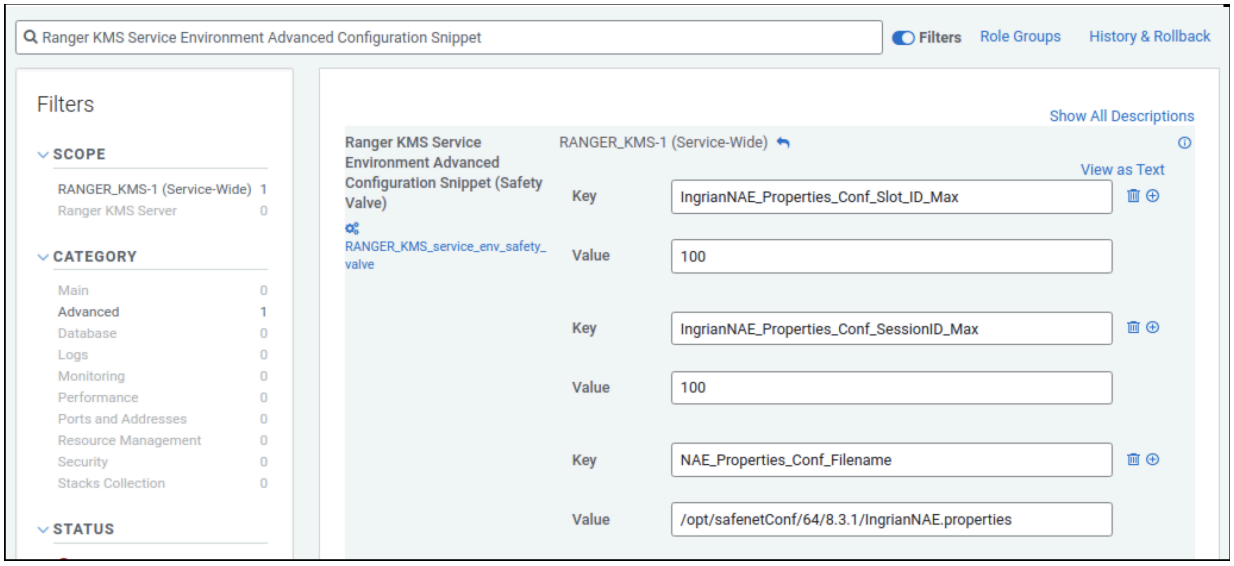
```
CA_File=/etc/security/serverKeys/KSCAN.crt
```

6. Go to CM Ranger KMS Configuration Search for Ranger KMS Server Advanced Configuration Snippet and click on + icon to add the properties.
7. Add the following properties:

```
IngrianNAE_Properties_Conf_Slot_ID_Max=100
```

```
IngrianNAE_Properties_Conf_SessionID_Max=100
```

```
NAE_Properties_Conf_Filename=/opt/safenetConf/64/8.3.1/IngrianNAE.proper
ties
```



8. Save changes.
9. Search for the following properties and set the values as follows:

```
ranger.kms.keysecure.enabled = true

ranger.kms.keysecure.UserPassword.Authentication = true

ranger.kms.keysecure.masterkey.name = MasterKey1

ranger.kms.keysecure.login.username = keysecure-username

ranger.kms.keysecure.login.password = keysecure-password

ranger.kms.keysecure.masterkey.size = 256

ranger.kms.keysecure.sunpkcs11.cfg.filepath = /opt/safenetConf/64/8.3.1/s
unpkcs11.cfg
```

10. Save changes.
11. Restart Ranger KMS.

Results

The master key with alias MasterKey1 is created in KeySecure.

What to do next

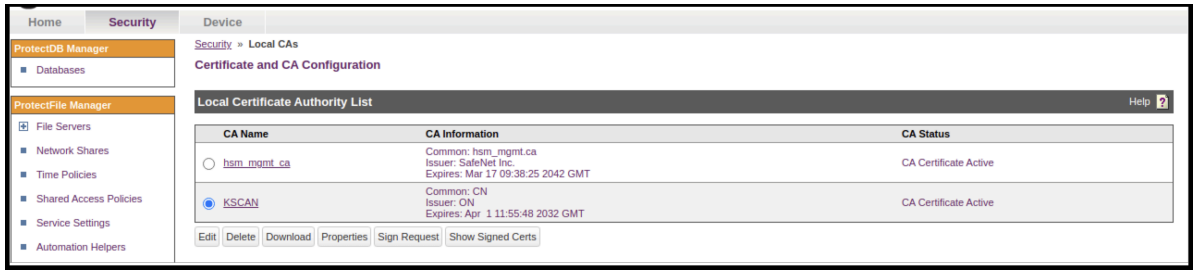
Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Migrating the master key from Ranger KMS DB to KeySecure

How to migrate the Ranger KMS DB Master Key to SafeNet KeySecure HSM. These steps need to be performed when the cluster is ready with all the required services and encryption zone keys are present in the Ranger KMS DB.

Procedure

1. Once Ranger KMS is installed and running, note down the KMS current running process directory.
2. Stop the Ranger KMS service.
3. If SSL is enabled on KeySecure ,then download the CA certificate file (e.g. KSCAN.crt) created on key secure cluster while configuring SSL on keysecure instance.
   a) Log in to KeySecure.
   b) Click on Security tab and then go to Device CAs and SSL Certificates section
   c) Click on Local CA's link
   d) Select the appropriate certificate and click on download as shown in the below image.



   e) Once the certificate is downloaded, copy/scp it to the Ranger KMS host at location "/etc/security/serverKeys/". Make sure the Ranger KMS user has read-write access to the file.

```
chown  kms:kms /etc/security/serverKeys/KSCAN.crt
```

```
chmod  755 /etc/security/serverKeys/KSCAN.crt
```

4. Create a directory and copy the files IngrianNAE.properties, libIngPKCS11.so and sunpkcs11.cfg from Gemalto SafeNet KeySecure to the directory.

```
mkdir -p /opt/safenetConf/64/8.3.1
```

   and then copy the above mentioned files under this location.

