

Cloudera Manager 7.1.2

Cloudera Navigator Encrypt

Date published: 2020-05-28

Date modified: 2020-07-10

CLOUDERA

<https://docs.cloudera.com/>

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

Cloudera Navigator Encrypt Overview.....	5
Process-Based Access Control List.....	5
Encryption Key Storage and Management.....	7
Registering Cloudera Navigator Encrypt with Key Trustee Server.....	8
Prerequisites.....	8
Registering with Key Trustee Server.....	8
Preparing for Encryption Using Cloudera Navigator Encrypt.....	12
Navigator Encrypt Commands.....	12
Preparing for Encryption.....	12
Block-Level Encryption with dm-crypt.....	13
Block-Level Encryption with a Loop Device.....	14
Navigator Encrypt and Device UUIDs.....	15
Pass-through Mount Options for navencrypt-prepare.....	16
Encrypting and Decrypting Data Using Cloudera Navigator Encrypt.....	16
Before You Begin.....	17
Encrypting Data.....	17
Decrypting Data.....	18
Converting from Device Names to UUIDs for Encrypted Devices.....	19
Preparing for UUID Conversion.....	19
Converting a Single Device.....	20
Converting All Available Devices.....	20
Finalizing the Conversion.....	21
Rolling Back the UUID Conversion.....	22
Navigator Encrypt Access Control List.....	22
Managing the Access Control List.....	22
ACL Profile Rules.....	26
Maintaining Cloudera Navigator Encrypt.....	27
Manually Backing Up Navigator Encrypt.....	27
Validating Navigator Encrypt Configuration.....	28
Restoring Mount Encryption Keys (MEKs) and Control File.....	28
Access Modes.....	28
Changing and Verifying the Master Key.....	29
Listing Categories.....	30
Updating ACL Fingerprints.....	30
Managing Mount Points.....	30
Navigator Encrypt Kernel Module Setup.....	31
Navigator Encrypt Configuration Directory Structure.....	31

Collecting Navigator Encrypt Environment Information.....	32
Upgrading Navigator Encrypt Hosts.....	33

Cloudera Navigator Encrypt Overview

Cloudera Navigator Encrypt transparently encrypts and secures data at rest without requiring changes to your applications, and ensures minimal performance lag in the encryption or decryption process. Advanced key management with [Cloudera Navigator Key Trustee Server](#) and process-based access controls in Navigator Encrypt enable organizations to meet compliance regulations and prevent unauthorized parties or malicious actors from gaining access to encrypted data.

Navigator Encrypt features include:

- **Automatic key management:** Encryption keys are stored in Key Trustee Server to separate the keys from the encrypted data. If the encrypted data is compromised, it is useless without the encryption key.
- **Transparent encryption and decryption:** Protected data is encrypted and decrypted seamlessly, with minimal performance impact and no modification to the software accessing the data.
- **Process-based access controls:** Processes are authorized individually to access encrypted data. If the process is modified in any way, access is denied, preventing malicious users from using customized application binaries to bypass the access control.
- **Performance:** Navigator Encrypt supports the Intel AES-NI cryptographic accelerator for enhanced performance in the encryption and decryption process.
- **Compliance:** Navigator Encrypt enables you to comply with requirements for HIPAA-HITECH, PCI-DSS, FISMA, EU Data Protection Directive, and other data security regulations.
- **Multi-distribution support:** Navigator Encrypt supports Debian, Ubuntu, RHEL, CentOS, and SLES.
- **Simple installation:** Navigator Encrypt is distributed as RPM and DEB packages, as well as SLES KMPs.
- **Multiple mountpoints:** You can separate data into different mountpoints, each with its own encryption key.

Navigator Encrypt can be used with many kinds of data, including (but not limited to):

- Databases
- Temporary files (YARN containers, spill files, and so on)
- Log files
- Data directories
- Configuration files

Navigator Encrypt uses dmccrypt for its underlying cryptographic operations. Navigator Encrypt uses several different encryption keys:

- **Master Key:** The master key can be a single passphrase, dual passphrase, or RSA key file. The master key is stored in Key Trustee Server and cached locally. This key is used when registering with a Key Trustee Server and when performing administrative functions on Navigator Encrypt clients.
- **Mount Encryption Key (MEK):** This key is generated by Navigator Encrypt using openssl rand by default, but it can alternatively use /dev/urandom. This key is generated when preparing a new mount point. Each mount point has its own MEK. This key is uploaded to Key Trustee Server.
- **dmccrypt Device Encryption Key (DEK):** This key is not managed by Navigator Encrypt or Key Trustee Server. It is managed locally by dmccrypt and stored in the header of the device.

Related Information

[Installing Cloudera Navigator Encrypt](#)

Process-Based Access Control List

The access control list (ACL) controls access to specified data. The ACL uses a process fingerprint, which is the SHA256 hash of the process binary, for authentication. You can create rules to allow a process to access specific files or directories. The ACL file is encrypted with the client master key and stored locally for quick access and updates.

Here is an example rule:

```
"ALLOW @mydata * /usr/bin/myapp"
```

This rule allows the /usr/bin/myapp process to access any encrypted path (*) that was encrypted under the category @mydata.



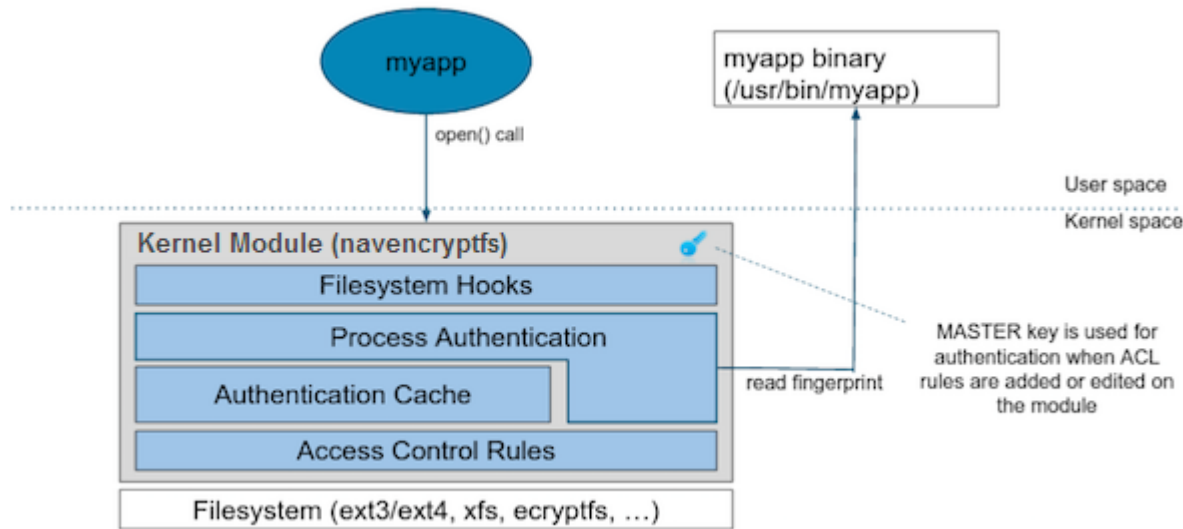
Note: You have the option of using wildcard characters when defining process-based ACLs. The following example shows valid wildcard definitions:

```
"ALLOW @* * *"  
"ALLOW @* path/* /path/to/process"
```

Navigator Encrypt uses a kernel module that intercepts any input/output (I/O) sent to an encrypted and managed path. The Linux module file name is navencryptfs.ko and it resides in the kernel stack, injecting file system hooks. It also authenticates and authorizes processes and caches authentication results for increased performance.

Because the kernel module intercepts and does not modify I/O, it supports any file system (ext3, ext4, xfs, and so on).

