

# Encrypting Data at Rest in Cloudera Manager

Date published: 2020-11-30

Date modified: 2020-11-30



# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>Encrypting Data at Rest.....</b>	<b>4</b>
<b>Data at Rest Encryption Reference Architecture.....</b>	<b>4</b>
<b>Data at Rest Encryption Requirements.....</b>	<b>5</b>
<b>Resource Planning for Data at Rest Encryption.....</b>	<b>9</b>
<b>HDFS Transparent Encryption.....</b>	<b>10</b>
Optimizing Performance for HDFS Transparent Encryption.....	14
Enabling HDFS Encryption Using the Wizard.....	15
Enabling HDFS Encryption Using Navigator Key Trustee Server.....	16
Enabling HDFS Encryption Using a Java KeyStore.....	21
Managing Encryption Keys and Zones.....	22
Validating Hadoop Key Operations.....	22
Creating Encryption Zones.....	22
Adding Files to an Encryption Zone.....	23
Deleting Encryption Zones.....	24
Backing Up Encryption Keys.....	24
Rolling Encryption Keys.....	24
Re-encrypting Encrypted Data Encryption Keys (EDEKs).....	26
Configuring the Key Management Server (KMS).....	30
Configuring the KMS Using Cloudera Manager.....	31
Securing the Key Management Server (KMS).....	32
Enabling Kerberos Authentication for the KMS.....	32
Configuring TLS/SSL for the KMS.....	33
Configuring KMS Access Control Lists (ACLs).....	33
Migrating Keys from a Java KeyStore to Cloudera Navigator Key Trustee Server.....	47
Migrating a Key Trustee KMS Server Role Instance to a New Host.....	48
Assumptions and Requirements.....	48
Migrating a Key Trustee KMS Server Role Instance to a New Host.....	49
Configuring CDP Services for HDFS Encryption.....	50
HBase.....	51
Hive.....	51
Hue.....	54
Impala.....	54
MapReduce and YARN.....	55
Search.....	55
Spark.....	56
Sqoop.....	56

## Encrypting Data at Rest

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..

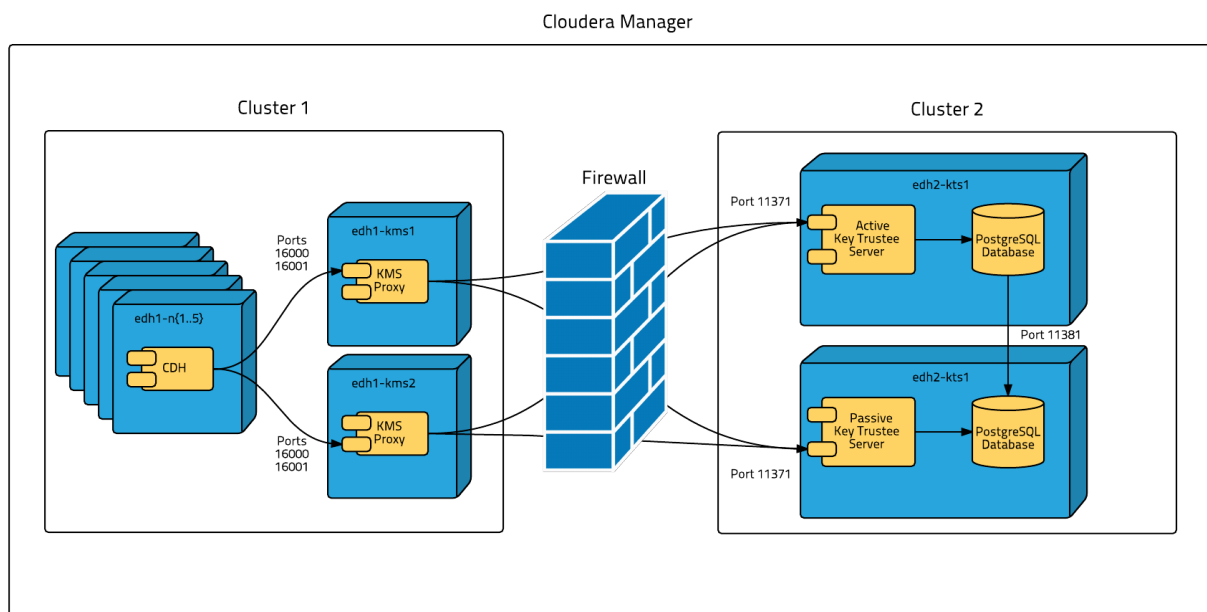
Before configuring encryption for data at rest, familiarize yourself with the requirements in "Data at Rest Encryption Requirements".

### Related Information

[Data at Rest Encryption Requirements](#)

## Data at Rest Encryption Reference Architecture

The following diagram illustrates the supported architecture for deploying Cloudera Navigator encryption for data at rest:



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

- You can restart your EDH 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 Key Trustee 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 Key Trustee 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.

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. Run the following commands on RHEL 6-compatible systems:

```
sudo yum install rng-tools
sudo echo 'EXTRAOPTIONS="-r /dev/urandom"' >> /etc/sysconfig/rngd
sudo service rngd start
sudo chkconfig rngd on
```

For RHEL 7, run the following commands:

```
sudo yum install rng-tools
cp /usr/lib/systemd/system/rngd.service /etc/systemd/system/
sed -i -e 's/ExecStart=\/sbin\/rngd -f/ExecStart=\/sbin\/rngd -f -r \\/dev\/u
random/' /etc/systemd/system/rngd.service
systemctl daemon-reload
systemctl start rngd
systemctl enable rngd
```

Make sure that the hosts running Key Trustee Server, Key Trustee 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 `--skip-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".

## Key Trustee KMS Requirements

### Recommended Hardware and Supported Distributions

The recommended minimum hardware specifications are as follows:

- Processor: 2 GHz 64-bit quad core
- Memory: 16 GB RAM
- Storage: 40 GB on moderate- to high-performance disk drives

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

The Key Trustee 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 Key Trustee KMS 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

[Encrypting Data at Rest](#)

[Resource Planning for Data at Rest Encryption](#)

[AES-NI](#)

[JCE for Java SE 7](#)

[JCE for Java SE 8](#)

[Product Compatibility Matrix for Cloudera Navigator Encryption](#)

[Cloudera Enterprise Upgrade Guide](#)

[Registering Cloudera Navigator Encrypt with Key Trustee Server>Registration Options](#)



# Resource Planning for Data at Rest Encryption

For production environments, you must configure high availability for:

- Key Trustee Server
- Key Trustee KMS

## Key Trustee Server and Key Trustee KMS HA Planning

For high availability, you must provision two dedicated Key Trustee Server hosts and at least two dedicated Key Trustee KMS hosts, for a minimum of four separate hosts. Do not run multiple Key Trustee Server or Key Trustee 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 Key Trustee 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 6 Security Guide
- 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 Key Trustee 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 "Cloudera Navigator Key Trustee Server High Availability".

For information about enabling Key Trustee KMS high availability, refer to "Enabling Key Trustee KMS High Availability".

## 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 Key Trustee 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.

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 6 Security Guide](#)

[RHEL 7 Security Guide](#)

[AES-NI](#)

[Cloudera Navigator Key Trustee Server High Availability](#)

[Enabling Key Trustee KMS 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.

Introduced in CDH 5.3, transparent encryption for HDFS (see "New in CDH 5.3: Transparent Encryption in 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.



**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. See "Encryption Overview" for more information.

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. See "Optimizing Performance for HDFS Transparent Encryption" for details.

Key Concepts and Architecture

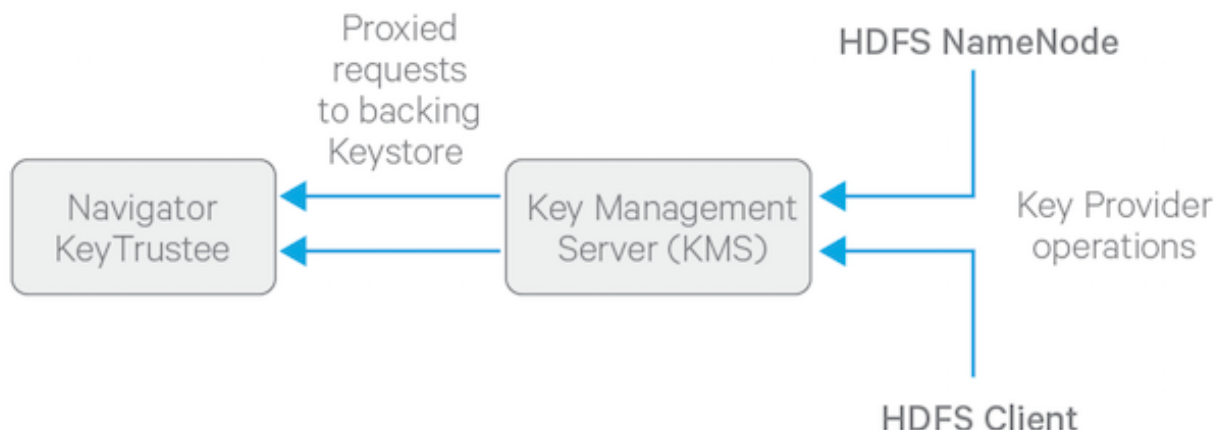
### Keystores and the Hadoop Key Management Server

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 Hadoop Key Management Server (KMS), which serves as a proxy between HDFS clients and the backing keystore. Both the keystore and Hadoop 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:

- Cloudera Navigator Key Trustee Server is a key store for managing encryption keys. To integrate with the Navigator Key Trustee Server, Cloudera provides a custom KMS service, Key Trustee 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 Hadoop Key Management Server. The keystore can be either the Cloudera Navigator Key Trustee Server or a support HSM.



To get started with deploying the KMS and a keystore, see “Enabling HDFS Encryption Using the Wizard”.

For information on configuring and securing the KMS, see “Configuring the Key Management Server (KMS)” and “Securing the Key Management Server (KMS)”

#### KMS Solutions

This section describes the various KMS solutions available, and identifies criteria often used for selecting each.