   Note: Make sure you have provided read-write access (chmod 775) to Ranger KMS service users (chown kms:kms) to all the 3 files.

5. Update the IngrianNAE.properties file and initialize the below mentioned properties.

```
NAE_IP = Your SafeNet KeySecure IP address (Provide public IP if its on a
 different network.)
```

```
NAE_Port=9000 (should be the same port provided on KeySecure instance un
der Device tab)
```

```
Protocol=tcp (valid values ssl or tcp, should be the same protocol provi
ded on KeySecure instance)
```

6. Initialize the following property only when the protocol of the KeySecure instance is SSL. This is the location of the certificate downloaded from KeySecure and copied on Ranger KMS host.

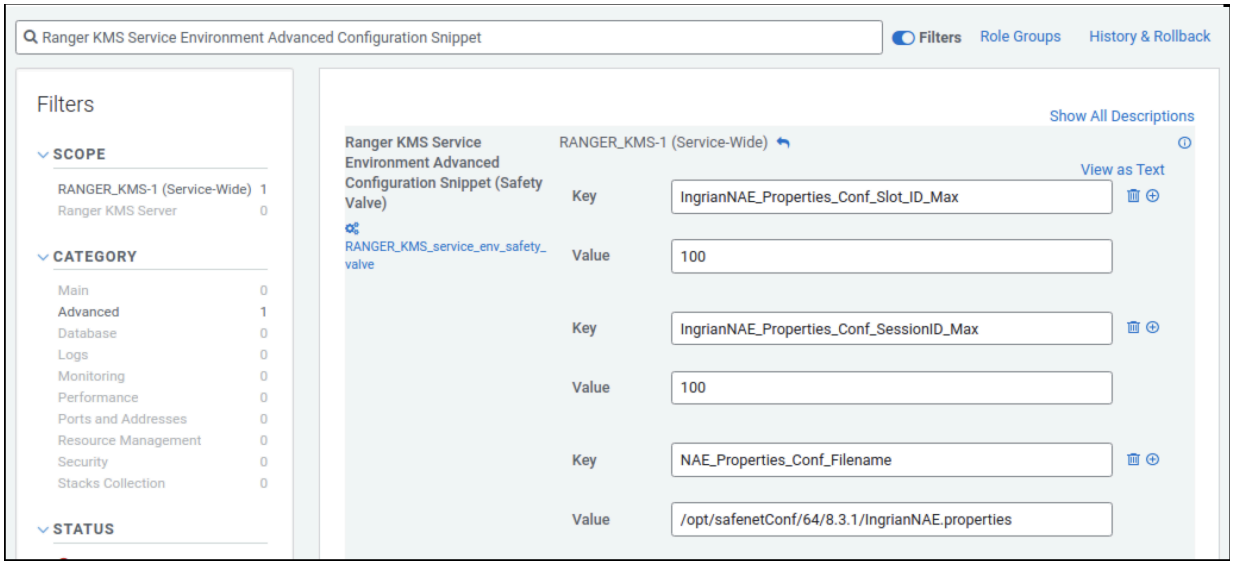
```
CA_File=/etc/security/serverKeys/KSCAN.crt
```

7. Go to CM Ranger KMS Configuration Search for Ranger KMS Server Advanced Configuration Snippet and click on + icon to add the properties.
8. Add the following properties:

```
IngrianNAE_Properties_Conf_Slot_ID_Max=100
```

```
IngrianNAE_Properties_Conf_SessionID_Max=100
```

```
NAE_Properties_Conf_Filename=/opt/safenetConf/64/8.3.1/IngrianNAE.proper
ties
```



9. Save changes.
10. Go to Ranger KMS home directory and export the environment variables.

```
export JAVA_HOME=/usr/java/jdk1.8.0_232-cloudera
export RANGER_KMS_HOME=/opt/cloudera/parcels/CDH/lib/ranger-kms
export RANGER_KMS_CONF=/var/run/cloudera-scm-agent/process/current_proc
sess_dir-RANGER_KMS_SERVER/conf
export SQL_CONNECTOR_JAR=/path/to/SQL-Connector.jar
export HADOOP_CREDSTORE_PASSWORD=hadoop_cred_store_password
```

11. Run the DBMKTOKEYSECURE.sh script.

```
./DBMKTOKEYSECURE.sh masterKey-name keysecure-UserName keysecure-passwor
d /path/to/sunpkcs/config/file/sunpkcs11.cfg
```

where keySecureMasterKeyName : Name of the key which needs to be created on KeySecure , keySecureUsername : User created on KeySecure cluster , and keySecurePassword : Password of the user

Master Key from Ranger KMS DB has been successfully imported into CipherTrust Manager.

12. Go to CM Ranger KMS Configuration. Search for the following properties and set the values.

```
ranger.kms.keysecure.enabled = true
ranger.kms.keysecure.UserPassword.Authentication = true
ranger.kms.keysecure.masterkey.name = masterkeyname (must be the same name
 used above with migration utility ./DBMKTOKEYSECURE.sh )
ranger.kms.keysecure.login.username = keysecure-username
ranger.kms.keysecure.login.password = keysecure-password
ranger.kms.keysecure.masterkey.size=256
```

```
ranger.kms.keysecure.sunpkcs11.cfg.filepath=/opt/safenetConf/64/8.3.1/s
unpkcs11.cfg
```

13. Restart Ranger KMS.

Results
Master Key from Ranger KMS DB has been successfully migrated to CipherTrust Manager.

What to do next
Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Migrating the Master Key from KeySecure HSM to Ranger KMS DB
How to migrate the Master key from KeySecure HSM to Ranger KMS DB. These steps need to be performed when the cluster is ready with all the required services and encryption zone keys are present in the Ranger KMS DB.