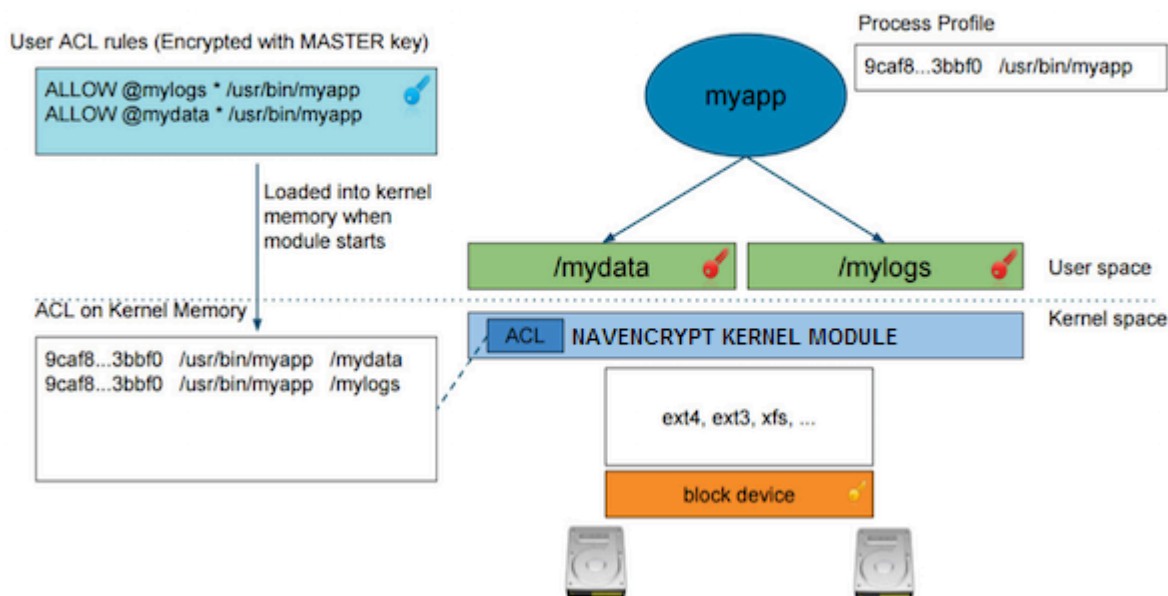
The following diagram shows /usr/bin/myapp sending an open() call that is intercepted by navencrypt-kernel-module as an open hook:



The kernel module calculates the process fingerprint. If the authentication cache already has the fingerprint, the process is allowed to access the data. If the fingerprint is not in the cache, the fingerprint is checked against the ACL. If the ACL grants access, the fingerprint is added to the authentication cache, and the process is permitted to access the data.

When you add an ACL rule, you are prompted for the master key. If the rule is accepted, the ACL rules file is updated as well as the navencrypt-kernel-module ACL cache.

The next diagram illustrates different aspects of Navigator Encrypt:



The user adds a rule to allow `/usr/bin/myapp` to access the encrypted data in the category `@mylogs`, and adds another rule to allow `/usr/bin/myapp` to access encrypted data in the category `@mydata`. These two rules are loaded into the `navencrypt-kernel-module` cache after restarting the kernel module.

The `/mydata` directory is encrypted under the `@mydata` category and `/mylogs` is encrypted under the `@mylogs` category using `dmccrypt` (block device encryption).

When `myapp` tries to issue I/O to an encrypted directory, the kernel module calculates the fingerprint of the process (`/usr/bin/myapp`) and compares it with the list of authorized fingerprints in the cache.

Encryption Key Storage and Management

The master key and mount encryption keys are securely deposited in Key Trustee Server. One MEK per mount point is stored locally for offline recovery and rapid access. The locally-stored MEKs are encrypted with the master key.

The connection between Navigator Encrypt and Key Trustee Server is secured with TLS/SSL certificates.

The following diagram demonstrates the communication process between Navigator Encrypt and Key Trustee Server:



The master key is encrypted with a local GPG key. Before being stored in the Key Trustee Server database, it is encrypted again with the Key Trustee Server GPG key. When the master key is needed to perform a Navigator Encrypt operation, Key Trustee Server decrypts the stored key with its server GPG key and sends it back to the client (in this case, Navigator Encrypt), which decrypts the deposit with the local GPG key.

All communication occurs over TLS-encrypted connections.

Registering Cloudera Navigator Encrypt with Key Trustee Server

Prerequisites

Functioning Navigator Key Trustee Server

After installing Navigator Encrypt on a host, you must register the host with Navigator Key Trustee Server. If you have not yet installed Navigator Key Trustee Server, see [Installing Cloudera Navigator Key Trustee Server](#) for instructions.

Key Trustee Server Organization

To register with Key Trustee Server, you must have an existing organization. See [Managing Key Trustee Server Organizations](#) for information about creating and viewing organizations in Key Trustee Server.

Master Password

The Master Key is the primary Navigator Encrypt administrator access code and is configured by the Navigator Encrypt administrator during installation. The Master Key can take any one of three different forms:

- If you choose a passphrase (single), it must be between 15 and 32 characters long.
- If you choose passphrase (dual), both must be between 15 and 32 characters long.
- If you choose the RSA option, enter a path to the RSA key file, and if it has RSA passphrase, enter it for this private key.



Warning: It is extremely important that you keep your master password secret and safe. In the event that you lose your master password, you will never be able to recover it, leaving your encrypted data irretrievably locked away.

Registering with Key Trustee Server

After installing Navigator Encrypt on a host, you must register the host with Navigator Key Trustee Server in order to encrypt and decrypt data. The following section lists the command options for registering your Navigator Encrypt client.



Note: Do not run Navigator Encrypt commands with the screen utility.

If the TLS certificate is signed by an internal CA that is not publicly recognized, then you must add the root certificate to the host certificate truststore of each Navigator Encrypt client.

Example command:

```
sudo navencrypt register --server=https://keytrustee01.example.com:11371
--passive-server=https://keytrustee02.example.com:11371 --o
rg=your_keytrustee_org --auth=org_auth_token
```


Table 1: Registration Options

Command Option	Explanation
<code>--clientname=my_client_name</code>	User-defined unique name for this client to be used for administration and reports. You can verify your client name in the <code>/etc/navencrypt/keytrustee/clientname</code> file.
<code>--server=https://keytrustee01.example.com:11371</code>	Target Active Key Trustee Server for key storage. Replace <code>keytrustee01.example.com:11371</code> with the hostname and port of the Active Key Trustee Server. The default port is 11371.
<code>--passive-server=https://keytrustee02.example.com:11371</code>	Target Passive Key Trustee Server for key storage. Replace <code>keytrustee02.example.com:11371</code> with the hostname and port of the Passive Key Trustee Server. The default port is 11371.
<code>--org=your_keytrustee_org</code>	Key Trustee organization name configured by the Key Trustee Server administrator
<code>--auth=org_auth_token</code>	Organization authorization token, a pre-shared secret by the Navigator Key Trustee Server administrator
<code>--skip-ssl-check</code>	Skip SSL certificate verification. Use with self-signed certificates on the Navigator Key Trustee Server
<code>--trustee</code>	Add trustees for retrieval of the master key
<code>--votes</code>	Configure voting policy for trustees
<code>--recoverable</code>	Master Key will be uploaded without encrypting it with your local GPG Navigator Key Trustee
<code>--scheme "<scheme>"</code>	Key Trustee Server scheme that Navigator Encrypt uses for public key operations. Specify "http" or "https".
<code>--port</code>	Key Trustee Server port that Navigator Encrypt uses for public key operations.

Registering with Previous Versions of Key Trustee Server

By default, new installations of Navigator Key Trustee Server 5.4.0 use a single HTTPS port for key storage and public key operations. Previous versions and upgrades use separate ports for key storage and public key operations. For backward compatibility, Navigator Encrypt 3.7.0 introduces the `--scheme` and `--port` parameters for the `navencrypt register` command.

For example, to register a version 3.7.0 Navigator Encrypt client with a version 3.8.0 Key Trustee Server using HTTPS over port 443 for key storage and HTTP over port 80 for public key operations, run the following command:

```
sudo navencrypt register --server=https://keytrustee.example.com:443 --org=key_trustee_org --auth=auth_token --scheme "http" --port 80
```

Navigator Encrypt versions lower than 3.7.0 do not support the `--scheme` and `--port` parameters. For these versions of Navigator Encrypt, you must ensure that the Key Trustee Server is configured to use port 443 (HTTPS) for key storage and port 80 (HTTP) for public key operations.

Navigator Encrypt versions lower than 3.8.0 do not support the `--passive-server` parameter.

Updating Key Trustee Server Ports

The `navencrypt register` command does not provide the ability to change the ports for existing registrations. If the Key Trustee Server ports are changed, you must update `/etc/navencrypt/keytrustee/ztrustee.conf` with the new port and scheme parameters (`HKP_PORT` and `HKP_SCHEME`, respectively).