#### Key Trustee KMS with Key Trustee Server

Choose KT KMS with Key Trustee Server if:

- Enterprise-grade key management is required
- Encryption zone key protection by an HSM is not required

#### Key Trustee KMS with Key Trustee Server and Key HSM

Choose KT KMS with Key Trustee Server and Key HSM if:

- Enterprise-grade key management is required
- Encryption zone key protection by an HSM (as root of trust) is required
- Performance for encryption zone key operations is critical

#### 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.



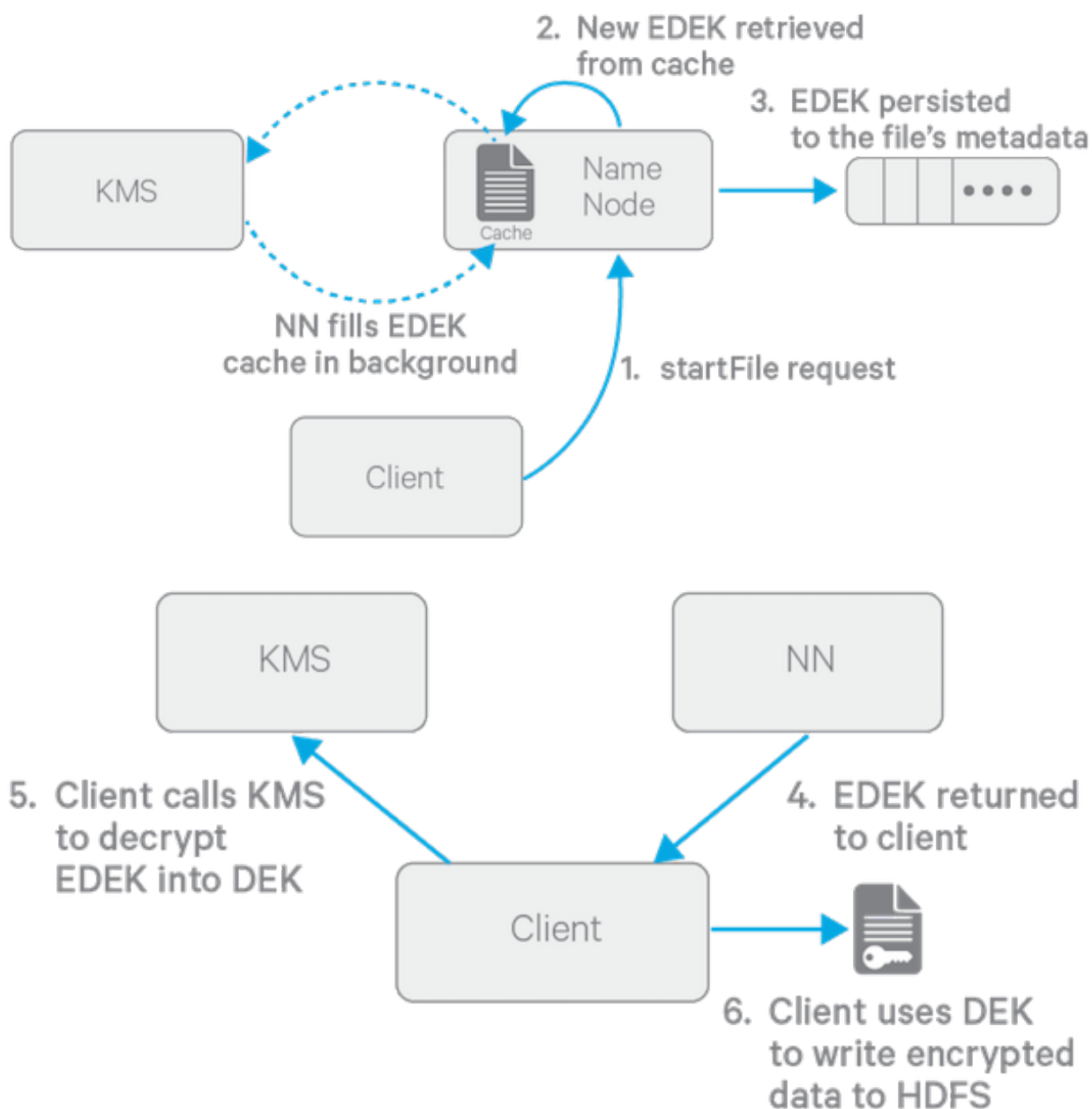
EDEKs are stored persistently on the NameNode as part of each file's metadata, using [HDFS extended attributes \(implemented as of Cloudera CDH 5.2\)](#). EDEKs can be safely stored and handled by the NameNode because the hdfs user does not have access to the EDEK's encryption keys (EZ keys). Even if 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.

For more information on creating and managing encryption zones, see “Managing Encryption Keys and Zones”.

### Accessing Files Within an Encryption Zone

To encrypt a new file, the HDFS client requests a new EDEK from the NameNode. The NameNode 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 HDFS client provides the NameNode with the file's EDEK and the version number of the EZ key that was used to generate the EDEK. The NameNode 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.

### Related Information

[Transparent Encryption in HDFS](#)

[AES-NI Standard Instructions](#)

[Java Cryptography Extension \(JCE\) Unlimited Strength Jurisdiction Policy Files 7 Download](#)

[Optimizing Performance for HDFS Transparent Encryption](#)

[Enabling HDFS Encryption Using the Wizard](#)

[Configuring the Key Management Server \(KMS\)](#)

[Securing the Key Management Server \(KMS\)](#)

[Managing Encryption Keys and Zones](#)

## Optimizing Performance for HDFS Transparent Encryption



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

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.

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/libcrypto.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.

### Ubuntu Precise, Ubuntu Bionic, and Ubuntu Trusty

Install the `libssl-devel` package on all clients: `sudo apt-get install libssl-dev`. This installs OpenSSL 1.1.1 which is supported.

## Enabling HDFS Encryption Using the Wizard

To accommodate the security best practice of [separation of duties](#), enabling HDFS encryption using the wizard requires different Cloudera Manager user roles for different steps.

Launch the Set up HDFS Data At Rest Encryption wizard in one of the following ways:

- Cluster Set up HDFS Data At Rest Encryption  
Minimum Required Role: [Key Administrator](#) or [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.
- AdministrationSecuritySet up HDFS Data At Rest Encryption  
Minimum Required Role: [Key Administrator](#) or [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.
- HDFS serviceActionsSet up HDFS Data At Rest Encryption  
Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

On the first page of the wizard, select the root of trust for encryption keys:

- Cloudera Navigator Key Trustee Server
- A file-based password-protected Java KeyStore

Cloudera strongly recommends using Cloudera Navigator Key Trustee Server as the root of trust for production environments. The file-based Java KeyStore root of trust is insufficient to provide the security, scalability, and manageability required by most production systems. More specifically, the Java KeyStore KMS does not provide:

- Scalability, so you are limited to only one KMS, which can result in bottlenecks
- High Availability (HA)
- Recoverability, so if you lose the node where the Java KeyStore is stored, then you can lose access to all the encrypted data

Ultimately, the Java KeyStore does not satisfy the stringent security requirements of most organizations for handling master encryption keys.

Choosing a root of trust displays a list of steps required to enable HDFS encryption using that root of trust. Each step can be completed independently. The Status column indicates whether the step has been completed, and the Notes column provides additional context for the step. If your Cloudera Manager user account does not have sufficient privileges to complete a step, the Notes column indicates the required privileges.

Available steps contain links to wizards or documentation required to complete the step. If a step is unavailable due to insufficient privileges or a prerequisite step being incomplete, no links are present and the Notes column indicates the reason the step is unavailable.

Continue to the section for your selected root of trust for further instructions:

### Related Information

[Encrypting Data at Rest](#)

[Data at Rest Encryption Requirements](#)

[Data at Rest Encryption Reference Architecture](#)

[Resource Planning for Data at Rest Encryption](#)

[Configuring KMS Access Control Lists \(ACLs\)](#)

## Enabling HDFS Encryption Using Navigator Key Trustee Server

Enabling HDFS encryption using Key Trustee Server as the key store involves multiple components. For an overview of the components involved in encrypting data at rest, see “Encrypting Data at Rest”. For guidelines on deploying the Navigator Key Trustee Server in production environments, “Data at Rest Encryption Requirements”.

Before continuing, make sure the Cloudera Manager server host has access to the internal repository hosting the Key Trustee Server software.

After selecting Cloudera Navigator Key Trustee Server as the root of trust, the following steps are displayed:

### 1. Enable Kerberos

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

### 2. Enable TLS/SSL

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

### 3. Add a dedicated cluster for the Key Trustee Server

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

If you haven't already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.



This step creates a new cluster in Cloudera Manager for the Key Trustee Server hosts to isolate them from other enterprise data hub (EDH) services for increased security and durability. For more information, see “Data at Rest Encryption Reference Architecture”.

To complete this step:

1. Click Add a dedicated cluster for the Key Trustee Server.
2. Leave Enable High Availability checked to add two hosts to the cluster. For production environments, you must enable high availability for Key Trustee Server. Failure to enable high availability can result in complete data loss in the case of catastrophic failure of a standalone Key Trustee Server. Click Continue.
3. Search for new hosts to add to the cluster, or select the Currently Managed Hosts tab to add existing hosts to the cluster. After selecting the hosts, click Continue.
4. Select the KEYTRUSTEE\_SERVER parcel to install Key Trustee Server using parcels, or select None if you want to use packages. If you do not see a parcel available, click More Options and add the repository URL to the Remote Parcel Repository URLs list. After selecting a parcel or None, click Continue.

If you selected None, click Continue again, and skip to [4. Install Key Trustee Server binary using packages or parcels](#) on page 17.

5. After the KEYTRUSTEE\_SERVER parcel is successfully downloaded, distributed, unpacked, and activated, click Continue.
6. Click Continue to complete this step and return to the main page of the wizard.

#### 4. Install Key Trustee Server binary using packages or parcels

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.



**Note:** If you selected None on the parcel selection page in step 3. [Add a dedicated cluster for the Key Trustee Server](#) on page 16, the step title is changed to Install Parcel for Key Trustee Server. If you are using packages, skip this step and see “Installing Cloudera Navigator Key Trustee Server” for package-based installation instructions. After installing Key Trustee Server using packages, continue to [5. Install Parcel for Key Trustee KMS](#) on page 17.

If you haven't already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.

This step is completed automatically during [3. Add a dedicated cluster for the Key Trustee Server](#) on page 16 if you are using parcels. If the step is incomplete for any reason (such as the wizard being interrupted or a failure installing the parcel), complete it manually:

1. Click Install Key Trustee Server binary using packages or parcels.
2. Select the KEYTRUSTEE\_SERVER parcel to install Key Trustee Server, or select None if you need to install Key Trustee Server manually using packages. If you do not see a parcel available, click More Options and add the repository URL to the Remote Parcel Repository URLs list. After selecting a parcel, click Continue.
3. After the KEYTRUSTEE\_SERVER parcel is successfully downloaded, distributed, unpacked, and activated, click Finish to complete this step and return to the main page of the wizard.