Procedure
1. Ensure Ranger KMS is running.
2. Find the KMS current process directory.

```
ps -ef | grep proc_rangerkms
```

3. Stop Ranger KMS service from CM.
4. Go to the Ranger KMS home directory and export the following environment variables.

```
export JAVA_HOME = /usr/java/jdk1.8.0_232-cloudera
export RANGER_KMS_HOME = /opt/cloudera/parcels/CDH/lib/ranger-kms
export RANGER_KMS_CONF = /var/run/cloudera-scm-agent/process/current_pr
ocsess_dir-RANGER_KMS_SERVER/conf
export SQL_CONNECTOR_JAR = /path/to/SQL-Connector.jar
export HADOOP_CREDSTORE_PASSWORD = hadoop_cred_store_password
```

5. Run the ./KEYSECUREMKTOKMSDB script by passing the master key password.

```
./KEYSECUREMKTOKMSDB masterKeyPassword
```

   Master Key from Key Secure has been successfully imported into Ranger KMS DB.
6. Once the script runs successfully, go to CM Ranger KMS Configuration. Search for dbks-site.xml and update the value of ranger.kms.keysecure.enabled to false.
7. Search for ranger.db.encrypt.key.password and set its value to the master-key password which you used above while invoking the migration script.
8. Restart Ranger KMS.

Note: While restarting KMS service, if seeing this error “[op=KeyNameQuery] [] Unsupported custom attribute type for key names query” in the keysecure log , then SSH to KMS host , edit IngrianNAE.properties file , and set the value of Product_Code=HNWs to empty i.e. - Product_Code=

What to do next
Once Ranger KMS has started successfully, verify zone key creation, and zone encryption/decryption.

Connecting KeySecure HSM to CipherTrust Manager after migration from Key Secure HSM
How to configure the KeySecure HSM to connect to CipherTrust.

About this task

After the Thales team successfully migrates the keys from Key Secure HSM to CipherTrust Manager, you must configure the Key HSM to connect to CipherTrust. You must perform the following steps on both the Active and Passive KTS nodes.

Before you begin

The Thales team must have successfully migrated the keys from Key Secure HSM to CipherTrust Manager.

Procedure

1. Stop the Key HSM service.

```
$ service keyhsm stop
```

2. Back up the existing application.properties file.
3. If SSL is enabled on CipherTrust Manager, run the following command:

```
$ echo "thales_machine_ip nae.keysecure.local" >> /etc/hosts
```

4. Set up the Key HSM service.

```
$ keyhsm setup keysecure
```

```
-- Configuring keyHsm General Setup --
Cloudera Recommends to use 127.0.0.1 as the listener port for Key HSM
Please enter Key HSM SSL listener IP address: [127.0.0.1]
Will attempt to setup listener on 127.0.0.1
Please enter Key HSM SSL listener PORT number: 9090

validate Port:                                    :[ Successful ]


 -- Ingrian HSM Credential Configuration --
Please enter HSM login USERNAME: testuser (user created on CipherTrust
Manager)
Please enter HSM login PASSWORD:

Please enter HSM IP Address or Hostname: ec2-3-144-233-194.us-east-2.com
pute.amazonaws.com
Please enter HSM Port number: 9000
Valid address:                                    :[ Successful ]

Use SSL? [Y/n] (As per the configuration done on CipherTrust Manager)


Configuration saved in 'application.properties' file
Configuration stored in: 'application.properties'. (Note: You can also use
 keyhsm settings to quickly view your current configuration)
```

5. Validate the Key HSM.

```
$ service keyhsm validate
```

```
Check Key HSM is stopped                          :[ Successful ]
Configuration Available                           :[ Successful ]
Port 127.0.0.1:9090 available                     :[ Successful ]
Unlimited-Strength JCE                            :[ Successful ]
Validate cipher list                              :[ Successful ]
HSM availability                                  :[ Successful ]
```

```
All services available:                                 :[ Successful ]
```

6. Start the Key HSM service.

```
$ service keyhsm start
Starting KeyHSM, please wait...
```

   The KeyHSM service is started.

7. Configure Key HSM to trust KTS by providing the full path to the file.

```
$ keyhsm trust /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.
pem
```

8. Configure KTS to trust the Key HSM server.

```
$ ktadmin keyhsm --server http://127.0.0.1:9090 --trust
```

   Note: Perform the next two steps once above steps are successfully done on both Active and Passive KTS nodes.

9. Restart KTS from Cloudera Manager.
10. Restart Ranger KMS KTS service from Cloudera Manager.

Using the Ranger Key Management Service

Ranger KMS can be accessed by logging into the Ranger web UI as the KMS administrator.

Role Separation

Ranger uses separate admin users for Ranger and Ranger KMS.

- The Ranger admin user manages Ranger access policies.
- The Ranger KMS admin user (keyadmin by default) manages access policies and keys for Ranger KMS, and has access to a different set of UI features than the Ranger admin user.

Using separate administrator accounts for Ranger and Ranger KMS separates encryption work (encryption keys and policies) from cluster management and access policy management.