For example, see the following `ztrustee.conf` excerpt from a registered client that has been upgraded to Navigator Encrypt 3.7.0:

```
{
```

```

"LOCAL_FINGERPRINT" :    "2048R/182AAA838DC300AC334258D8E7F299BFB68
A6F6F" ,
  "REMOTES" :           {
    "kts01.example.com" : {
      "REMOTE_FINGERPRINT" :    "4096R/AF6400E12DC149799CA8
CE6BF1604C34D830DE20" ,
      "REMOTE_SERVER" :        "https://kts01.example.com" ,
      "DEFAULT" :              true ,
      "SSL_INSECURE" :         false ,
      "PROTOCOL" :             "json-encrypt"
    }
  }
}

```

In this example, the Key Trustee Server (keytrustee.example.com) is using the default configuration of port 443 (HTTPS) for key storage and port 80 (HTTP) for public key operations.

If the Key Trustee Server is then updated to use port 11371 (HTTPS) for both key storage and public key operations, you must update ztrustee.conf as follows (changes in bold):

```

{
  "LOCAL_FINGERPRINT" :    "2048R/182AAA838DC300AC334258D8E7F299BFB68
A6F6F" ,
  "REMOTES" :           {
    "kts01.example.com" : {
      "REMOTE_FINGERPRINT" :    "4096R/AF6400E12DC149799CA8
CE6BF1604C34D830DE20" ,
      "REMOTE_SERVER" :        "https://kts01.examp1
e.com:11371" ,
      "HKP_PORT" :             11371 ,
      "HKP_SCHEME" :          "https" ,
      "DEFAULT" :              true ,
      "SSL_INSECURE" :         false ,
      "PROTOCOL" :             "json-encrypt"
    }
  }
}

```

Updating Navigator Encrypt for High Availability Key Trustee Server

If you registered a Navigator Encrypt client with a standalone Key Trustee Server, and then configured [high availability](#) for Key Trustee Server, you can edit /etc/navencrypt/keytrustee/ztrustee.conf to enable the client to take advantage of the high availability features. The following example shows the contents of ztrustee.conf after adding the required REMOTE_SERVERS entry (changes in bold):

```

{
  "LOCAL_FINGERPRINT" :    "2048R/182AAA838DC300AC334258D8E7F299BFB68
A6F6F" ,
  "REMOTES" :           {
    "kts01.example.com" : {
      "REMOTE_FINGERPRINT" :    "4096R/AF6400E12DC149799CA8
CE6BF1604C34D830DE20" ,
      "REMOTE_SERVER" :        "https://kts01.example.co
m:11371" ,
      "HKP_PORT" :             11371 ,
      "HKP_SCHEME" :          "https" ,
      "DEFAULT" :              true ,
      "REMOTE_SERVERS" :       ["https://
kts01.example.com:11371" , "https://kts02.example.com:11371" ] ,
      "SSL_INSECURE" :         true ,
      "PROTOCOL" :             "json-encrypt"
    }
  }
}

```

```
}
}
```

Configuration Files

The installer creates the `/etc/navencrypt` directory. All configuration settings are saved in this directory. Do not delete any file from `/etc/navencrypt`. These files provide the necessary information for the Navigator Encrypt application to function properly.



Warning: Perform backups of encrypted data, mount-points, and Navigator Encrypt configuration directories on a regular basis. To do this, ensure you have a backup of `/etc/navencrypt`. Failure to backup this directory will make your backed up encrypted data unrecoverable in the event of data loss.

Change Master Key by UUID

It is possible to re-use a previously used Master Key by its UUID. For example, if you currently have a Master key with a single passphrase, you can see the corresponding Navigator Key Trustee UUID in the `/etc/navencrypt/control` file:

```
$ cat /etc/navencrypt/control
{
  "app": {
    "name": "navencrypt",
    "version": "3.5"
  },
  "keys": {
    "master": {
      "type": "single-passphrase",
      "uuid": "qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L"
    },
    "targets": []
  }
}
```



Note: If the control file is accidentally deleted, you can restore it using the `navencrypt control --restore-control-file` command.

You can copy the UUID (`qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L` in this example) and run `navencrypt key --change` with option `--new-master-key-uuid` to change a Master Key by using its UUID only:

```
$ sudo navencrypt key --change --new-master-key-uuid=qMAKRMdk4HVbhfzR79cp9w92YBmNHJ5nSLhfd8ZVo6L
>> Type your OLD Master key
Type MASTER passphrase 1:
Type MASTER passphrase 2:
Verifying Master Key against Navigator Key Trustee (wait a moment)...
OK
Changing Master key (wait a moment)...
* Setting up EXISTING MASTER key...
* Uploading CONTROL content...
* Re-encrypting local keys...
Master key successfully changed.
```



Note: The `navencrypt key` command fails if no volumes are encrypted or the kernel module is not loaded.

Preparing for Encryption Using Cloudera Navigator Encrypt

Before you can encrypt data, you must prepare a storage repository to hold the encrypted data and a mount point through which to access the encrypted data. The storage repository and mount point must exist before encrypting data using the `navencrypt-move` command.

Data stored and retrieved from the repository is encrypted and decrypted transparently.

Cloudera Navigator Encrypt does not support:

- Encrypting a directory that contains or is contained within a mount point for another service (including Navigator Encrypt and NFS). See [Encrypting Data](#) on page 17 for more information.
- Encrypting immutable files or directories containing immutable files.
- Installation or use in chroot environments, including creating chroot environments within an encrypted directory.
- Encrypting HDFS data files.

Navigator Encrypt Commands



Note: Do not run Navigator Encrypt commands with the `screen` utility.

The following table lists the commands used to encrypt data:

Table 2: Navigator Encrypt Commands

Command	Description
<code>navencrypt</code>	Manage, update, and verify your data.
<code>navencrypt-prepare</code>	Prepare your system for encryption by creating mount-points and specifying storage.
<code>navencrypt-prepare --undo</code>	Remove a mountpoint that is no longer in use.
<code>navencrypt-move</code>	Encrypt/decrypt your data to/from the encrypted file system.
<code>navencrypt-profile</code>	Generate process profile information in JSON format.
<code>navencrypt-module-setup</code>	Build or rebuild the Navigator Encrypt kernel module.

Preparing for Encryption



Note: When using an HSM with Key Trustee Server and Navigator Encrypt, encrypting many block devices may exceed the capacity of the HSM. A key is created in the HSM for each encrypted block device, so be sure that your HSM can support your encryption requirements.

To get an in-depth look at the details behind the `navencrypt-prepare` command, or to use a unique configuration, use the interactive prompt by executing `navencrypt-prepare` with no options. This launches an interactive console that guides you through the following operations:

- Creating internal encryption keys
- Registering internal keys in Navigator Key Trustee
- Registering mount point in `/etc/navencrypt/ztab`
- Mounting current mount point
- Establishing encryption method (`dm-crypt` for devices)

Using the console, you can choose how you want your data stored and accessed. Navigator Encrypt offers block-level encryption with dm-crypt, which protects your data by encrypting the entire device. This enables full disk encryption and is optimized for some system configurations. You can use block-level encryption with logical devices such as a loop device.

See [Block-Level Encryption with dm-crypt](#) on page 13 for more information.

To prepare for encryption, you must specify a location to store the encrypted data and a mount point through which to access the data. The storage location and mount point must be created before encrypting data.



Note: If you are performing a file system check as part of your preparation work, then note that the crypto device must be mapped and active. Also, be aware that if you execute fsck in force mode (-f), there is a risk of data loss. If the force operation fails, it could cause file system corruption at the device header.

Block-Level Encryption with dm-crypt

When choosing block-level encryption in the interactive console, you must specify two parameters:

1. The first parameter is the block device that you want to store the encrypted file system in. Because this device stores all of the encrypted data, it must be as large as or larger than the target data. The device must exist and be empty. Supported storage devices are:
 - Physical block devices (for example, a disk device)
 - Virtual block devices (for example, a block device created by [LVM](#))
 - Loop devices (see [Block-Level Encryption with a Loop Device](#) on page 14 for instructions on creating a loop device)



Important: If the block device to be used for encryption was previously used by the host, entries for it must be removed from the file /etc/fstab before running the navencrypt-prepare command.

2. The second parameter is the mount point for the encrypted file system. This is the location where you can access the encrypted data stored in the first parameter. The mount point must already exist. It is not created by the navencrypt-prepare command.