#### 5. Install Parcel for Key Trustee KMS

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

If you haven't already done so, you must create an internal repository to install Cloudera Navigator before you can set up and use Navigator Key Trustee Server.

This step installs the Key Trustee KMS parcel. If you are using packages, skip this step and see “Installing Cloudera Navigator Key Trustee Server” for instructions. After installing Key Trustee KMS using packages, continue to [6. Add a Key Trustee Server Service](#) on page 18.

To complete this step for parcel-based installations:

1. Click Install Parcel for Key Trustee KMS.
2. Select the KEYTRUSTEE parcel to install Key Trustee KMS. If you do not see a parcel available, click More Options and add the repository URL to the Remote Parcel Repository URLs list. After selecting a parcel, click Continue.
3. After the KEYTRUSTEE parcel is successfully downloaded, distributed, unpacked, and activated, click Finish to complete this step and return to the main page of the wizard.

## 6. Add a Key Trustee Server Service

Minimum Required Role: [Key Administrator](#) (also provided by Full Administrator)

This step adds the Key Trustee Server service to Cloudera Manager. To complete this step:

1. Click Add a Key Trustee Server Service.
2. Click Continue.
3. On the Customize Role Assignments for Key Trustee Server page, select the hosts for the Active Key Trustee Server and Passive Key Trustee Server roles. Make sure that the selected hosts are not used for other services (see “Resource Planning for Data at Rest Encryption” for more information), and click Continue.
4. The Entropy Considerations page provides commands to install the rng-tools package to increase available entropy for cryptographic operations. For more information, see “Data at Rest Encryption Requirements”. After completing these commands, click Continue.
5. The Synchronize Active and Passive Key Trustee Server Private Keys page provides instructions for generating and copying the Active Key Trustee Server private key to the Passive Key Trustee Server. Cloudera recommends following security best practices and transferring the private key using offline media, such as a removable USB drive. For convenience (for example, in a development or testing environment where maximum security is not required), you can copy the private key over the network using the provided rsync command.

After you have synchronized the private keys, run the `ktadmin init` command on the Passive Key Trustee Server as described in the wizard. After the initialization is complete, check the box to indicate you have synchronized the keys and click Continue in the wizard.

6. The Setup TLS for Key Trustee Server page provides instructions on replacing the auto-generated self-signed certificate with a production certificate from a trusted Certificate Authority (CA). For more information, see “Managing Key Trustee Server Certificates”. Click Continue to view and modify the default certificate settings.
7. On the Review Changes page, you can view and modify the following settings:

- Database Storage Directory (db\_root)

Default value: `/var/lib/keytrustee/db`

The directory on the local filesystem where the Key Trustee Server database is stored. Modify this value to store the database in a different directory.

- Active Key Trustee Server TLS/SSL Server Private Key File (PEM Format) (ssl.privatekey.location)

Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem`

The path to the Active Key Trustee Server TLS certificate private key. Accept the default setting to use the auto-generated private key. If you have a CA-signed certificate, change this path to the CA-signed certificate private key file. This file must be in [PEM](#) format.

- Active Key Trustee Server TLS/SSL Server Certificate File (PEM Format) (ssl.cert.location)

Default value: `/var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem`

The path to the Active Key Trustee Server TLS certificate. Accept the default setting to use the auto-generated self-signed certificate. If you have a CA-signed certificate, change this to the path to the CA-signed certificate. This file must be in PEM format.

- Active Key Trustee Server TLS/SSL Server CA Certificate (PEM Format) (ssl.cacert.location)

Default value: (none)

The path to the file containing the CA certificate and any intermediate certificates (if any intermediate certificates exist, then they are required here) used to sign the Active Key Trustee Server certificate. If you

have a CA-signed certificate, set this value to the path to the CA certificate or certificate chain file. This file must be in PEM format.

- Active Key Trustee Server TLS/SSL Private Key Password (ssl.privatekey.password)

Default value: (none)

The password for the Active Key Trustee Server private key file. Leave this blank if the file is not password-protected.

- Passive Key Trustee Server TLS/SSL Server Private Key File (PEM Format) (ssl.privatekey.location)

Default value: /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee-pk.pem

The path to the Passive Key Trustee Server TLS certificate private key. Accept the default setting to use the auto-generated private key. If you have a CA-signed certificate, change this path to the CA-signed certificate private key file. This file must be in [PEM](#) format.

- Passive Key Trustee Server TLS/SSL Server Certificate File (PEM Format) (ssl.cert.location)

Default value: /var/lib/keytrustee/.keytrustee/.ssl/ssl-cert-keytrustee.pem

The path to the Passive Key Trustee Server TLS certificate. Accept the default setting to use the auto-generated self-signed certificate. If you have a CA-signed certificate, change this to the path to the CA-signed certificate. This file must be in PEM format.

- Passive Key Trustee Server TLS/SSL Server CA Certificate (PEM Format) (ssl.cacert.location)

Default value: (none)

The path to the file containing the CA certificate and any intermediate certificates (if any intermediate certificates exist, then they are required here) used to sign the Passive Key Trustee Server certificate. If you have a CA-signed certificate, set this value to the path to the CA certificate or certificate chain file. This file must be in PEM format.

- Passive Key Trustee Server TLS/SSL Private Key Password (ssl.privatekey.password)

Default value: (none)

The password for the Passive Key Trustee Server private key file. Leave this blank if the file is not password-protected.

After reviewing the settings and making any changes, click Continue.

8. After all commands complete successfully, click Continue. If the Generate Key Trustee Server Keyring appears stuck, make sure that the Key Trustee Server host has enough entropy. See “Data at Rest Encryption Requirements” for more information.
9. Click Finish to complete this step and return to the main page of the wizard.

## 7. Add a Key Trustee KMS Service

Minimum Required Role: [Key Administrator](#) (also provided by Full Administrator)

This step adds a Key Trustee KMS service to the cluster. The Key Trustee KMS service is required to enable HDFS encryption to use Key Trustee Server for cryptographic key management. Key Trustee KMS high availability uses ZooKeeper to automatically configure load balancing.



**Note:** Before performing this step, you must have already added hosts to the environment for KMS.

To complete this step:

1. Click Add a Key Trustee KMS Service.
2. Select an existing Key Trustee Server pair or specify an external Key Trustee Server pair. If you have an existing Key Trustee Server pair outside of Cloudera Manager control, select the External Key Trustee Server option and specify the fully-qualified domain names (FQDNs) of the Key Trustee Server pair. Click Continue.

3. Select cluster hosts for the Key Trustee KMS service. For production environments, select at least two hosts for high availability. If you proceed with only one host, you can enable high availability later.

Make sure that the selected hosts are not used for other services (see “Resource Planning for Data at Rest Encryption” for more information), and click Continue.

4. The Entropy Considerations page provides commands to install the rng-tools package to increase available entropy for cryptographic operations. For more information, see “Data at Rest Encryption Requirements”. After completing these commands, click Continue.
5. The Setup Organization and Auth Secret page generates the necessary commands to create an organization in Key Trustee Server. An organization is required to be able to register the Key Trustee KMS with Key Trustee Server.

Enter an organization name and click Generate Instruction. Run the displayed commands to generate an organization and obtain the auth\_secret value for the organization. Enter the secret in the auth\_secret field and click Continue.

6. The Setup Access Control List (ACL) page allows you to generate ACLs for the Key Trustee KMS or to provide your own ACLs. To generate the recommended ACLs, enter the username and group responsible for managing cryptographic keys and click Generate ACLs. To specify your own ACLs, select the Use Your Own kms-acls.xml File option and enter the ACLs. For more information on the KMS Access Control List, see “Configuring KMS Access Control Lists (ACLs)”.

After generating or specifying the ACL, click Continue.

7. The Setup TLS for Key Trustee KMS page provides high-level instructions for configuring TLS communication between the Key Trustee KMS and the Key Trustee Server, as well as between the EDH cluster and the Key Trustee KMS. See “Securing the Key Management Server (KMS)” for more information.

Click Continue.

8. The Review Changes page lists all of the settings configured in this step. Click the “?” icon next to any setting for information about that setting. Review the settings and click Continue.
9. After the First Run commands have successfully completed, click Continue.
10. The Synchronize Private Keys and HDFS Dependency page provides instructions for copying the private key from one Key Management Server Proxy role to all other roles.



#### Caution:

It is very important that you perform this step. Failure to do so leaves Key Trustee KMS in a state where keys are intermittently inaccessible, depending on which Key Trustee KMS host a client interacts with, because cryptographic key material encrypted by one Key Trustee KMS host cannot be decrypted by another. If you are already running multiple Key Trustee KMS hosts with different private keys, immediately back up (“Backing up Key Trustee Server and Clients”) all Key Trustee KMS hosts, and contact Cloudera Support for assistance correcting the issue.

To determine whether the Key Trustee KMS private keys are different, compare the MD5 hash of the private keys. On each Key Trustee KMS host, run the following command:

```
md5sum /var/lib/kms-keytrustee/keytrustee/.keytrustee/secring.gpg
```

If the outputs are different, contact Cloudera Support for assistance. Do not attempt to synchronize existing keys. If you overwrite the private key and do not have a backup, any keys encrypted by that private key are permanently inaccessible, and any data encrypted by those keys is permanently irretrievable. If you are configuring Key Trustee KMS high availability for the first time, continue synchronizing the private keys.

Cloudera recommends following security best practices and transferring the private key using offline media, such as a removable USB drive. For convenience (for example, in a development or testing environment where maximum security is not required), you can copy the private key over the network using the provided rsync command.

After you have synchronized the private keys, check the box to indicate you have done so and click Continue.

11. After the Key Trustee KMS service starts, click Finish to complete this step and return to the main page of the wizard.

For parcel-based Key Trustee KMS releases 5.8 and higher, Cloudera Manager automatically backs up Key Trustee KMS (using the `ktbackup.sh` script) after adding the Key Trustee KMS service. It does not schedule automatic backups using cron. For package-based installations, you must manually back up Key Trustee Server and configure a cron job.

The backup is stored in `/var/lib/kms-keytrustee` in cleartext. For more information about using the backup script and configuring the cron job (including how to encrypt backups), see “Backing up Key Trustee Server and Clients”.

## 8. Restart stale services and redeploy client configuration

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

This step restarts all services which were modified while enabling HDFS encryption. To complete this step:

1. Click Restart stale services and redeploy client configuration.
2. Click Restart Stale Services.
3. Ensure that Re-deploy client configuration is checked, and click Restart Now.
4. After all commands have completed, click Finish.