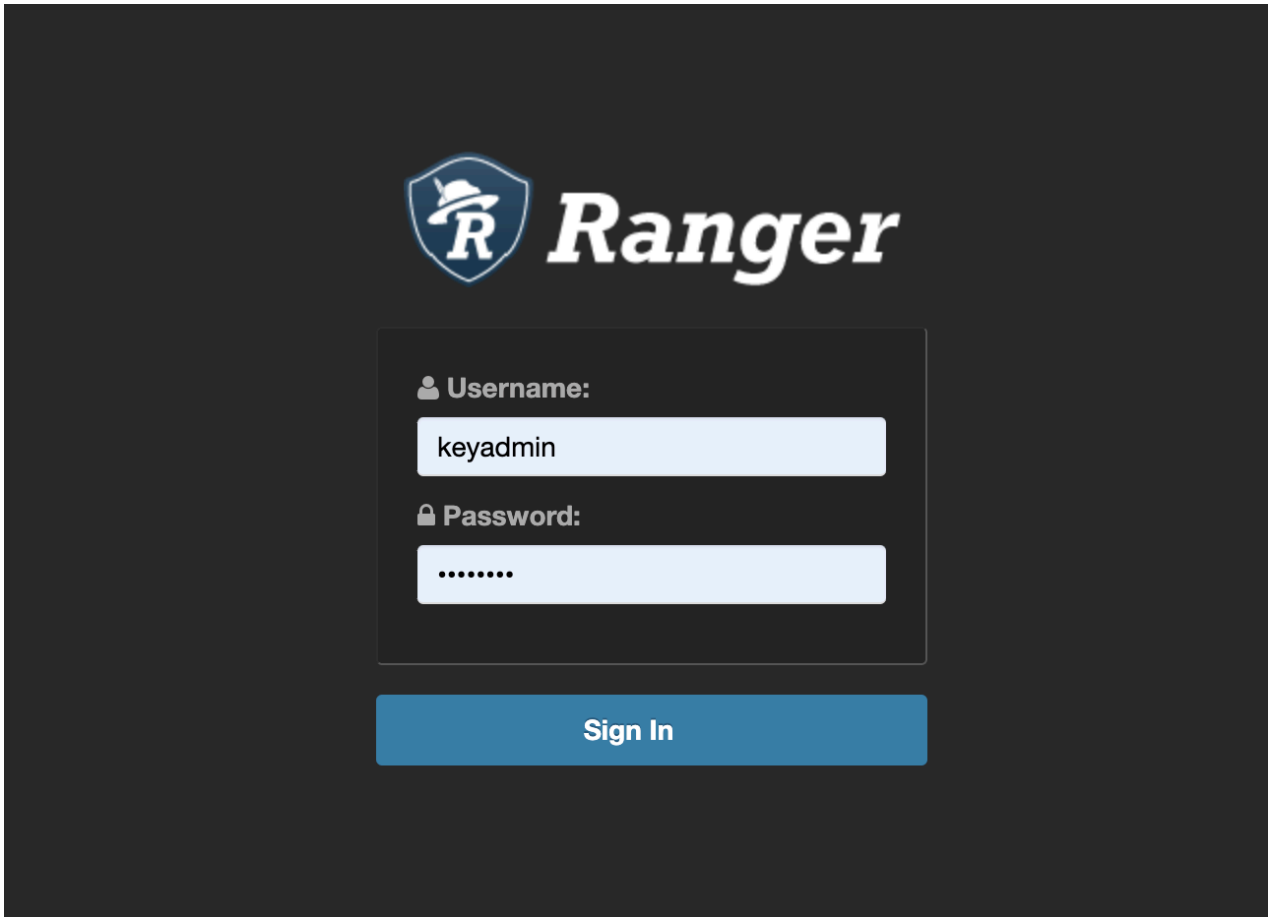
Note:

For more information about creating, deleting, listing, and rolling over existing keys using Ranger REST APIs, see https://ranger.apache.org/apidocs/resource_XKeyREST.html.