The entire device in the first parameter is used for encrypted data.



Note: Do not manually unmount the encryption mount point (for example, using umount). If you do so, you must manually close the dm-crypt device using the following procedure:

1. Run dmsetup table to list the dm-crypt devices.
2. Run cryptsetup luksClose <device_name> to close the device for the unmounted mount point.

After specifying these two parameters and following the interactive console (discussed further in [Preparing for Encryption](#) on page 12), you are ready to encrypt your data.

The syntax for the prepare command is as follows:

```
sudo navencrypt-prepare <device_name> <mount_point>
```

When specifying the mount point path, do not use a trailing / in the path names. The mount point directory must exist prior to running the navencrypt-prepare command.

For RHEL 7, run the following command after the navencrypt-prepare command completes:

```
sudo systemctl start navencrypt-mount
```

The following example shows successful output from a navencrypt-prepare command using dm-crypt for block-level encryption:

```
$ sudo /usr/sbin/navencrypt-prepare urandom /mnt/dm_encrypted
Type MASTER passphrase:
Encryption Type: dmCrypt (LUKS)
```

```

Cipher:          aes
Key Size:       256
Random Interface: /dev/urandom
Filesystem:     ext4
Verifying MASTER key against Navigator Key Trustee (wait a moment) ... OK
Generation Encryption Keys with /dev/urandom ... OK
Preparing dmCrypt device (--use-urandom) ... OK
Creating ext4 filesystem ... OK
Registering Encryption Keys (wait a moment) ... OK
Mounting /dev/sdal ... OK

```

After you have successfully prepared a client for encryption, you can encrypt and decrypt data using the commands described in [Encrypting and Decrypting Data Using Cloudera Navigator Encrypt](#) on page 16.

Block-Level Encryption with a Loop Device

A block-level encrypted device can be a physical device or a storage space treated as a device.

To configure a loop device, use the `dd` command to create a storage space:



Warning: The space for the loop device is pre-allocated. After the loop device is created, the size cannot be changed. Make sure you are allocating enough storage for the current encrypted data as well as any future data.

If your disks are mounted individually with a file system on each disk, and your storage needs exceed the capacity of a single disk, you can create a loop device on each disk for which you want to allocate space for encrypted data. If you have consolidated storage (for example, using LVM), you can create a single loop device or multiple devices.

```
sudo dd if=/dev/zero of=/dmccrypt/storage bs=1G count=500
```

The `dd` command above creates a 500 GB file. Modify the `bs` and `count` values to generate the required file size.

After generating the file, run `losetup -f` to view unused loop devices. Use the available loop device with the `navencrypt pt-prepare -d` command, demonstrated below.

Specifically for loop devices, the `-d` parameter enables Navigator Encrypt to manage the loop device association. You no longer need to use the `losetup` command to associate the file with the loop device, and the loop device is automatically prepared at boot. For RHEL 7-compatible OS, you must run the following commands to ensure that a loop device is available at boot:

```

sudo bash -c 'echo "loop" > /etc/modules-load.d/loop.conf'
sudo bash -c 'echo "options loop max_loop=8" > /etc/modprobe.d/loop_options.conf'

```



Warning: Loop devices are not created by Navigator Encrypt. Instead, Navigator Encrypt assigns a datastore to a loop device when the `navencrypt-prepare --datastore` option is used. So, it is up to the system administrator to create persistent `/dev/loopX` devices, which are required to prepare a virtual block device. If the loop device being prepared is not persistent, then Navigator Encrypt will not mount the device upon a reboot.

The data storage directory name (`/dmccrypt/storage` in the previous example) must contain only alphanumeric characters, spaces, hyphens (-), or underscores (_). Other special characters are not supported.

The following example shows the output from a successful command:

```

$ losetup -f
/dev/loop0
$ sudo navencrypt-prepare -d /dmccrypt/storage /dev/loop0 /dmccrypt/mountpoint
Type MASTER passphrase:

Encryption Type:  dmCrypt (LUKS)

```

```

Cipher:          aes
Key Size:       256
Random Interface: OpenSSL
Filesystem:     ext4
Options:

Verifying MASTER key against KeyTrustee (wait a moment)    ... OK
Generation Encryption Keys with OpenSSL                    ... OK
Assigning '/dev/loop0' -> '/dmccrypt/storage'               ... OK
Preparing dmCrypt device                                  ... OK
Creating ext4 filesystem                                   ... OK
Registering Encryption Keys (wait a moment)                ... OK
Mounting /dev/loop0                                       ... OK

```

For upgraded Navigator Encrypt clients that already use loop devices, you can enable Navigator Encrypt to manage the loop device file association (instead of configuring the system to run the `losetup` command at boot) by adding the `nav_datastore` option to the entry in `/etc/navencrypt/ztab`. For example:

```

# <target mount-dir>      <source device>      <type>      <options>
/dmccrypt/mountpoint      /dev/loop0          luks        key=keytrustee,nav_datastor
e='/dmccrypt/storage'

```



Important: Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

After you have created the loop device, continue with the instructions in [Block-Level Encryption with dm-crypt](#) on page 13.

Navigator Encrypt and Device UUIDs

Navigator Encrypt has always prepared and identified devices simply using a device name, such as `/dev/sdb1` or `/dev/loop0`. However, we know that using a device name or label could lead to a conflict and impact system operations.

Navigator Encrypt also supports preparing devices using a UUID, in addition to device name. This UUID is simply a symbolic link to the actual device, and is created when preparing a device with Navigator Encrypt during a `navencrypt-prepare` operation.

The advantage of using a device UUID is that if a device's name changes, the UUID associated with that device does not change. To ensure that Navigator Encrypt recognizes devices even when the device name changes, enter the command:

```
navencrypt-prepare --use-uuid /dev/sda1 /mountpoint
```

To unprepare (ensure the device UUID is included), enter either of the following commands:

```

navencrypt-prepare --undo-force /dev/disk/by-uuid/3a602a15-11f7-46ac-ae98-0a
51e1b25cf9
navencrypt-prepare --undo /dev/disk/by-uuid/3a602a15-11f7-46ac-ae98-0a51e
1b25cf9

```



Note: While the device name is still used in the `navencrypt-prepare` statement, rest assured that Navigator Encrypt handles the device by the UUID, which is calculated and used for mount during boot. The device name is used for convenience so that you do not have to explicitly input the UUID in the command. Ultimately, Navigator Encrypt handles the device via the device UUID rather than the device name.



Important: UUID device support does not include loop devices; rather, it only applies to physical devices. When preparing loop devices with Navigator Encrypt, always use the device name.

Pass-through Mount Options for navencrypt-prepare

Navigator Encrypt 3.5 and higher provides the ability to specify options to pass to the mount command that is executed during `/etc/init.d/navencrypt-mount start` (systemctl start navencrypt-mount on RHEL 7). These options are specified with the `-o` option when preparing a mountpoint with the `navencrypt-prepare` command.

The following shows an example `navencrypt-prepare` command output when passing mount options with the `-o` option:

```
$ sudo navencrypt-prepare -o discard,resize /mnt/t2 /mnt/t2
Type MASTER passphrase:
Encryption Type: dmCrypt (LUKS)
Cipher: aes
Key Size: 256
Random Interface: OpenSSL
Filesystem: ext4
Options: discard,resize
Verifying MASTER key against Navigator Key Trustee(wait a moment) ... OK
Generation Encryption Keys with OpenSSL ... OK
Registering Encryption Keys (wait a moment) ... OK
Mounting /mnt/t2 ... OK
```

You can verify the results by viewing the `/etc/navencrypt/ztab` file:

```
$ cat /etc/navencrypt/ztab
/mnt/t2 /mnt/t2 dmccrypt key=keytrustee,cipher=aes,keysize=256,discard,resize
```

Options can be added or removed to existing mount points prepared with versions of Navigator Encrypt prior to 3.5 by editing the `/etc/navencrypt/ztab` file and adding the comma-separated options (no spaces) to the end of each line as seen in the previous example above.



Important: Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.

To see the mounted file systems and options, run `mount`:

```
$ mount
/mnt/t2 on /mnt/t2 type dmccrypt (rw,dmccrypt_sig=6de3db1e87077adb,ecryptfs_unlink_sigs,noauto,\
dmccrypt_cipher=aes,dmccrypt_key_bytes=32,discard,resize)
```

For a list of available mount options, see the man pages for `cryptsetup` and `dmCrypt` respectively.