## 9. Validate Data Encryption

Minimum Required Role: [Key Administrator](#) or [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

This step launches a tutorial with instructions on creating an encryption zone and putting data into it to verify that HDFS encryption is enabled and working.

## Enabling HDFS Encryption Using a Java KeyStore



**Note:** Cloudera strongly recommends using Cloudera Navigator Key Trustee Server as the root of trust for production environments. The file-based Java KeyStore root of trust is insufficient to provide the security, scalability, and manageability required by most production systems.

After selecting A file-based password-protected Java KeyStore as the root of trust, the following steps are displayed:

### 1. Enable Kerberos

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

### 2. Enable TLS/SSL

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

### 3. Add a Java KeyStore KMS Service

Minimum Required Role: [Key Administrator](#) (also provided by Full Administrator)

This step adds the Java KeyStore KMS service to the cluster. The Java KeyStore KMS service uses a password-protected Java KeyStore for cryptographic key management. To complete this step:

1. Click Add a Java KeyStore KMS Service.
2. Select a cluster host for the Java KeyStore KMS service. Click Continue.
3. The Setup TLS for Java KeyStore KMS page provides high-level instructions for configuring TLS communication between the EDH cluster and the Java KeyStore KMS.

Click Continue.

4. The Review Changes page lists the Java KeyStore settings. Click the “?” icon next to any setting for information about that setting. Enter the location and password for the Java KeyStore and click Continue.

5. Click Continue to automatically configure the HDFS service to depend on the Java KeyStore KMS service.
6. Click Finish to complete this step and return to the main page of the wizard.

#### 4. Restart stale services and redeploy client configuration

Minimum Required Role: [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

This step restarts all services which were modified while enabling HDFS encryption. To complete this step:

1. Click Restart stale services and redeploy client configuration.
2. Click Restart Stale Services.
3. Ensure that Re-deploy client configuration is checked, and click Restart Now.
4. After all commands have completed, click Finish.

#### 5. Validate Data Encryption

Minimum Required Role: [Key Administrator](#) or [Cluster Administrator](#) (also provided by Full Administrator) This feature is not available when using Cloudera Manager to manage Data Hub clusters.

This step launches a tutorial with instructions on creating an encryption zone and putting data into it to verify that HDFS encryption is enabled and working.

## Managing Encryption Keys and Zones

Interacting with the KMS and creating encryption zones requires the use of two new CLI commands: `hadoop key` and `hdfs crypto`. The following sections will help you get started with creating encryption keys and setting up encryption zones.

Before continuing, make sure that your KMS ACLs have been set up according to best practices. For more information, see “Configuring KMS Access Control Lists (ACLs)”.

#### Related Information

[Configuring KMS Access Control Lists \(ACLs\)](#)

[Configuring CDP Services for HDFS Encryption](#)

[HDFS Transparent Encryption](#)

## Validating Hadoop Key Operations



**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.

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

```
sudo -u <key_admin> hadoop key create keytrustee_test
hadoop key list
```

## Creating 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.



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.

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

```
sudo -u hdfs hadoop key create <key_name>
```

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

```
sudo -u hdfs hadoop fs -mkdir /encryption_zone
sudo -u hdfs 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:

```
$ sudo -u hdfs 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.

For more information and recommendations on creating encryption zones for each CDP component, see “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:

```
sudo -u hdfs hadoop distcp /user/dir /encryption_zone
```



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

## DistCp Considerations

A common use case for `DistCp` is to replicate data between clusters for backup and disaster recovery purposes. This is typically performed by the cluster administrator, who is an HDFS superuser. To retain this workflow when using HDFS encryption, a new virtual path prefix has been introduced, `/.reserved/raw/`, that gives superusers direct access to the underlying block data in the filesystem. This allows superusers to `distcp` data without requiring access to encryption keys, and avoids the overhead of decrypting and re-encrypting data. It also means the source and destination data will be byte-for-byte identical, which would not have been true if the data was being re-encrypted with a new EDEK.



### Warning:

When using `/.reserved/raw/` to `distcp` encrypted data, make sure you preserve extended attributes with the `-px` flag. This is because encrypted attributes such as the EDEK are exposed through extended attributes and must be preserved to be able to decrypt the file.

This means that if the `distcp` is initiated at or above the encryption zone root, it will automatically create a new encryption zone at the destination if it does not already exist. Hence, Cloudera recommends you first create identical encryption zones on the destination cluster to avoid any potential mishaps.

## 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 `-skipcrccheck`, `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.

```
sudo -u hdfs hadoop fs -rm -r -skipTrash /encryption_zone
```



**Important:** The Key Trustee KMS does not directly execute 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



**Warning:** 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. If you are using the Key Trustee KMS and Key Trustee Server, see “Backing up Key Trustee Server and Clients” for instructions on backing up Key Trustee Server and Key Trustee KMS.

## Rolling Encryption Keys

Before attempting to roll an encryption key (also known as an encryption zone key, or EZ key), familiarize yourself with the concepts described in “HDFS Transparent Encryption”, as the material in these sections presumes you are familiar with the fundamentals of HDFS transparent encryption and Cloudera data at rest encryption.

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

To roll an EZ key:

1. (Optional) 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.



- (Optional) 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:
7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTX10bXCpF}
```

Log off as HDFS Superuser.

- 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
oadBalancingKMSSClientProvider@5ea434c8
for keyName: key1
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSSClientProvider@5ea434c8
has been updated.
```

- (Optional) 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" : "<description>",
```

```

    "created" : <millis-epoch>,
    "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: KMSSClientProvider[https://<KMS_HOSTNAME>:16000/kms/v1/]

for key name: <key-name>

key1 has been successfully rolled.

KMSSClientProvider[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" : "<description>",
  "created" : <millis-epoch>,
  "versions" : <versions> (For example, version 2)
}

```

## Re-encrypting Encrypted Data Encryption Keys (EDEKs)

Before attempting to re-encrypt an EDEK, familiarize yourself with the concepts described in “HDFS Transparent Encryption”, as the material in this section presumes you are familiar with the fundamentals of HDFS transparent encryption and Cloudera data at rest encryption.

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 (see [Rolling Encryption Keys](#) on page 24). 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.

### Benefits and Capabilities

In addition to minimizing security risks, re-encrypting the EDEK offers the following 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

- 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

This section identifies 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 doesn't 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.

1. Navigate to the cluster in which you will be rolling keys and re-encrypting the EDEK.
2. Log in as HDFS Superuser.
3. (Optional) To 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.

- 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.LoadBalancingKMSSClientProvider@5ea434c8
for keyName: key1
key1 has been successfully rolled.
org.apache.hadoop.crypto.key.kms.LoadBalancingKMSSClientProvider@5ea434c8
has been updated.
```

- Log out as Key Administrator, and log in as HDFS Superuser.
- (Optional) 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, unknownValue=null}, edek: 9aa13ea4a700f96287cfe1349f6ff4f2,
iv: d129c913c8a34cde6371ec95edfb7337, keyName: key1, ezKeyVersionName:
7mbvopZ0Weuvs0XtTkpGw3G92KuWc4e4xcTX10bXCpF}
```

- 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 1Million 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
```

**Table 1: Re-encryption Status Column Data**

Column Name	Description	Sample Data
ZoneName	The encryption zone name	/ez
Status	<ul style="list-style-type: none"> <li>Submitted: the command is received, but not yet being processed by the NameNode.</li> <li>Processing: the zone is being processed by the NameNode.</li> <li>Completed: the NameNode has finished processing the zone, and every file in the zone has been re-encrypted.</li> </ul>	Completed

Column Name	Description	Sample Data
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.	ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRT0HS3YjCz
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
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. (Optional) 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:
ZMHfRoGKeXXgf0QzCX8q16NczIw2sq0rWRT0HS3YjCz }
```

## Managing Re-encryption Operations

This section includes information that can help you manage various facets of the EDEK re-encryption process.

### 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.

## Configuring the Key Management Server (KMS)

Hadoop Key Management Server (KMS) is a cryptographic key management server based on the Hadoop KeyProvider API. It provides a KeyProvider implementation client that interacts with the KMS using the HTTP REST API. Both the KMS and its client support HTTP SPNEGO Kerberos authentication and TLS/SSL-secured communication. The KMS is a Java-based web application that uses a preconfigured Jetty server bundled with the Hadoop distribution.

For instructions on securing the KMS, see “Securing the Key Management Server (KMS)”.

Cloudera provides the following implementations of the Hadoop KMS:

- **Java KeyStore KMS** - The default Hadoop KMS included in CDP that uses a file-based Java KeyStore (JKS) for its backing keystore. For parcel-based installations, no additional action is required to install or upgrade the KMS. Cloudera strongly recommends not using Java Keystore KMS in production environments.
- **Key Trustee KMS** - A custom KMS that uses Cloudera Navigator Key Trustee Server for its backing keystore instead of the file-based Java KeyStore (JKS) used by the default Hadoop KMS. Cloudera strongly recommends using Key Trustee KMS in production environments to improve the security, durability, and scalability of your cryptographic key management. For more information about the architecture and components involved in encrypting data at rest for production environments, see “Encrypting Data at Rest” and “Data at Rest Reference

Architecture”. Also, integrating Key Trustee Server with Cloudera Navigator Key HSM provides an additional layer of protection.

- Navigator KMS Services backed by Thales HSM - A custom KMS that uses a supported Thales Hardware Security Module (HSM) as its backing keystore. This KMS service provides the highest level of key isolation to customers who require it.
- Navigator KMS Services backed by Luna HSM - A custom KMS that uses a supported Luna Hardware Security Module (HSM) as its backing keystore. This KMS provides the highest level of key isolation to customers who require it.

### Related Information

[Securing the Key Management Server \(KMS\)](#)

[Encrypting Data at Rest](#)

[Data at Rest Encryption Reference Architecture](#)

## Configuring the KMS Using Cloudera Manager

If you are using Cloudera Manager, you can view and edit the KMS configuration by navigating to the following pages, depending on the KMS implementation you are using:

- Key Trustee KMS serviceConfiguration
- Java KeyStore KMS serviceConfiguration

For instructions about configuring the KMS and its clients using the command line for package-based installations, continue reading:

### Configuring the KMS Cache Using Cloudera Manager

By default, the KMS caches keys to reduce the number of interactions with the key provider. You can disable the cache by setting the `hadoop.kms.cache.enable` property to `false`.

The cache is only used with the `getCurrentKey()`, `getKeyVersion()` and `getMetadata()` methods.

For the `getCurrentKey()` method, entries are cached for a maximum of 30000 milliseconds to prevent stale keys.

For the `getKeyVersion()` method, entries are cached with a default inactivity timeout of 600000 milliseconds (10 minutes).

You can configure the cache and its timeout values by adding the following properties to KMS `serviceConfigurationAdvancedKey Management Server Proxy Advanced Configuration Snippet (Safety Valve)` for `kms-site.xml`:

```
<property>
  <name>hadoop.kms.cache.enable</name>
  <value>true</value>
</property>
<property>
  <name>hadoop.kms.cache.timeout.ms</name>
  <value>600000</value>
</property>

<property>
  <name>hadoop.kms.current.key.cache.timeout.ms</name>
  <value>30000</value>
</property>
```

### Configuring the Audit Log Aggregation Interval

Audit logs are generated for `GET_KEY_VERSION`, `GET_CURRENT_KEY`, `DECRYPT_EEK`, and `GENERATE_EEK` operations.

Entries are aggregated by user, key, and operation for a configurable interval, after which the number of aggregated operations by the user for a given key is written to the audit log.

The interval is configured in milliseconds by adding the `hadoop.kms.aggregation.delay.ms` property to KMS `serviceConfigurationAdvanced` Key Management Server Proxy Advanced Configuration Snippet (Safety Valve) for `kms-site.xml`:

```
<property>
  <name>hadoop.kms.aggregation.delay.ms</name>
  <value>10000</value>
</property>
```

## Securing the Key Management Server (KMS)

Cloudera provides the following implementations of the Hadoop KMS:

- **Java KeyStore KMS** - The default Hadoop KMS included in CDP that uses a file-based Java KeyStore (JKS) for its backing keystore. For parcel-based installations, no additional action is required to install or upgrade the KMS. Cloudera strongly recommends not using Java KeyStore KMS in production environments.
- **Key Trustee KMS** - A custom KMS that uses Cloudera Navigator Key Trustee Server for its backing keystore instead of the file-based Java KeyStore (JKS) used by the default Hadoop KMS. Cloudera strongly recommends using Key Trustee KMS in production environments to improve the security, durability, and scalability of your cryptographic key management. For more information about the architecture and components involved in encrypting data at rest for production environments, see “Encrypting Data at Rest” and “Data at Rest Reference Architecture”. Also, integrating Key Trustee Server with Cloudera Navigator Key HSM provides an additional layer of protection.
- **Navigator KMS Services backed by Thales HSM** - A custom KMS that uses a supported Thales Hardware Security Module (HSM) as its backing keystore. This KMS service provides the highest level of key isolation to customers who require it.
- **Navigator KMS Services backed by Luna HSM** - A custom KMS that uses a supported Luna Hardware Security Module (HSM) as its backing keystore. This KMS provides the highest level of key isolation to customers who require it.

This topic contains information on securing the KMS using Kerberos, TLS/SSL communication, and access control lists (ACLs) for operations on encryption keys. Cloudera Manager instructions can be performed for both Key Trustee KMS and Java KeyStore KMS deployments. Command-line instructions apply only to Java KeyStore KMS deployments. Key Trustee KMS is not supported outside of Cloudera Manager.

### Related Information

[Encrypting Data at Rest](#)

[Data at Rest Encryption Reference Architecture](#)

## Enabling Kerberos Authentication for the KMS

Minimum Required Role: [Full Administrator](#). This feature is not available when using Cloudera Manager to manage Data Hub clusters.

To enable Kerberos for the KMS using Cloudera Manager:


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 AdministrationSecurityKerberos 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


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

The steps for configuring and enabling Hadoop TLS/SSL for the KMS are as follows:

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.

**Table 2: KMS TLS/SSL Properties**

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.

## Configuring KMS Access Control Lists (ACLs)

An Access Control List (ACL) is a list of specific permissions or controls that allow individual users, groups, a host, or applications to perform specific actions upon specific objects. The Hadoop KMS supports a range of ACLs that control access to encryption keys and key operations on a granular basis.

KMS ACLs indirectly impact data access by controlling key access, and are decoupled from HDFS file permissions and ACLs. KMS ACLs alone do not directly control data access. Instead, KMS ACLs control whether or not an authorized client can perform a specific operation on a named encryption key.


While KMS ACLs play a primary role in controlling encryption key security, it is important to understand that they are not the only mechanism by which access is controlled. A user's role also factors into the level of access.

Proper configuration of KMS ACLs depends on a variety of variables such as workload, CDP components in use, and how your clusters are configured. This documentation does not take into consideration or describe these outside variables. For details about your specific component's ACL behavior and requirements, refer to the product documentation for the CDP components in your configuration.

### KMS ACLs and Roles

Cloudera's framework for key management is based on enforcing a secure-by-default configuration based upon the KMS ACLs and roles described here.

**Table 3: KMS ACLs and Roles**

Role	Description	Allowed To:	Not Allowed To:
Key Administrators	The sole purpose of a Key Administrator is to create and manage keys. This user is whitelisted in a number of areas so that they can handle defined and undefined keys within the context of the KMS.	<ul style="list-style-type: none"> <li>Create and manage encryption zone keys</li> <li>Add, update, or otherwise modify ACLs that protect all encryption zone keys</li> </ul>	
HDFS Superusers	<p>Responsible for HDFS administration, HDFS Superusers are not granted rights to decrypt data within encryption zones. Rather, they are authorized to only create zones and attach keys to those zones for the data sets that they manage. HDFS Superusers are usually also HDFS Superusers.</p> <p> <b>Important:</b> To maintain the separation of duties, the HDFS Encryption Wizard automatically blacklists all of the specified HDFS Superuser Groups set in dfs.permissions.supergroup. Do not authorize HDFS Superusers to create, manage, or read keys, or to decrypt EEKs.</p>	<ul style="list-style-type: none"> <li>Create encryption zones</li> </ul>	<ul style="list-style-type: none"> <li>Manage, create, or read keys</li> <li>Add, update, or otherwise modify ACLs that protect all keys</li> <li>Decrypt EEKs</li> </ul>
HDFS Service User	There is only one HDFS Service User; this is the user the HDFS service runs as within the Hadoop framework. HDFS Service Users are granted special permissions to generate keys (EEKs) that populate per encryption zone key caches.	<ul style="list-style-type: none"> <li>Generate keys that are made available for use through the per encryption zone key caches</li> </ul>	
End Users	Producers and consumers of data who store or retrieve information within specific encryption zones.		<ul style="list-style-type: none"> <li>Read and write in encrypted data spaces</li> </ul>

### KMS ACL Classes

ACLs are implemented in the upstream Hadoop KMS and apply to any variant of the service, whether it is through Key Trustee Server (KTS) or the Java Keystore (JKS). Understanding how these ACLs are defined and work enables you to more accurately create ACLs that meet both your needs and the needs of your customers.

The KMS ACL classes provide fine-grain control of key access at different decision points (for example, preventing a user from accessing the KMS altogether), and are applied either KMS-wide or key-specific:

- KMS-wide

You can use KMS-wide ACLs to specify the types of operations a user can perform. Configure KMS-wide operations using the classes:

- `hadoop.kms.acl.<OPERATION>`
- `hadoop.kms.blacklist.<OPERATION>`

KMS-wide ACLs precede key-specific ACLs. In other words, the permission check on a key-specific ACL only occurs if permission is already granted at the KMS-wide level; if it is, then the permission check against key-specific ACLs can proceed.

Users accessing a KMS-wide ACL are first checked for inclusion in the ACL for the requested operation, and then checked for exclusion in the denylist before the operation is performed or access is granted.

- Key-specific

You can use key-specific ACLs, set on a per-encryption zone key basis, to specify the types of operations a user can perform. Configure key-specific operations using the classes:

- `default.key.acl.<OPERATION>`
- `whitelist.key.acl.<OPERATION>`
- `key.acl.<key_name>.<OPERATION>`



**Important:**

The KMS supports both allowlist and denylist ACLs. Denylist entries apply and are implemented KMS-wide, while allowlist entries apply and are implemented at the key-specific level.



There are five distinct classes of ACLs, each defined by the operations defined within.

### The `hadoop.kms.acl.<OPERATION>` Class

This class controls permissions to perform KMS-wide operations. The only ACLs that fall back to a value of '\*' when the values are empty are those within the `hadoop.kms.acl` definition.

You can define fine-grain access for the following operations in this ACL class:

**Table 4: `hadoop.kms.acl.<OPERATION>` Class**

<code>hadoop.kms.acl.&lt;OPERATION&gt;</code>	<code>&lt;OPERATION&gt;</code> Name	Description
CREATE	Create Key	Creates an encryption zone key.  <b>Note:</b> Encryption zone key material is returned only if the user also has GET permission.
DELETE	Delete Key	Deletes an encryption zone key.
ROLLOVER	Rollover Key	Rolls an encryption zone key to the new version, using different material. The encrypted encryption keys (also known as EEKs or EDEKs) are always generated using the latest version of the encryption key.  <b>Note:</b> Encryption zone key material is returned only if the user also has GET permission.
	Invalidate Key Cache	Invalidates all cached EEKs.
GET	Get Current Key	Gets the latest version of the encryption zone key, including its material.
	Get Key Version	Gets a specified version of the encryption zone key, including its material.
	Get Key Versions	Gets all versions of the encryption zone key, including their materials.
GET_KEYS	Get Key Names	Lists all the encryption zone key names on the Key Trustee KMS. No additional key material is returned.



hadoop.kms.acl.<OPERATION>	<OPERATION> Name	Description
GET_METADATA	Get Key Metadata	Gets the metadata of an encryption zone key, including the cipher, key length, description, time created, and number of versions and other attributes.
	Get Keys Metadata	Gets the metadata for a collection of encryption zone keys.
SET_KEY_MATERIAL		For ROLLOVER and/or CREATE encryption key operations, determines whether or not the user can provide key material.  If encryption zone key material is not provided, then the Key Trustee KMS randomly generates the material.
GENERATE_EEK	Generate Encrypted Key for Current KeyVersion	Generates an encrypted encryption zone key from a given encrypted key name, using its current key version.
	Re-encrypt Encrypted Key With The Latest KeyVersion	Takes a previously generated EEK, and re-encrypts it using the latest KeyVersion of the same encrypted key. If the latest KeyVersion is the same as the one used to generate the EEK, then the same EEK is returned.
	Batch Re-encrypt Encrypted Keys With The Latest KeyVersion	Takes a batch of previously generated EEKs, and re-encrypts them using the latest KeyVersion of the same encrypted key. If the latest KeyVersion is the same as the one used to generate the EEK, then the same EEK is returned.