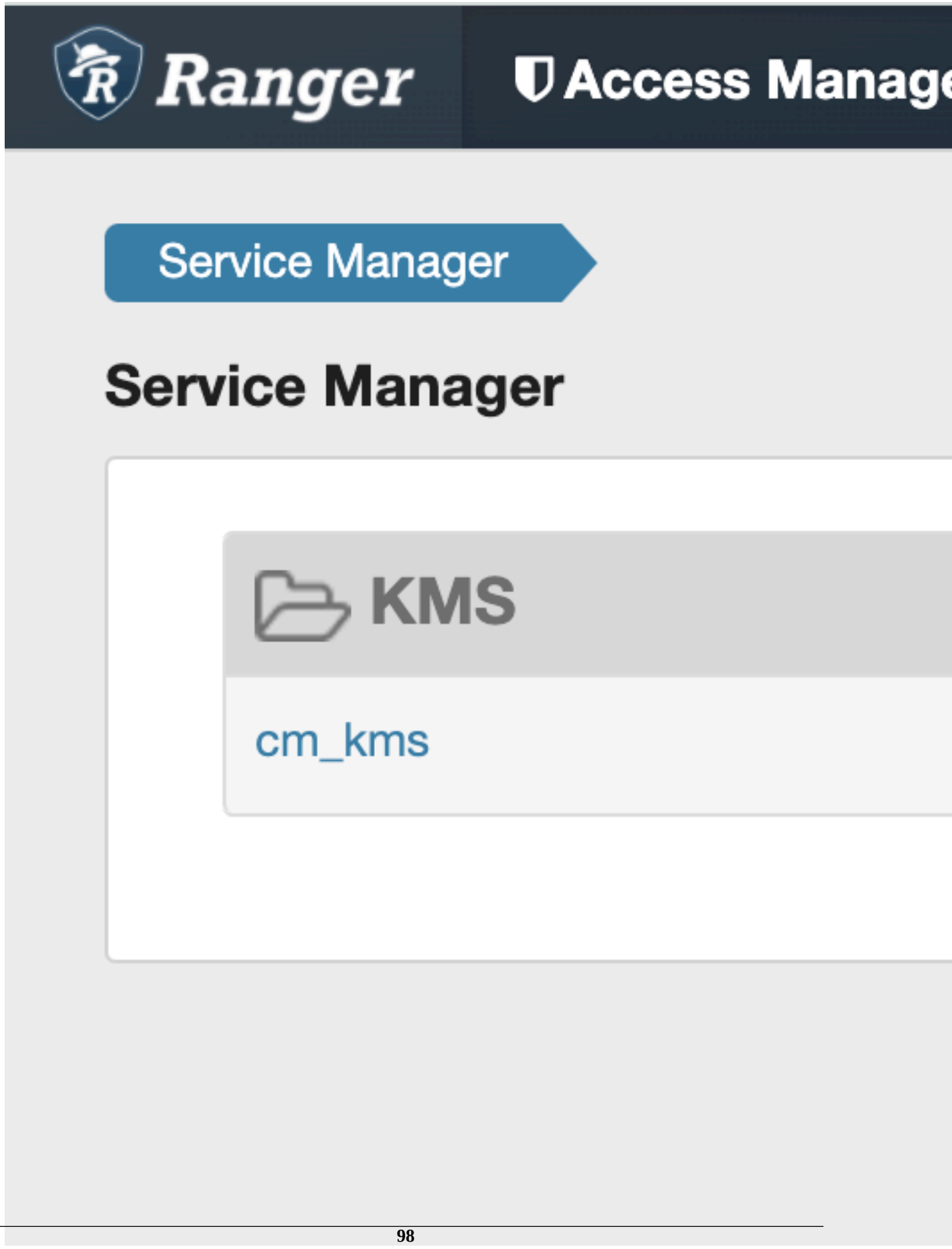
Accessing the Ranger KMS Web UI

How to access the Ranger Key Management Service web UI.

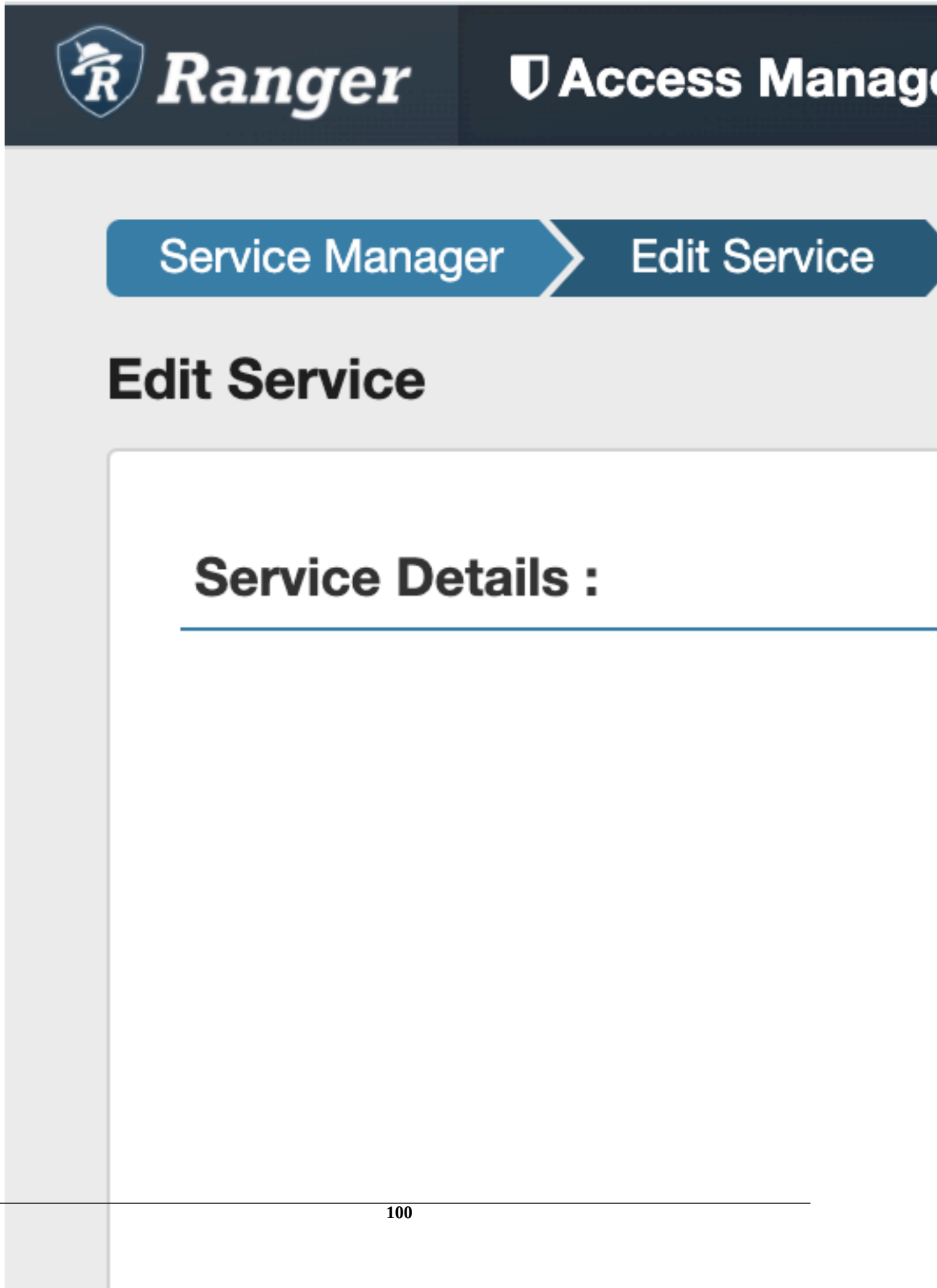
To access Ranger KMS, click the Ranger Admin web UI link, enter your Ranger KMS admin user name and password, then click Sign In.



After logging in, the Service Manager page appears.


Ranger
Access Manage
Service Manager
Service Manager
KMS
cm_kms

To edit Ranger KMS repository properties, click the Edit icon for the service and update the settings on the Edit Service page.