Encrypting and Decrypting Data Using Cloudera Navigator Encrypt



Warning: Before encrypting or decrypting any data, stop all processes (for example, MySQL, MongoDB, PostgreSQL, and so on) that have access to the target data. Failure to do so could lead to data corruption.

After the encrypted file system is created and initialized, it is ready to hold data. All encryption and decryption functionality is performed with a single command: `navencrypt-move`.

Do not manually create directories or files under a Cloudera Navigator Encrypt mount point; use only the `navencrypt-move` command to encrypt and decrypt data. See [Preparing for Encryption Using Cloudera Navigator Encrypt](#) on page 12 for more information about mount points.

After encrypting a file or directory, all data written and read through the mount point is transparently encrypted and decrypted.

Before You Begin

Navigator Encrypt does not support encrypting data in certain environments, including the following:

- Do not attempt to encrypt a directory that contains or is contained within a mount point for another service (including Navigator Encrypt and NFS). For example:
 - If your encryption mount point is `/var/lib/navencrypt/mount`, do not attempt to encrypt `/var`, `/var/lib`, `/var/lib/navencrypt`, `/var/lib/navencrypt/mount`, or anything under `/var/lib/navencrypt/mount/`.
 - If you have mounted an NFS file system at `/mnt/home`, do not attempt to encrypt `/mnt`, `/mnt/home`, or anything under `/mnt/home`.
- Do not attempt to encrypt immutable files or directories containing immutable files.
- Do not use Navigator Encrypt within a chroot environment, or create a chroot environment within an encrypted directory.
- If your Key Trustee Server is managed by Cloudera Manager, do not encrypt the Cloudera Manager database with Navigator Encrypt; doing so prevents Cloudera Manager from starting.

Encrypting Data

Do not manually create directories or files under a Navigator Encrypt mount point; use only the `navencrypt-move` command to encrypt data.


Here is an example command to encrypt data, with an explanation for each option:

```
sudo navencrypt-move encrypt @<category> <directory_or_file_to_encrypt> <encrypted_mount_point>
```



Important: Do not run `navencrypt-move` commands simultaneously in multiple terminals. Doing so results in failure to encrypt or decrypt all of the specified data. No data is lost, as the source data is not removed, but you must re-run the failed operations sequentially.

Table 3: navencrypt-move Command Options

Command Option	Explanation
<code>navencrypt-move</code>	Main command interface for all actions that require moving data either to or from the encrypted file system. For more information see the <code>navencrypt-move</code> man page (<code>man navencrypt-move</code>).
<code>encrypt</code>	Identifies the cryptographic operation, in this case, encrypting data. The <code>decrypt</code> option is described later in Decrypting Data on page 18.  Note: By default, all Navigator Encrypt encryption commands require free space equal to twice the size of the encrypted data. If your environment does not have enough free space, add <code>--per-file</code> to the end of the command. This moves each file individually. Per-file encryption only requires free space equal to twice the size of the largest individual file, but is a slower operation.

Command Option	Explanation
@<category>	The access category that is applied to the data being encrypted. Encrypted data is protected by process-based access controls that restrict access to only the processes that you allow. You can use any naming convention you want (the @ symbol is required), but Cloudera recommends keeping it simple and memorable. For example, you can use a name referencing the data type being encrypted, such as @mysql for a MySQL deployment. See Listing Categories on page 30 for instructions on viewing existing categories.
<directory_or_file_to_encrypt>	The data that you want to encrypt. This can be a single file or an entire directory. Navigator Encrypt starts after the system boots, so do not encrypt required system files and directories (such as the root partition, /var, and so on). Some examples of recommended data directories to encrypt are /var/lib/mysql/data, /db/data, and so on.
<encrypted_mount_point>	Where you want to store the data. This is the path to the mount point specified during the navencrypt-prepare command.

When a file is encrypted, a symbolic link (symlink) is created which points to a mount point @<category> directory. The navencrypt-move command moves all specified data to the encrypted file system and replaces it with a symlink to the mount point for that encrypted file system.

Encrypting a directory is similar to encrypting a file. The following command encrypts a directory:

```
sudo /usr/sbin/navencrypt-move encrypt @mycategory /path/to/directory_to_encrypt/ /path/to/mount
```

In this command, a directory is specified instead of a file name, and a symlink is created for that particular directory. To see the effects of this command, run:

```
ls -l <directory_to_encrypt>
du -h <encrypted_storage_directory>
```

The output demonstrates the new file system layout. Everything that was in the target directory is now securely stored in the encrypted file system.

Decrypting Data

The decryption command requires only the path to the original data, which is now a symbolic link, as an argument. The following example demonstrates how to decrypt a file using the navencrypt-move command:

```
sudo /usr/sbin/navencrypt-move decrypt /path/to/encrypted/directory_or_file
```



Important: Do not run navencrypt-move commands simultaneously in multiple terminals. Doing so results in failure to encrypt or decrypt all of the specified data. No data is lost, as the source data is not removed, but you must re-run the failed operations sequentially.

As with encryption, you can specify a directory instead of a file:

```
sudo /usr/sbin/navencrypt-move decrypt /path/to/encrypted/directory
```

Converting from Device Names to UUIDs for Encrypted Devices

When configuring a Navigator Encrypt mount point, you can use either the device name (sample device name: `/dev/sdb1`) or the UUID (sample UUID: `3a602a15-11f7-46ac-ae98-0a51e1b25cf9`) to configure the mount point. For more information, see [Navigator Encrypt and Device UUIDs](#) on page 15.

The UUID mount point configuration choice is preferable for Navigator Encrypt, because the device name can change when the system is rebooted or new disks are added, but the UUID never changes. You can only use the UUID (`--use-uuid`) configuration option during the initial mount point configuration. You cannot use the UUID mount point configuration option on existing mount points that were created in Navigator Encrypt 3.12.0 or earlier, or that were configured to use the device name instead of the UUID.

For configurations where a device name was used, and you wish to convert to a UUID, you have the option to use the UUID conversion utility (`navencrypt-prepare --convert-uuid`), which converts existing mount points from using the device name to using the UUID. The UUID conversion utility updates the Navigator Encrypt configuration files to use UUIDs, and then reloads the updated mount points. You can perform this conversion on mount points with encrypted data without any impact on that data.



Important: Cloudera strongly recommends that Navigator Encrypt mount points that are currently using the device name be converted to use the device UUID. This will prevent error situations when the Navigator Encrypt mount points cannot be mounted, and the encrypted data cannot be accessed because the device name changed.

Preparing for UUID Conversion

The UUID conversion utility runs as part of the `navencrypt-prepare` command:

```
# navencrypt-prepare -h
...
--convert-uuid --all|--device=DEVICE
    Converts devices that are stored using the device name to instead use i
    ts uuid.
    '--all' will attempt to convert all device names in the navencrypt ztab
    file.
    '--device=DEVICE' will convert a single device.
...
```

As shown, you have two options when running the UUID conversion utility:

- `--device=DEVICE`

Use this option to run the conversion against a single specified device.

- `--all`

Use this option to run the conversion against all mount points that are currently using the device name instead of the UUID as defined in the `/etc/navencrypt/ztab` file.

The conversion utility first backs up all files in `/etc/navencrypt` to a timestamped directory under `/tmp`. Ensure that the user running the `navencrypt-prepare` operation has permission to write to `/tmp`. If this backup directory is not created, then the conversion script will exit without making any changes to the Navigator Encrypt configuration files.



Note: Because the conversion utility backs up the Navigator Encrypt configuration files, it is not necessary to back up the files separately. However, if you want the files backed up elsewhere (somewhere other than the /tmp folder), then manually perform this backup by copying the /etc/navencrypt directory to the desired location before running the conversion utility. Note that the conversion utility will still attempt to create its own backup under /tmp, and will not run if this backup cannot be created.

Converting a Single Device

Use the UUID conversion utility `--device=DEVICE` option to convert from a device name to UUID for a mount point on a single device. It is recommended that you test the conversion against at least one device before running it against all devices:

```
# navencrypt-prepare --convert-uuid --device=/dev/xvdd

Backing up /etc/navencrypt to /tmp/navencrypt_bkup_20180917_135716 ...

Stopping navencrypt-mount...

Stopping navencrypt directories

Unmounting /dev/xvdd ... [ OK ]
Unmounting /dev/xvde ... [ OK ]
Unmounting /dev/xvdf ... [ OK ]
* Unloading module ... [ OK ]

Running conversion on /dev/xvdd
...
..
```

You must [approve and finalize](#) the conversion before it is complete.