DECRYPT_EEK	Decrypt Encrypted Key	Decrypts an EEK and returns the decrypted encryption zone key.

### The `hadoop.kms.blacklist.<OPERATION>` Class

This class controls permissions to perform KMS-wide operations. Users and groups listed here will be prevented from performing any other listed OPERATION on any encryption keys.

You can define fine-grain access for the following operations in this ACL category:

**Table 5: `hadoop.kms.blacklist.<OPERATION>` Class**

hadoop.kms.blacklist.<OPERATION>	<OPERATION> Name	Description
CREATE	Create Key	Creates an encryption zone key.   <b>Note:</b> Encryption zone key material is returned only if the user also has GET permission.
DELETE	Delete Key	Deletes an encryption zone key.
ROLLOVER	Rollover Key	Rolls an encryption zone key to the new version, using different material.  The encrypted encryption keys (EEKs) are always generated using the latest version of the encryption key.   <b>Note:</b> Encryption zone key material is returned only if the user also has GET permission.
	Invalidate Key Cache	Invalidates all cached EEKs.
GET	Get Current Key	Gets the latest version of the encryption zone key, including its material.
	Get Key Version	Gets a specified version of the encryption zone key, including its material.
	Get Key Versions	Gets all versions of the encryption zone key, including their materials.
GET_KEYS	Get Key Names	Lists all the encryption zone key names on the Key Trustee KMS. No additional key material is returned.
GET_METADATA	Get Key Metadata	Gets the metadata of an encryption zone key, including the cipher, key length, description, time created, and number of versions and other attributes.
	Get Keys Metadata	Gets the metadata for a collection of encryption zone keys.

hadoop.kms.blacklist.<OPERATION>	<OPERATION> Name	Description
SET_KEY_MATERIAL		For ROLLOVER and/or CREATE encryption key operations, determines whether or not the user can provide key material.  If encryption zone key material is not provided, then the Key Trustee KMS randomly generates the material.
GENERATE_EEK	Generate Encrypted Key for Current KeyVersion	Generates an encrypted encryption zone key (EEK) from a given encrypted key name, using its current key version.
	Re-encrypt Encrypted Key With The Latest KeyVersion	Takes a previously generated EEK, and re-encrypts it using the latest KeyVersion of the same encrypted key. If the latest KeyVersion is the same as the one used to generate the EEK, then the same EEK is returned.
	Batch Re-encrypt Encrypted Keys With The Latest KeyVersion	Takes a batch of previously generated EEKs, and re-encrypts them using the latest KeyVersion of the same encrypted key. If the latest KeyVersion is the same as the one used to generate the EEK, then the same EEK is returned.
DECRYPT_EEK	Decrypt Encrypted Key	Decrypts an EEK and returns the decrypted encryption zone key.

### The key.acl.<key-name>.<OPERATION> Class

This class controls permissions to perform operations for a specific key, and applies to key-specific ACLs.



#### Important:

In the process of evaluating key access, key.acl.<OPERATION> overrides default operations provided in default.key.acl.<OPERATION> or whitelist.key.acl.<OPERATION>.

Be aware that the key.acl does not fall back to a value of '\*' when you use empty values.

You can define fine-grain access for the following operations in this ACL class:

**Table 6: key.acl<key-name>.<OPERATION> Class**

key.acl.key-name. <OPERATION>	<OPERATION> Name	Description
MANAGEMENT	createKey, deleteKey, rolloverNewVersion	Use for the following operations: <ul style="list-style-type: none"> <li>createKey</li> <li>deleteKey</li> <li>rolloverNewKeyVersion</li> </ul>
GENERATE_EEK	Generate Encryption Key	Use for the following operations: <ul style="list-style-type: none"> <li>generateEncryptedKey</li> <li>reencryptEncryptedKey</li> <li>reencryptEncryptedKeys</li> <li>warmUpEncryptedKeys</li> </ul>
DECRYPT_EEK	Decrypt Encrypted Key	Use for the decryptEncryptedKey operation.
READ	Read	Use for the following operations: <ul style="list-style-type: none"> <li>getKeyVersion</li> <li>getKeyVersions</li> <li>getMetadata</li> <li>getKeysMetadata</li> <li>getCurrentKey</li> </ul>
ALL	All	Use to specify all operations in this class.

### The default.key.acl.<OPERATION> Class

This class controls permission to perform operations for keys that are not otherwise specified by key.acl.<key-name>.<OPERATION>, and applies to key-specific ACLs.

The `default.key.acl.<OPERATION>` applies to all keys for which an ACL has not been explicitly configured. Be aware that if all of the following conditions exist, key access is denied:

- There is no key-specific ACL configured
- There is no KMS ACL configured for the requested operation
- There is no whitelist key ACL configured for the requested operation

Also note that the `default.key.acl` does not fall back to a value of '\*' when you use empty values. All `default.key.acls` are (by default) empty, which means that you must create the required key definition entries for each key you wish to use.



**Note:** The `default.key.acl` class does not support the ALL operation. If specified, it will be ignored.

**Table 7: `default.key.acl.<OPERATION>` Class**

<code>default.key.acl. &lt;OPERATION&gt;</code>	<code>&lt;OPERATION&gt;</code> Name	Description
MANAGEMENT	Manage the Zone Key	Use for the following operations: <ul style="list-style-type: none"> <li>• <code>createKey</code></li> <li>• <code>deleteKey</code></li> <li>• <code>rolloverNewKeyVersion</code></li> </ul>
GENERATE_EEK	Create Encryption Key	Use for the <code>decryptEncryptedKey</code> operation.
DECRYPT_EEK	Decrypt Encryption Key	Use for the following operations: <ul style="list-style-type: none"> <li>• <code>getKeyVersion</code></li> <li>• <code>getKeyVersions</code></li> <li>• <code>getMetadata</code></li> <li>• <code>getKeysMetadata</code></li> <li>• <code>getCurrentKey</code></li> </ul>
READ	Read	Use for the following operations: <ul style="list-style-type: none"> <li>• <code>getKeyVersion</code></li> <li>• <code>getKeyVersions</code></li> <li>• <code>getMetadata</code></li> <li>• <code>getKeysMetadata</code></li> <li>• <code>getCurrentKey</code></li> </ul>

### The `whitelist.key.acl` Class

This class controls permissions to perform key operations across all keys, and applies to key-specific ACLs.

**Table 8: `whitelist.key.acl.<OPERATION>` Class**

<code>whitelist.key.acl.&lt;OPERATION&gt;</code>	<code>&lt;OPERATION&gt;</code> Name	Description
MANAGEMENT	Manage the Zone Key	Use for the following operations: <ul style="list-style-type: none"> <li>• <code>createKey</code></li> <li>• <code>deleteKey</code></li> <li>• <code>rolloverNewKeyVersion</code></li> </ul>
GENERATE_EEK	Create Encryption Key	Use for the <code>decryptEncryptedKey</code> operation.

whitelist.key.acl.<OPERATION>	<OPERATION> Name	Description
DECRYPT_EEK	Decrypt Encryption Key	Use for the following operations: <ul style="list-style-type: none"> <li>• getKeyVersion</li> <li>• getKeyVersions</li> <li>• getMetadata</li> <li>• getKeysMetadata</li> <li>• getCurrentKey</li> </ul>
READ	Read	Use for the following operations: <ul style="list-style-type: none"> <li>• getKeyVersion</li> <li>• getKeyVersions</li> <li>• getMetadata</li> <li>• getKeysMetadata</li> <li>• getCurrentKey</li> </ul>

**Important:**

To successfully create an encryption zone, the HDFS Superuser must include the following two entries in the whitelist.key.acl.<OPERATION>:

- GENERATE\_EEK
- READ

These whitelist entries enable the NameNode to build per encryption zone key EEK caches when an encryption zone is created.

**KMS ACL Evaluation Flow**

Before diving into the details of how KMS ACLs are evaluated, it is critical that you understand the key rules that the KMS 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.

The better you understand these rules, the more likely it is that you will be successful creating and managing KMS ACL sets to achieve your desired goals.



The KMS evaluates `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 disallowed or otherwise not allowed 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.



The KMS evaluates `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.



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

The KMS evaluates whether or not there is a `key.acl.KEY.<OPERATION>` 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 evaluating the `default.key.acl.<OPERATION>` and `key.acl.<OPERATION>`, 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` would override `default.key.acl.READ` for `prodkey`, so it will be configured with its own READ ACLs).



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

### KMS ACL Syntax and Tips

#### Blacklist and Whitelist Syntax

The ACL syntax for both allowed and disallowed entries is as follows:

- Users only: `user1,user2,userN`

There are no spaces following the commas separating the users in the list.

- Groups only: `nobody group1,group2,groupN`

There is a space between `nobody` and the comma-separated group list. The `nobody` user, if it exists, must not have privileges to log in to or interact with the system. If you are uncertain about its access privileges, specify a different nonexistent user in its place.

- Users and Groups: `user1,user2,userN group1,group2,groupN`

The comma-separated user list is separated from the comma-separated group list by a space.

#### Blocking User Access

If you wish to block access to an operation entirely, use the value of an empty space, or some non-existent values (for example, `'NOUSERS NOGROUPS'`). By doing this, you ensure that no user maps to a particular operation by default. Alternatively, you can restrict those features to Key Administrators only by setting the value to `Keyadmin` users and/or groups.

#### Group Membership in KMS ACLs

The group membership used by ACL entries depends upon the configured group mapping mechanism for HDFS. By default, group membership is determined on the local Linux system running the KMS service. If you have configured HDFS to use LDAP for group mapping, then group membership for the ACL entries is determined using the configured LDAP settings.


### Configuring KMS ACLs Using Cloudera Manager

Minimum Required Role: [Key Administrator](#) (also provided by Full Administrator)

The KMS installation wizard includes an option to generate the recommended ACLs. To view or edit the ACLs:

1. Go to the KMS service.



2. Click Configuration.
3. In the Search field, type acl to show the Key Management Server Advanced Configuration Snippet (Safety Valve) for kms-acls.xml (in the Key Management Server Default Group category).
4. Add or edit the ACL properties according to your cluster configuration. See "Recommended KMS Access Control List" for example ACL entries.
5. Click Save Changes.
6. Return to the Home page by clicking the Cloudera Manager logo, and return to the KMS service.
7. Click  (Refresh Needed) . Then click Refresh Cluster.
8. Click Finish.

### Recommended KMS ACL

Cloudera recommends the following ACL definition for secure production settings. Replace keyadmin and keyadmin group with the user and group responsible for maintaining encryption keys.



**Note:** If you are entering the ACL using Cloudera Manager, omit the surrounding <configuration> and </configuration> tags; Cloudera Manager adds this automatically.