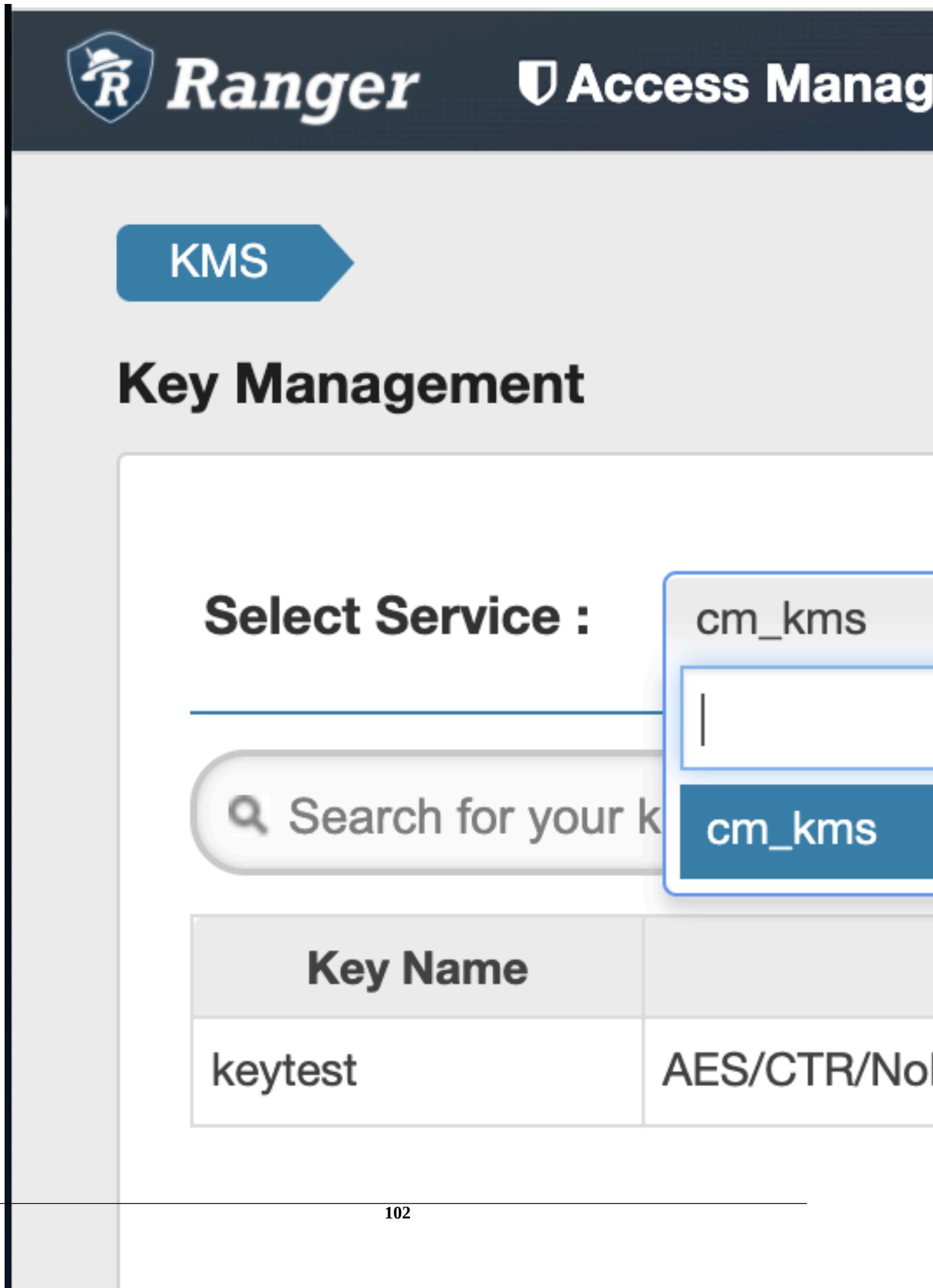

Ranger
Access Manag
Service Manager
Edit Service
Edit Service
Service Details :

List and Create Keys

How to list and create Ranger KMS keys.