Converting All Available Devices

Use the UUID conversion utility `--all` option to convert from the device name to the device UUID on all available devices. The conversion utility skips loop devices because they do not have a dedicated UUID:

```
# navencrypt-prepare --convert-uuid --all

Backing up /etc/navencrypt to /tmp/navencrypt_bkup_20180917_135731 ...

Stopping navencrypt-mount...
Stopping navencrypt directories
* Unmounting /dev/xvdd ... [ OK ]
* Unmounting /dev/xvde ... [ OK ]
* Unmounting /dev/xvdf ... [ OK ]
* Unloading module ... [ OK ]

Running conversion on /dev/xvdd...

Running conversion on /dev/xvde...

Running conversion on /dev/xvdf...
...
```

You must [approve and finalize](#) the conversion before it is complete.

Finalizing the Conversion

Before the device name-to-UUID conversion is applied to the configuration files, you must review and approve the differences between the `/etc/navencrypt/ztab` and `/etc/navencrypt/control` by either accepting the changes ("y"), or rejecting them ("n").

```
# navencrypt-prepare --convert-uuid --all
...
Running conversion on /dev/xvdd...
Running conversion on /dev/xvde...
Running conversion on /dev/xvdf...

Showing diff of ztab and control files

-----
                                ZTAB
-----
2,4c2,4
< /navencrypt_mount/block1      /dev/xvdd      luks      key=keytrustee
< /navencrypt_mount/block2      /dev/xvde      luks      key=keytrustee
< /navencrypt_mount/block3      /dev/xvdf      luks      key=keytrustee
---
/navencrypt_mount/block1        /dev/disk/by-uuid/4206d6d5-6014-435a-b342-1d3
dad5559a2      luks      key=keytrustee
> /navencrypt_mount/block2      /dev/disk/by-uuid/b84c4f38-bc74-40bc-87eb-2
e857a996933    luks      key=keytrustee
> /navencrypt_mount/block3      /dev/disk/by-uuid/622312d0-0e6c-4e37-adeb-f60
66a1df07d      luks      key=keytrustee
-----

                                CONTROL
-----
8c8
<   "name" :      "/dev/xvdd" ,
---
>   "name" :      "/dev/disk/by-uuid/4206d6d5-6014-435a-b342-1d3dad5559a2" ,
11c11
<   "name" :      "/dev/xvde" ,
---
>   "name" :      "/dev/disk/by-uuid/b84c4f38-bc74-40bc-87eb-2e857a996933" ,
14c14
<   "name" :      "/dev/xvdf" ,
---
>   "name" :      "/dev/disk/by-uuid/622312d0-0e6c-4e37-adeb-f6066a1df07d" ,

Accept changes? [y/N] y
Moving KeyTrustee deposit for /dev/xvdd...
Moving KeyTrustee deposit for /dev/xvde...
Moving KeyTrustee deposit for /dev/xvdf...
Operation complete
Overwriting old files...
Starting navencrypt-mount...
Starting navencrypt directories
* Mounting '4206d6d5-6014-435a-b342-1d3dad5559a2'          [ OK ]
* Mounting 'b84c4f38-bc74-40bc-87eb-2e857a996933'       [ OK ]
* Mounting '622312d0-0e6c-4e37-adeb-f6066a1df07d'       [ OK ]
```

If you reject the changes, the updated files are saved to `/etc/navencrypt/ztab.new` and `/etc/navencrypt/control.new`, and the existing `ztab` and `control` files remain unchanged.



Warning: Do not manually copy the `ztab.new` and `control.new` files over the existing `ztab` and `control` files. If upon review the new files are correct, then re-issue `navencrypt-prepare --convert-uuid` and select “y” to use the updated files. If you modify the `ztab` and `control` files outside of the `navencrypt-prepare` script, then you will encounter a conflict in the saved keytrustee deposits that will prevent the Navigator Encrypt mount points from mounting.

```
# navencrypt-prepare --convert-uuid --all
...
Accept changes? [y/N] n

Changes will not be applied. Proposed changes are saved to /etc/navencrypt/
ztab.new and /etc/navencrypt/control.new
```

Rolling Back the UUID Conversion

If you experience any problems with the device name-to-UUID conversion, you can restore the previous mount point state by replacing the contents of the `/etc/navencrypt/directory` with the backup created by the conversion utility:

```
# service navencrypt-mount stop
# rm /etc/navencrypt/keytrustee/deposits/dev.disk.by-uuid.*
# cp -rp /tmp/navencrypt_bkup_date_time/* /etc/navencrypt/
# service navencrypt-mount start
```

Navigator Encrypt Access Control List

Managing the Access Control List

Cloudera Navigator Encrypt manages file system permissions with an access control list (ACL). This ACL is a security access control created by Cloudera that enables a predefined Linux process to access a file or directory managed by Navigator Encrypt.

The ACL uses rules to control process access to files. The rules specify whether a Linux process has access permissions to read from or write to a specific Navigator Encrypt path.

A rule is defined in the following order:

```
# TYPE @CATEGORY PATH PROCESS PARAMETERS
```

The following table defines the ACL rule components:

Table 4: ACL Rule Components

Component	Description
TYPE	Specifies whether to allow or deny a process. It can have either of the following values: ALLOW or DENY.
@CATEGORY	This is a user-defined shorthand, or container, for the encrypted dataset that the process will have access to. For example, if you are encrypting the directory <code>/var/lib/mysql</code> , you could use the category <code>@mysql</code> to indicate that this rule is granting access to a process on the MySQL data. See Listing Categories on page 30 for instructions on viewing existing categories.
PATH	Specifies the rights permissions of a specific path. For example: <code>*</code> , <code>www/*.htaccess</code> . Omit the leading slash (<code>/</code>).
PROCESS	Specifies the process or command name for the rule.

Component	Description
PARAMETERS	<p>Tells the process the parent-child process to be executed:</p> <p>--shell defines the script for Navigator Encrypt to allow for executable process. Supported shells are /usr/bin/bash, /bin/bash, /usr/bin/dash, and /bin/bash.</p> <p>--children defines for Navigator Encrypt which child processes to allow that are executed by a process/script.</p> <p>Example: --shell=/bin/bash, --children=/bin/df,/bin/lis</p>

All rules are stored in an encrypted policy file together with a set of process signatures that are used by Navigator Encrypt to authenticate each Linux process. This file is encrypted with the Navigator Encrypt key you defined during installation.

Cloudera recommends using permissive mode to assist with the initial ACL rule creation for your environment. In permissive mode, Navigator Encrypt allows full access to the encrypted data by all processes, but logs them in dmesg as action="denied" messages. Consult these messages to identify required ACL rules. To set Navigator Encrypt to permissive mode, use the following command:

```
sudo /usr/sbin/navencrypt set --mode=permissive
```

To view the current mode, run `navencrypt status -d`. For more information on access modes, see [Access Modes](#).

deny2allow

After you generate the action="denied" messages, use the `navencrypt deny2allow` command to show which ACL rules are required, based on the action="denied" messages in dmesg. To show which ACL rules are required, perform the following steps:

1. Save the dmesg content to a file:

```
sudo dmesg > /tmp/dmesg.txt
```

2. Use the dmesg.txt file content as input to the `deny2allow` command to analyze the action="denied" messages and display a list of required ACL rules based on the action="denied" messages. Here is an example command and output:

```
$ sudo /usr/sbin/navencrypt deny2allow /tmp/dmesg.txt
ALLOW @mysql employees/* /usr/sbin/mysqld
ALLOW @mysql * /bin/bash
ALLOW @mysql * /bin/lis
```

If you need to clear the dmesg log and start fresh, run `dmesg -c`.

If a rule is displayed in the output from the command, it does not automatically mean the ACL rule must be added. You must determine which rules are actually needed. For example, the rule for `ls` would not typically be added as an ACL rule.