```
<configuration>
<!--
KMS ACLs control which users can perform various actions on the KMS, and which
users and groups have access to which keys.
This file includes the following sections:
  * ACLs for KMS operations
  ** Access to specific KMS operations
  ** Blacklists for specific operations
  * ACLs for keys
  ** Default ACLs for keys
  ** Whitelist ACLs for keys
  ** Key-specific ACLs
-->
<!--
KMS ACLs that govern access to specific key operations. If access is not granted
for an operation here, then the operation is forbidden, even if a key ACL
allows it.
The ACL value should be either a username or a username and groupname separated
by whitespace.
A value of "*" (for the username or groupname) indicates that all users are
granted access to that operation. Any operation for which there is no ACL or an
empty (zero-length) ACL is treated as having an ACL with a value of "*".
To disallow all users, add an ACL with a value of " " (a single space).

Note: This convention applies only to the KMS-wide ACLs beginning with 'hadoop.kms.acl'.
-->
<property>
  <name>hadoop.kms.acl.CREATE</name>
  <value>keyadmin keyadmin group</value>
  <description>
    ACL for create-key operations.
    If the user is not in the GET ACL, the key material is not returned
    as part of the response.
  </description>
</property>

<property>
  <name>hadoop.kms.acl.DELETE</name>
```

```

    <value>keyadmin keyadmingroup</value>
    <description>
      ACL for delete-key operations.
    </description>
  </property>

  <property>
    <name>hadoop.kms.acl.ROLLOVER</name>
    <value>keyadmin keyadmingroup</value>
    <description>
      ACL for rollover-key operations.
      If the user does is not in the GET ACL, the key material is not returned
      as part of the response.
    </description>
  </property>
  <property>
    <name>hadoop.kms.acl.GET</name>
    <value></value>
    <description>
      ACL for get-key-version and get-current-key operations.
    </description>
  </property>

  <property>
    <name>hadoop.kms.acl.GET_KEYS</name>
    <value>keyadmin keyadmingroup</value>
    <description>
      ACL for get-keys operations.
    </description>
  </property>

  <property>
    <name>hadoop.kms.acl.SET_KEY_MATERIAL</name>
    <value></value>
    <description>
      Complementary ACL for CREATE and ROLLOVER operations to allow the cli
      ent to provide the
      key material when creating or rolling a key.
    </description>
  </property>

  <property>
    <name>hadoop.kms.acl.GENERATE_EEK</name>
    <value>hdfs supergroup</value>
    <description>
      ACL for generateEncryptedKey CryptoExtension operations.
    </description>
  </property>

  <!--
    KMS blacklists to prevent access to operations. These settings override t
    he permissions granted by the KMS ACLs listed above.

    The blacklist value should be either a username or a username and groupn
    ame separated by whitespace.

    A blank value indicates that no user is blacklisted from the operation. A
    value of "*" (for either the username or
    groupname) indicates that all users are blacklisted from the operation.
    Any operation for which there is no blacklist
    will be treated as having a blacklist with an empty value.
  -->
  <!--

```

```

In this template the hdfs user is blacklisted for everything except GET_
METADATA, GET_KEYS, and GENERATE_EEK. The
GET and SET_KEY_MATERIAL operations are blacklisted for all users because
Hadoop users should not need to perform
those operations, and access to the key material should be as restricted
as possible.
-->

<property>
  <name>hadoop.kms.blacklist.CREATE</name>
  <value>hdfs supergroup</value>
</property>

<property>
  <name>hadoop.kms.blacklist.DELETE</name>
  <value>hdfs supergroup</value>
</property>

<property>
  <name>hadoop.kms.blacklist.ROLLOVER</name>
  <value>hdfs supergroup</value>
</property>
<property>
  <name>hadoop.kms.blacklist.GET</name>
  <value>*</value>
</property>

<property>
  <name>hadoop.kms.blacklist.GET_KEYS</name>
  <value></value>
</property>

<property>
  <name>hadoop.kms.blacklist.SET_KEY_MATERIAL</name>
  <value>*</value>
</property>

<property>
  <name>hadoop.kms.blacklist.DECRYPT_EEK</name>
  <value>hdfs supergroup</value>
</property>

<property>
  <name>keytrustee.kms.acl.UNDELETE</name>
  <value></value>
  <description>
    ACL that grants access to the UNDELETE operation on all keys.
    Only used by Key Trustee KMS.
  </description>
</property>

<property>
  <name>keytrustee.kms.acl.PURGE</name>
  <value></value>
  <description>
    ACL that grants access to the PURGE operation on all keys.
    Only used by Key Trustee KMS.
  </description>
</property>

<!--
Default key ACLs that govern access to key operations for key-operation
pairs that do not have a
key-specific ACL already. Key-specific ACLs override the default key ACLs.

```

The ACL value should be either a username or a username and group name separated by whitespace.

An empty value for an ACL indicates that no user is granted access to that operation. A value of "\*" (for the username or groupname) indicates that all users are granted access to that operation.

Any operation for which there is no ACL will be treated as having an ACL with an empty value.

-->

```
<property>
  <name>default.key.acl.MANAGEMENT</name>
  <value></value>
  <description>
    Default ACL that grants access to the MANAGEMENT operation on all keys.
  </description>
</property>
<property>
  <name>default.key.acl.GENERATE_EEK</name>
  <value></value>
  <description>
    Default ACL that grants access to the GENERATE_EEK operation on all keys.
  </description>
</property>
<property>
  <name>default.key.acl.DECRYPT_EEK</name>
  <value></value>
  <description>
    Default ACL that grants access to the DECRYPT_EEK operation on all keys.
  </description>
</property>
<property>
  <name>default.key.acl.READ</name>
  <value></value>
  <description>
    Default ACL that grants access to the READ operation on all keys.
  </description>
</property>
```

<!--

Whitelist key ACLs that grant access to key-specific operations. Any permissions granted here will be added to whatever permissions are granted by the specific key ACL or the default key ACL.

Note that these whitelist ACLs grant access to operations on specific keys. If the operations are not allowed because of the KMS ACLs/blacklists, then they will not be permitted, regardless of the whitelist settings.

The ACL value should be either a username or a username and group name separated by whitespace.

An empty value for an ACL indicates that no user is granted access to that operation. A value of "\*" (for the username or groupname) indicates that all users are granted access to that operation. Any operation for which there is no ACL will be treated as having an ACL with an empty value.

-->

```

<property>
  <name>whitelist.key.acl.MANAGEMENT</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    Whitelist ACL for MANAGEMENT operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.READ</name>
  <value>hdfs supergroup</value>
  <description>
    Whitelist ACL for READ operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.GENERATE_EEK</name>
  <value>hdfs supergroup</value>
  <description>
    Whitelist ACL for GENERATE_EEK operations for all keys.
  </description>
</property>

<property>
  <name>whitelist.key.acl.DECRYPT_EEK</name>
  <value>keyadmin keyadmingroup</value>
  <description>
    Whitelist ACL for DECRYPT_EEK operations for all keys.
  </description>
</property>
<!--
  Key ACLs that grant access to specific key operations. Any permissions g
ranted here are added
  to whatever permissions are granted by the whitelists.
  The key ACL name should be key.acl.<keyname>.<OPERATION>.

  The ACL value should be either a username or a username and group name sep
arated by whitespace.

  An empty value for an ACL indicates that no user is granted access to t
hat operation. A value
  of "*" (for the username or group name) indicates that all users are gr
anted access to that operation.
  Any key operation for which there is no ACL will default to the default AC
L for the operation.

  Normally adding users or groups for a specific key and DECRYPT_EEK is su
fficient to allow access
  to data protected with HDFS data at rest encryption.
-->
<!--
  The following ACLs are required for proper functioning of services. Clou
dera Manager does not create keys or
  encryption zones; however, our best practices recommend encryption zones
on certain directories.
  An assumption in these ACLs is that the user has followed the recommended
naming scheme and named the keys
  according to documented best practices: "hive-key" for the Hive service,
  "hbase-key" for the Hbase service, etc. If the key names are different, th
en none of this will work
  out of the box, and you will need to edit these ACLs to match your key n
ames.
-->

```

```

<property>
  <name>key.acl.hive-key.DECRYPT_EEK</name>
  <value>hive hive</value>
  <description>
    Gives the Hive user and the Hive group access to the key named "hive-key".
    This allows the Hive service to read and write files in /user/hive/.
    Also note that the Impala user ought to be a member of the Hive group to
    enjoy this same access.
  </description>
</property>

<property>
  <name>key.acl.hive-key.READ</name>
  <value>hive hive</value>
  <description>
    Required because Hive compares key strengths when joining tables.
  </description>
</property>

<property>
  <name>key.acl.hbase-key.DECRYPT_EEK</name>
  <value>hbase hbase</value>
  <description>
    Gives the hbase user and hbase group access to the key named "hbase-key".
    This allows the hbase service to read and write files in /hbase.
  </description>
</property>

<property>
  <name>key.acl.solr-key.DECRYPT_EEK</name>
  <value>solr solr</value>
  <description>
    Gives the solr user and solr group access to the key named "solr-key".
    This allows the solr service to read and write files in /solr.
  </description>
</property>

<property>
  <name>key.acl.mapred-key.DECRYPT_EEK</name>
  <value>mapred,yarn hadoop</value>
  <description>
    Gives the mapred user and mapred group access to the key named "mapred-key".
    This allows mapreduce to read and write files in /user/history.
    This is required by YARN.
  </description>
</property>

<property>
  <name>key.acl.hue-key.DECRYPT_EEK</name>
  <value>oozie,hue oozie,hue</value>
  <description>
    Gives the appropriate users and groups access to the key named "hue-key".
    This allows Hue and Oozie to read and write files in /user/hue.
    Oozie is required here because it will attempt to access workflows in /
    user/hue/oozie/workspaces.
  </description>
</property>

```

```

<!-- This example is required if there are encryption zones on user's home d
irectories. -->
<!--
<property>
  <name>key.acl.username-key.DECRYPT_EEK</name>
  <value>username username,hive,hbase,solr,oozie,hue,yarn</value>
  <description>
    Designed to be placed on a key that protects the EZ /user/username, and
    assumes that
    the key name is also "username-key"; this shows that a number of serv
    ices can reach in
    to access data. Remove those that are not needed for your use case.
  </description>
</property>
-->