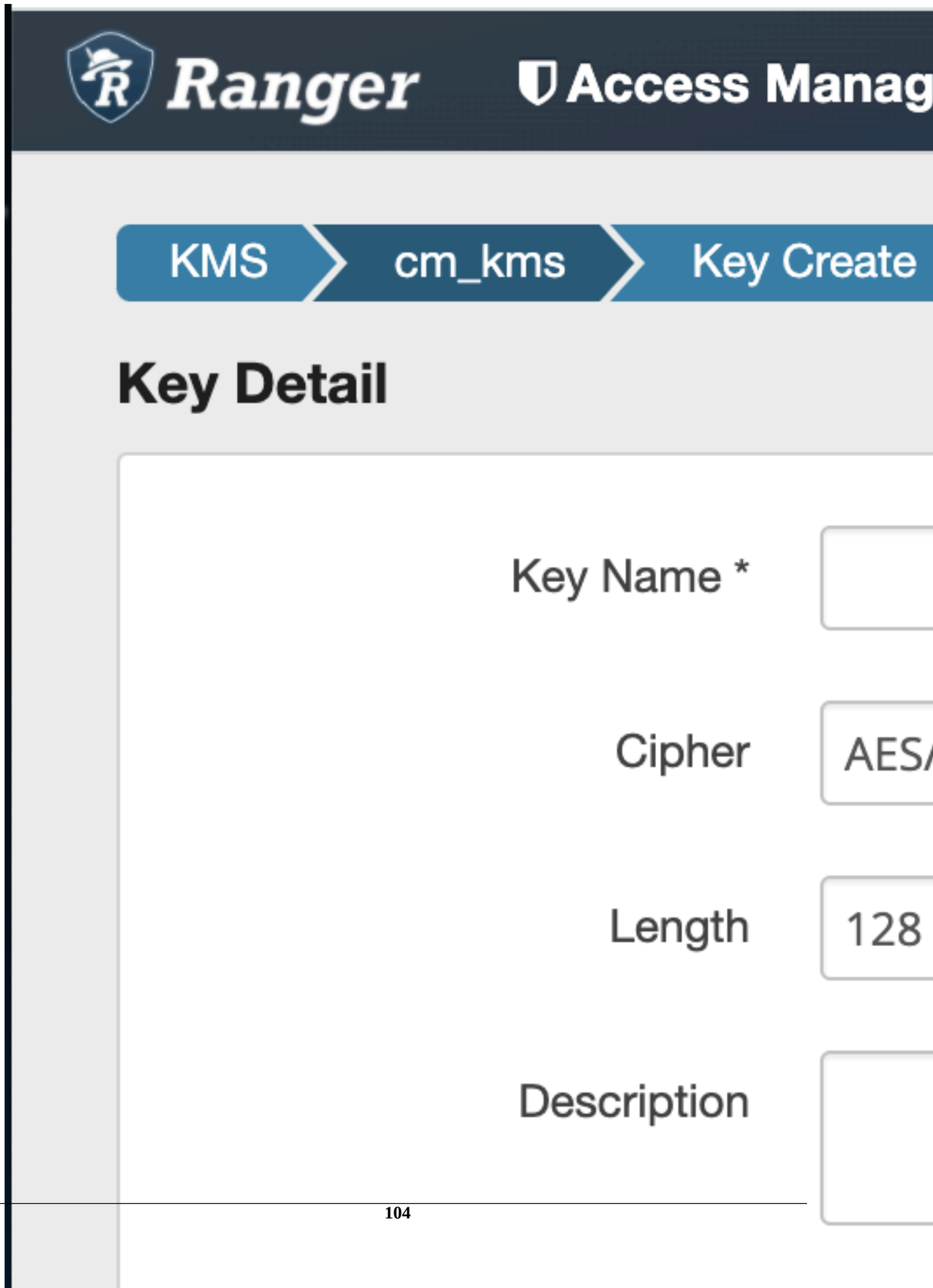
List existing keys

1. Log in to Ranger as the Ranger KMS admin user.
2. Click Encryption in the top menu to display the Key Management page.
3. Use the Select Service box to Select a Ranger KMS service. The keys for the service are dislayed.


Ranger
Access Manag
KMS
Key Management
Select Service :
cm_kms
Search for your k
cm_kms
Key Name
keytest
AES/CTR/No

Create a new key

1. Click Add New Key.
2. On the Key Detail page, add a valid key name.
3. Specify a cipher. Ranger KMS supports AES/CTR/NoPadding as the cipher suite.
4. Specify the key length: 128 or 256 bits.
5. Add other attributes as needed, then click Save.


Ranger
Access Manag
KMS
cm_kms
Key Create
Key Detail
Key Name *
Cipher
AES
Length
128
Description

Roll Over an Existing Key

How to roll over an existing Ranger KMS key.

About this task

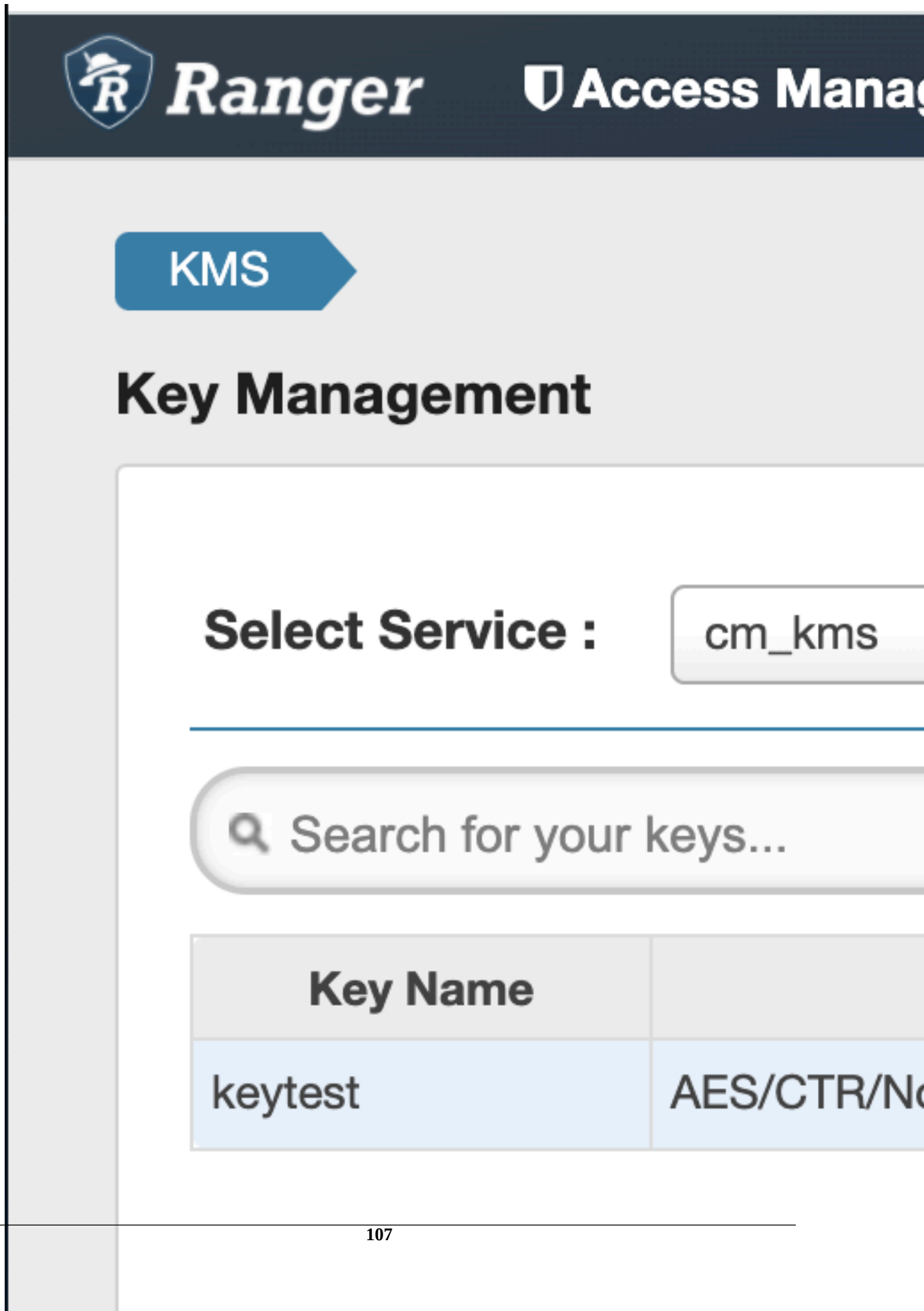
Rolling over (or "rotating") a key retains the same key name, but the key will have a different version. This operation re-encrypts existing file keys, but does not re-encrypt the actual file. Keys can be rolled over at any time.