After the initial ACL rules are created, disable permissive mode with the following command:

```
sudo /usr/sbin/navencrypt set --mode=enforcing
```

Adding ACL Rules

Rules can be added one at a time using the command line or by specifying a policy file containing multiple rules. The following example shows how to add a single rule using the `navencrypt acl --add` command:

```
sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/sbin/mysqld"
```

See [Listing Categories](#) on page 30 for instructions on viewing existing categories.

The following example shows how to add multiple rules using a policy file:

```
sudo /usr/sbin/navencrypt acl --add --file=/mnt/private/acl_rules
```

The contents of the policy file should contain one rule per line. For example:

```
ALLOW @mysql * /usr/sbin/mysqld
ALLOW @log * /usr/sbin/mysqld
ALLOW @apache * /usr/lib/apache2/mpm-prefork/apache2
```

Navigator Encrypt releases 3.10 and higher support comments in the policy file. Comments begin with the hash (#) symbol. You can use comments to annotate the policy file, or to temporarily disable a rule for testing. For example:

```
# Cloudera Navigator Encrypt policy file
# Allow mysqld to access all database files
ALLOW @mysql * /usr/sbin/mysqld
# Allow mysqld to write logs
ALLOW @log * /usr/sbin/mysqld
# ALLOW @apache * /usr/lib/apache2/mpm-prefork/apache2
```

Using a policy file is the fastest way to add multiple rules because it only requires the security key one time.

It is also possible to overwrite the entire current rules set with the option `--overwrite`. When this command is executed, all current rules are replaced by the ones specified in the file that contains the new set of rules. Cloudera recommends to save a copy of your current set of rules by printing it with the option `--print`.

Here is an example command using the `--overwrite` option:

```
sudo /usr/sbin/navencrypt acl --overwrite --file=/mnt/private/acl_rules
```

Adding ACL Rules by Profile

If your environment requires more granular controls on the processes that can access the data, you can add extra controls by using profiles. Profiles set requirements on a process other than just having the correct fingerprint. They can include such things as process owner and group, required open files, and the current working directory. To see more about adding rules by profile, see [ACL Profile Rules](#) on page 26. For details about fingerprints, see [Process-Based Access Control List](#) on page 5.

Deleting ACL Rules

Rules can be deleted in one of two ways:

1. Manually specifying the rule to delete using the command line.
2. Specifying the line number of the rule to delete.

The following example shows how to delete a rule by passing it as a parameter:

```
sudo /usr/sbin/navencrypt acl --del --rule="ALLOW @mysql * /usr/sbin/mysqld"
```

If you remove a MySQL ALLOW rule, the MySQL cache must be cleaned by executing the `FLUSH TABLES;` MySQL statement. Otherwise, it will still be possible to view data from encrypted table.

The following example shows how to delete a rule by specifying a line number:

```
sudo /usr/sbin/navencrypt acl --del --line 3
```


It is also possible to delete multiple ACL rules in a single command:

```
sudo /usr/sbin/navencrypt acl --del --line=1,3
```

See [Printing ACL Rules](#) on page 25 for information on determining line numbers.

Deleting ACL Rules by Profile

See [ACL Profile Rules](#) on page 26 for instructions on deleting rules by profile.

Printing ACL Rules

You can print the current Access Control List using the following command:

```
sudo /usr/sbin/navencrypt acl --print
```

Save the ACL to a file with the `--file` option:

```
sudo /usr/sbin/navencrypt acl --print --file=policy-backup
```

To display additional information about the organization of the policy file, use the `--list` option:

```
sudo /usr/sbin/navencrypt acl --list
```

Universal ACL Rules

Universal ACLs will allow or deny a process access to all files or directories encrypted with Navigator Encrypt.

The rule `ALLOW @* * /process` allows the designated process to access anything from all encrypted categories.

The rule `ALLOW @data * *` allows all processes access to any path under the `@data` category.

The rule `ALLOW @* * *` allows all processes access to all encrypted categories. Cloudera does not recommend using this rule. Use it only in test environments.

Here is an example adding a universal ACL rule and then displaying it:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @* * /usr/sbin/mysqld"
Type MASTER passphrase:
1 rule(s) were added
# navencrypt acl --listType MASTER passphrase:
# - Type      Category      Path      Profile  Process
1   ALLOW     @*           *         /usr/sbin/mysqld
```

Enabling Shell Scripts to Be Detected by ACL

All of the previous rules work for binary files. There may be times a script, such as a shell script, must be allowed to access the encrypted directory.

You can add the script as a rule by indicating the executable binary process of this script using the `--shell` option, for example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash
```

The `--shell` option identifies which executable process is used to execute the script. Supported shells are `/usr/bin/bash`, `/bin/bash`, `/usr/bin/dash`, and `/bin/dash`.

If the script is altered, it will no longer be trusted by the ACL because the fingerprint has changed. If you edit the script you must invoke the update option to update the ACL with the new fingerprint.

In some cases, it may be necessary to grant permissions to sub-processes invoked by scripts. For example, it may be necessary to grant permissions to `/bin/bash` that also allow running the `/bin/df` command to allow the system administrator to check disk capacity through a script run using a crontab entry. By using the `--children` option, you can specify these permissions. For example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash --children=/bin/df
```

The `--children` option tells Navigator Encrypt to allow the `/bin/df` binary process if it is executed by `/root/script.sh`.

To allow more than one sub-process, identify them with the `--children` option as comma-separated values. For example:

```
ALLOW @scripts * /root/script.sh --shell=/bin/bash --children=/bin/df,/bin/ls
```

To add shell-children sub-processes, execute the `navencrypt acl --add` command, for example:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/bin/mysqld
_safe \
--shell=/bin/bash --children=/bin/df,/bin/ls"
```

ACL Profile Rules

If your environment requires more granular controls on the processes that can access the data, you can add extra controls by using profiles. Profiles set requirements on a process other than just having the correct fingerprint. They can include such things as process owner and group, required open files, and the current working directory.

A profile is generated by using the following command:

```
usr/sbin/navencrypt-profile --pid=<pid>
```

The output, by default, will be displayed on the screen. You can redirect the output to a file using the `>` or `>>` redirect operators. You can then edit the JSON output in the file to remove lines you do not want. By default, the profile includes the UID, the short name of the binary or script (identified as `comm`), and the full command line of the running process (including any parameters passed). You can generate information by using one of these flags:

- `-c, --with-cwd`

Output the current working directory

- `-e, --with-egid`

Output the egid

- `-g, --with-gid`

Output the gid

- `-u, --with-euid`

Output the euid

Example output from the `navencrypt-profile` command:

```
{
  "uid": "0",
  "comm": "NetworkManager",
  "cmdline": "NetworkManager -pid-file=/var/run/NetworkManager/NetworkManager.pid",
  "gid": "0",
  "cwd": "/",
  "fd0": "/dev/null",
  "fd1": "/dev/null",
```

```
"fd2" : "/dev/null"
}
```

Some distributions do not support `uid` and `gid`. Make sure that your profile file is correct by executing the following command to verify the expected IDs:

```
ps -p <pid_of_process> -o uid, egid
```

If `cmdline` parameters are variable, such as appending a process start timestamp to a file name, then the process profile will not match on subsequent restarts of the process because the current profile will have an updated timestamp and access will be denied by the ACL. You can mark those parameters as variable inside the profile file. For example, if the `cmdline` of a process is something like this:

```
"cmdline" : "NetworkManager -pid-file=/var/run/NetworkManager/NetworkManager.p
id \
-logfile=/var/log/NetworkManager/log-20130808152300.log"
```

Where `log-20130505122300.log` is a variable `cmdline` parameter, before adding the process profile to the ACL, edit the process profile file and use `##` to specify that a particular parameter is variable:

```
"cmdline" : "NetworkManager -pid-file=/var/run/NetworkManager/NetworkManager.p
id -logfile=##"
```

With the above configuration, the ACL will allow any value for the `-logfile` `cmdline` parameter.

To enable a profile in the ACL, use the additional parameter `--profile-file=<filename>` when adding the rule to the ACL:

```
$ sudo /usr/sbin/navencrypt acl --add --rule="ALLOW @mysql * /usr/sbin/mysql
d" \
--profile-file=/path/to/profile/file
```

To display the profile portion of the rules, use the `--all` parameter with `navencrypt acl --list`:

```
$ sudo /usr/sbin/navencrypt acl --list --all
Type MASTER passphrase:
# - Type Category Path Profile Process
1 ALLOW @mysql * YES /usr/sbin/mysqld
PROFILE:
{"uid": "120", "comm": "mysqld", "cmdline": "mysqld"}
```

Maintaining Cloudera Navigator Encrypt

Manually Backing Up Navigator Encrypt

It is recommended that you back up Navigator Encrypt configuration directory after installation, and again after any configuration updates.