</configuration>

```

## 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. If you are using the Java KeyStore KMS service, and want to use Key Trustee Server as the backing key store for “HDFS Transparent Encryption”, use the following procedure.

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

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.name=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.name=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

[HDFS Transparent Encryption](#)

## Migrating a Key Trustee KMS Server Role Instance to a New Host



**Note:** This migration procedure applies only to the Key Trustee KMS Server, and does not apply to the Key Trustee Server. Do not attempt this migration on the Key Trustee Server.

In some cases—for example, after upgrading your servers—it is desirable to migrate a Key Trustee KMS Server role instance to a new host. This procedure describes how to move a Key Trustee KMS proxy service role instance from an existing cluster host to another cluster host. The security and performance requirements for the KMS proxy are based on providing a dedicated system to the role, and not shared with CDP or other services. The KMS proxy represents a service that must be:

- secure
- isolated from non-administrator access
- maintained as a system with a higher level of isolation and security requirements compared to other cluster nodes

### Related Information

[Resource Planning for Data at Rest Encryption](#)

[Managing Encryption Keys and Zones](#)

## Assumptions and Requirements



**Warning:** Do not attempt this migration procedure on any Key Trustee KMS version lower than 5.16.0.

The following assumptions and requirements apply during the migration of a Key Trustee KMS server role instance to a new host as described in this procedure:

- Complete the steps one node at a time (migrate to the first new node, verify, then repeat the steps to migrate to second new node, verify, and so on).
- The sequence of restarts indicated throughout the steps are critical to successfully completing the migration without data loss. Do not skip any of the steps.



- As required for any KMS service that is configured for HA, Zookeeper must be deployed as a service (true by default). Refer to “Adding a Service” for details about how to add services.
- Review and examine TLS and Kerberos configuration requirements: the new KMS nodes must be ready with a Java Keystore and Truststore that present the correct host certificates while also trusting the Key Trustee Server. If the custom Kerberos keytab retrieval script is in use for Kerberos integration, it is important to have those keytabs ready and ingested before proceeding. Refer to “Using a custom Kerberos keytab retrieval script” for details.
- For this use case/procedure, assume that the existing KMS proxy host instances are named:
  - ktkms01.example.com
  - ktkms02.example.com
- Assume that the host destination instances are:
  - ktkms03.example.com
  - ktkms04.example.com

## Migrating a Key Trustee KMS Server Role Instance to a New Host

1. Back up the Key Trustee KMS private key and configuration directory. See “Back up Key Trustee Server clients” for more information.
2. Before adding the new role instance, see “Resource Planning for Data at Rest Encryption” for considerations when selecting a host.
3. Run the Add Role Instances (“Role Instances”) wizard for the Key Trustee KMS service ( Key Trustee KMS service Actions Add Role Instances ).
4. Click Select hosts and select the checkbox for the host to which you are adding the new Key Management Server proxy service role instance. Click OK and then Continue.
5. On the Review Changes page of the wizard, confirm the authorization code, organization name, and settings, and then click Finish.
6. Select and start the new KMS instance (Actions for SelectedStart).



### Important:

The initial startup of the KMS instance may fail with the following error message:

```
java.io.IOException: Unable to verify private key match between KMS
hosts. If the system has been recently upgraded, DO NOT TAKE FURTHER
ACTION and contact your support representative as soon as possible.
If this is a new installation, verify private key files have been s
ynced between all KMS hosts. Aborting to prevent data inconsistency.
```

If this occurs, it indicates that the KMS attempted to verify the Key Trustee private key has been synchronized with the new instance, but was unable to because that synchronization has not yet taken place. This is expected behavior at this point in the process. Proceed to the next step, and the new KMS instance will come up when the KMS service is restarted after the synchronization.

7. Verify that a Kerberos HTTP principal has been created for that specific host role instance in the Security configuration interface ( Administration Security Kerberos Credentials ).

For example, in this scenario the existing KMS Kerberos principal is HTTP/ktkms01.example.com@EXAMPLE.COM, and you must verify the new host principal HTTP/ktkms03.example.com@EXAMPLE.COM has been created before continuing. If you cannot verify the new host principal, then click Generate Missing Credentials on the Kerberos Credentials tab to generate it before continuing.

If the custom Kerberos keytab retrieval script is in use for Kerberos integration, it is important to have those keytabs ready and ingested before proceeding. Refer to “Using a custom Kerberos keytab retrieval script” for details.

8. Synchronize the Key Trustee KMS private key. In this use case you log into the original working KMS instance host, which is ktkms01.example.com, and synchronize the keys from your existing KMS host ktkms01.example.

com to the new host to which you are migrating, ktkms03.example.com. Copy the private key over the network by running the following command as a privileged (root) user on the original Key Trustee KMS host:

```
rsync -zav /var/lib/kms-keytrustee/keytrustee/.keytrustee root@ktkms03.e
xample.com:/var/lib/kms-keytrustee/keytrustee/.
```

Replace the host name (here we are using ktkms03.example.com) with the hostname of the Key Trustee KMS host to which you are migrating.



**Warning:** It is very important that you perform this step. If the KMS private key is not copied from the existing instance to the new instance, then the encryption keys will be inaccessible when the old KMS instance is removed from the cluster.

9. To verify that the Key Trustee KMS private keys successfully synchronized, compare the MD5 hash of the private keys. On each Key Trustee KMS host, run the following command:

```
$ md5sum /var/lib/kms-keytrustee/keytrustee/.keytrustee/secring.gpg
```

If the outputs are different, contact Cloudera Support for assistance. Do not attempt to synchronize existing keys. If you overwrite the private key and do not have a backup, any keys encrypted by that private key are permanently inaccessible, and any data encrypted by those keys is permanently irretrievable. If you are configuring Key Trustee KMS high availability for the first time, continue synchronizing the private keys.

10. Restart the Key Trustee KMS service (Key Trustee KMS serviceActionsRestart). After the restart completes, click Close.
11. Restart the cluster (“Starting, Stopping, Refreshing, and Restarting a Cluster”). This refreshes all the KMS instances in the cluster, and ensures all stale services are restarted.
12. Redeploy the client configuration (HomeClusterDeploy Client Configuration).
13. Run the steps in “Managing Encryption Keys and Zones”. Perform the check multiple times to exercise the Load balanced KMS nodes properly. If a command fails during this test, stop immediately, halt the cluster, and contact Cloudera Support.
14. Remove the old KMS instance being migrated. First stop the KMS instance (ktkms01.example.com) (Select KMS instanceActions for SelectedStop), and then delete it (Select KMS instanceActions for SelectedDelete).  
  
Delete the /var/lib/kms-keytrustee directories on the old KMS instance only after configuring and verifying the new KMS instances. Delete this material only after completing the following tasks:
  - a. Remove the old KMS instances from the KMS service.
  - b. Verify that you can read previously encrypted data only using the new KMS instances.
15. Restart the cluster (“Starting, Stopping, Refreshing, and Restarting a Cluster”). This refreshes all the KMS instances in the cluster, and ensures all stale services are restarted.
16. Redeploy the client configuration (HomeClusterwideDeploy Client Configuration).
17. Re-run the steps in “Managing Encryption Keys and Zones”. Perform the check multiple times to exercise the Load balanced KMS nodes properly. If a command fails during this test, stop immediately, and halt the cluster.
18. Repeat these steps for any additional KMS node migrations you wish to perform. So in the use case shown here, we would repeat the steps to migrate the ktkms02.example.com host to the ktkms04.example.com host.

## Configuring CDP Services for HDFS Encryption

This page contains recommendations for setting up HDFS Transparent Encryption with various CDP services.



**Important:** HDFS encryption does not support file transfer (reading, writing files) between zones through WebHDFS. For web-based file transfer between encryption zones managed by HDFS, use HttpFS with a load balancer (“Add the HttpFS role”) instead.



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

## Related Information

[HDFS Transparent Encryption](#)

[Configuring KMS Access Control Lists \(ACLs\)](#)

[HiveServer2 Security Configuration](#)

## HBase

### Recommendations

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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (ACLs)”), 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>
```

## 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: /user/hive

To use the auto-generated KMS ACLs (“Configuring KMS Access Control Lists (ACLs)”), 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 /user/hive to /user/hive-old, create an encryption zone at /user/hive, and then distcp all the data from /user/hive-old to /user/hive.

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

For more information, see [Changed Behavior after HDFS Encryption is Enabled](#) on page 52.

## 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

- 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 * FROM <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 ACLs (“Configuring KMS Access Control Lists (ACLs)”) 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 Security Configuration”, 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>
```

## Hue

### Recommendations

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 (“Configuring KMS Access Control Lists (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 (“Configuring KMS Access Control Lists (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>
```

## Impala

### Recommendations

- If HDFS encryption is enabled, configure Impala to encrypt data spilled to local disk.
- In releases lower than Impala 2.2.0 / CDH 5.4.0, Impala does not support the LOAD DATA statement when the source and destination are in different encryption zones. If you are running an affected release and need to use LOAD DATA with HDFS encryption enabled, copy the data to the table's encryption zone prior to running the statement.
- Use Cloudera Navigator to lock down the local directory where Impala UDFs are copied during execution. By default, Impala copies UDFs into /tmp, and you can configure this location through the --local\_library\_dir startup flag for the impalad daemon.
- 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 / CDH 5.5 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](#) on page 53.

## MapReduce and YARN

### MapReduce v1

#### Recommendations

MRv1 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.

### MapReduce v2 (YARN)

#### Recommendations

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 (“Configuring KMS Access Control Lists (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`.

## KMS ACL Configuration for MapReduce

In the KMS ACLs (“Configuring KMS Access Control Lists (ACLs)”), grant `DECRYPT_EEK` permission for the MapReduce key to the `mapred` and `yarn` users and the `hadoop` group:

```
<property>
  <name>key.acl.mapred-key.DECRYPT_EEK</name>
  <value>mapred,yarn hadoop</value>
  </description>
</property>
```

## Search

#### Recommendations

Make `/solr` an encryption zone. When you create the encryption zone, name the key `solr-key` to take advantage of auto-generated KMS ACLs (“Configuring KMS Access Control Lists (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 ACLs (“Configuring KMS Access Control Lists (ACLs)”), 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>
```

## 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.
- Spark does not encrypt shuffle data. To do so, configure the Spark local directory, spark.local.dir (in Standalone mode), to reside on an encrypted disk. For YARN mode, make the corresponding YARN configuration changes.

## KMS ACL Configuration for Spark

In the KMS ACLs (“Configuring KMS Access Control Lists (ACLs)”), 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>
```

## 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 more details, see [Hive](#) on page 51.
- 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.