After a key is rotated in Ranger KMS, new files will have the file key encrypted by the new master key for the encryption zone.

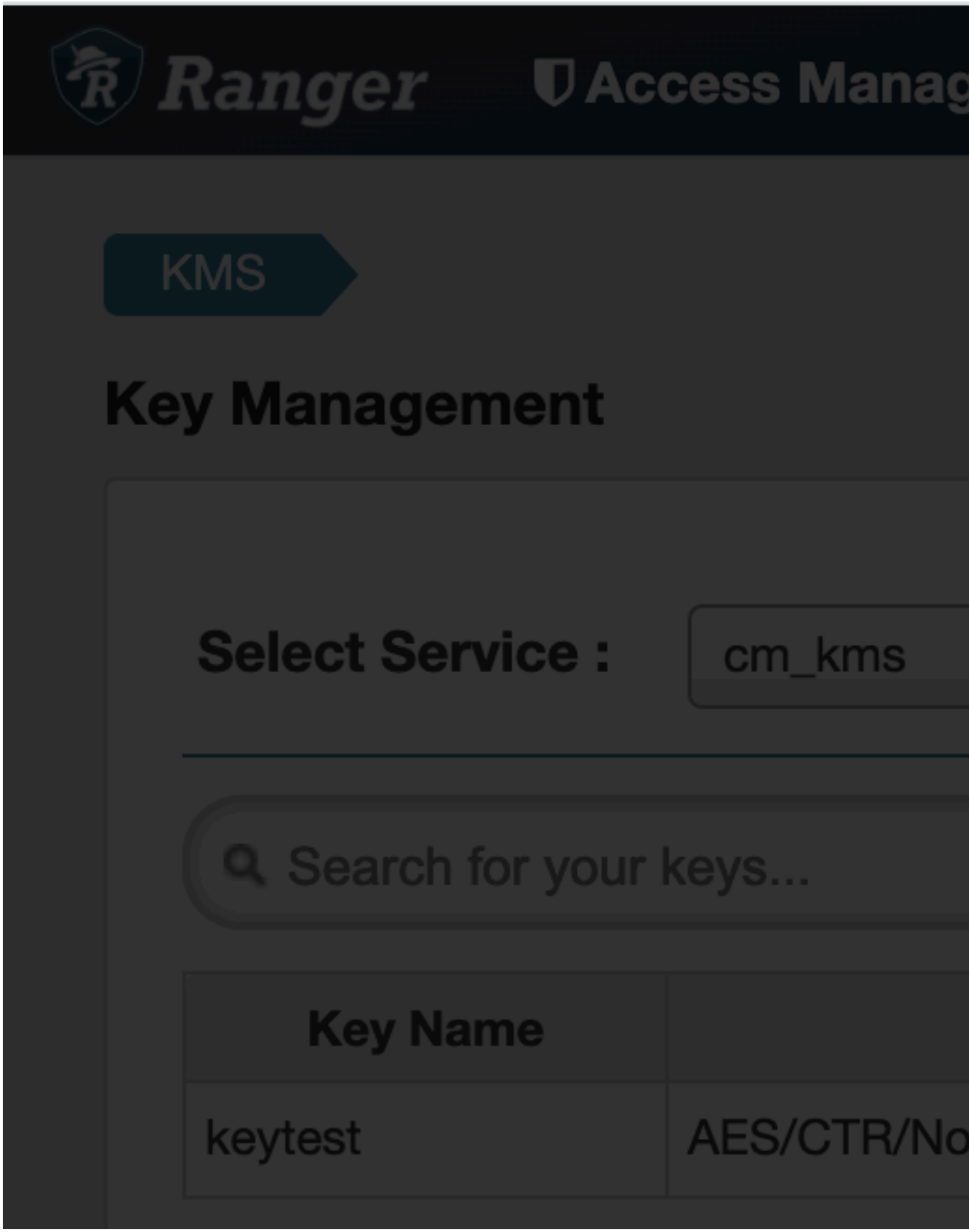
Procedure

1. Log in to Ranger as the Ranger KMS admin user, click Encryption in the top menu, then select a Ranger KMS service.

2. To rotate a key, click the Rollover icon for the key in the Action column.


Ranger
Access Manag
KMS
Key Management
Select Service :
cm_kms
Search for your keys...
Key Name
keytest
AES/CTR/N

3. Click OK on the confirmation pop-up.



Delete a Key

How to delete a Ranger KMS key.

About this task

Important:

Deleting a key associated with an existing encryption zone will result in data loss.

Note:

- Encryption zone keys should be deleted from the Ranger UI or Hadoop Command line.
- Encryption keys should NOT be deleted in the HSM before deleting from the Ranger UI or Hadoop command line.

Procedure

1. Log in to Ranger as the Ranger KMS admin user, click Encryption in the top menu, then select a Ranger KMS service.
2. Click on the Delete icon for the key in the Action column.
3. Click OK on the confirmation pop-up.