1. To manually back up the Navigator Encrypt configuration directory (`/etc/navencrypt`):

```
zip -r --encrypt nav-encrypt-conf.zip /etc/navencrypt
```

The `--encrypt` option prompts you to create a password used to encrypt the zip file. This password is also required to decrypt the file. Ensure that you protect the password by storing it in a secure location.

2. Move the backup file (`nav-encrypt-conf.zip`) to a secure location.



Warning: Failure to back up the configuration directory makes your backed-up encrypted data unrecoverable in the event of data loss.

Validating Navigator Encrypt Configuration

To validate the Navigator Encrypt deployment, run the following command:

```
sudo navencrypt status --integrity
```

This command verifies that:

- The mount encryption key (MEK) exists for each mount point.
- Each mount point in `/etc/navencrypt/ztab` has a corresponding entry in the control file (`/etc/navencrypt/control`).
- Each mount point directory exists.
- For [loop devices](#), the file used for encrypted storage exists.

The output is similar to the following:

```
$ sudo navencrypt status --integrity
Checking MEKs integrity

    Mountpoint: /dev/loop0
                MEK file exist: ..... [YES]
    Mountpoint: /dev/loop1
                MEK file exist: ..... [YES]

Checking Ztab Mountpoints integrity

    Mountpoint: /dev/loop0
                ztab vs control correspondence: ..... [YES]
                Mountpoint directory exists: ..... [YES]
    Mountpoint: /dev/loop1
                ztab vs control correspondence: ..... [YES]
                Mountpoint directory exists: ..... [YES]

Checking Datastore backend files

    Datastore: '/root/my_storage_test'
                Backend file exist: ..... [YES]
```

Restoring Mount Encryption Keys (MEKs) and Control File

Navigator Encrypt deposits its mount encryption keys (MEKs) and control file (`/etc/navencrypt/control`) in Cloudera Navigator Key Trustee Server. If these files are accidentally deleted, they can be restored from Key Trustee Server using the following commands:

- To restore MEKs:

```
sudo navencrypt key --restore-keys
```

- To restore the control file:

```
sudo navencrypt control --restore-control-file
```

Access Modes

Navigator Encrypt provides three different access modes:

- Enforcing is the default mode in which Navigator Encrypt validates access from all processes against the ACL. To protect your data, enforcing mode must be enabled.
- Permissive mode causes action="denied" messages to be logged in dmesg. It does not prevent access to the encrypted data. This mode is a dry-run feature to run and build ACL rules.
- Admin mode, as well as permissive mode, does not prevent access to the encrypted data. It allows any process to access the information because the ACL rules are not validated against the process. Admin mode does not cause action="denied" messages to be logged in dmesg.

To view the current access mode, run the following command:

```
sudo /usr/sbin/navencrypt status -d
```



Note: The navencrypt status command reports that the navencrypt module is not running if no volumes are encrypted or the kernel module is not loaded.

To change the access mode, use the following command:

```
sudo /usr/sbin/navencrypt set --mode={enforcing|permissive|admin}
```

You cannot change the Navigator Encrypt access mode unless the Navigator Encrypt module is running. To view the status of the Navigator Encrypt module, run `navencrypt status --module`.

To start the Navigator Encrypt module there must be at least one active mount-point. To verify the mount-points status, run the following command:

```
sudo /etc/init.d/navencrypt-mount status
```

For RHEL 7, use `systemctl` instead:

```
sudo systemctl status navencrypt-mount
```

Changing and Verifying the Master Key

You can perform two operations with the `navencrypt key` command: change and verify.

You can verify a key against the Navigator Encrypt module, the Navigator Key Trustee server, or both. For example:

```
sudo /usr/sbin/navencrypt key --verify
sudo /usr/sbin/navencrypt key --verify --only-module
sudo /usr/sbin/navencrypt key --verify --only-keytrustee
```



Note: The navencrypt key command fails if no volumes are encrypted or the kernel module is not loaded.

The master key can be changed in the event that another key-type authentication mechanism or a new master key is required. Valid master key types are single-passphrase, dual-passphrase, and RSA key files. To change the master key type, issue the following command and follow the interactive console:

```
sudo /usr/sbin/navencrypt key --change
```

You can use the `--trustees`, `--votes`, and `--recoverable` options for the new key as described in [Table 1: Registration Options](#) on page 9.

Listing Categories

To list the existing categories for each mount point, run the command `navencrypt-move --list-categories`. For example:

```
$ sudo navencrypt-move --list-categories
Navigator Encrypt Categories found per Mountpoint:
 /dmccrypt-storage
   @mysql
   @keytabs

 /home/jdoe/secrets
   @moms_recipes
   @world_domination_plan
```

Updating ACL Fingerprints

All rules reference a process fingerprint (a SHA256 digest) that is used to authenticate the process into the file system. If the file system detects a fingerprint that is different from the one stored in the ACL, the Linux process is denied access and treated as an untrusted process.

Occasionally this process fingerprint must be updated, such as when software is upgraded. When the fingerprint must be updated, the Navigator Encrypt administrator re-authenticates the process on the ACL by executing the command `navencrypt acl --update`.

The following example demonstrates how to determine when a process fingerprint has been changed and must be updated:

```
$ sudo /usr/sbin/navencrypt acl --list
Type MASTER passphrase:
# - Type Category Path Profile Process
1 !! ALLOW @mysql * /usr/sbin/mysqld
2 ALLOW @log * /usr/sbin/mysqld
3 !! ALLOW @apache * /usr/lib/apache2/mpm-prefork/
```

In the example above, the double exclamation (!!) characters indicate that a process fingerprint has changed and must be updated. Similarly, double E (EE) characters indicate a process read error. This error can be caused by a process that does not exist or that has permission issues.



Note:

For RHEL-compatible OSes, the prelink application may also be responsible for ACL fingerprint issues. Prelinking is intended to speed up a system by reducing the time a program needs to begin. Cloudera highly recommends disabling any automated prelink jobs, which are enabled by default in some systems. As an alternative, Cloudera recommends that you integrate a manual prelink run into your existing change control policies to ensure minimal downtime for applications accessing encrypted data.

To disable prelinking, modify the `/etc/sysconfig/prelink` file and change `PRELINKING=yes` to `PRELINKING=no`. Then, run the `/etc/cron.daily/prelink` script as root. Once finished, automatic prelinking is disabled.

For more information about how prelinking affects your system, see [prelink](#).

Managing Mount Points

The `/etc/init.d/navencrypt-mount` command mounts all mount points that were registered with the `navencrypt-prepare` command and are listed in the `/etc/navencrypt/ztab` file. The possible operations are:

- start

- stop
- status
- restart

For RHEL 7, use `systemctl [start|stop|status|restart] navencrypt-mount`.



Note: On RHEL 7, the `systemctl` command can obscure the exit status of the start and/or stop commands because `navencrypt-mount` is loaded as a kernel module instead of running as a background process. Cloudera strongly recommends that you run `systemctl status navencrypt-mount` whenever you run the start and/or stop command to verify that the status of the `navencrypt` kernel module is still healthy and functioning.



Note: Do not manually unmount the encryption mount point (for example, using `umount`). If you do so, you must manually close the `dm-crypt` device using the following procedure:

1. Run `dmsetup table` to list the `dm-crypt` devices.
2. Run `cryptsetup luksClose <device_name>` to close the device for the unmounted mount point.

When executing the stop operation, the encrypted mount point is unmounted, and your data becomes inaccessible.

The following example shows how to execute `navencrypt-mount status` with some inactive mount points:

```
sudo /etc/init.d/navencrypt-mount status
```

The following example shows how to execute the `navencrypt-mount stop` command:

```
sudo /etc/init.d/navencrypt-mount stop
```

The following example shows how to execute the `navencrypt-mount start` command:

```
sudo /etc/init.d/navencrypt-mount start
```

Here is an example command used to manually mount a directory:

```
sudo /usr/sbin/mount.navencrypt /path/to_encrypted_data/ /path/to/mountpoint
```

This command can be executed only if the `navencrypt-prepare` command was previously executed.

Navigator Encrypt Kernel Module Setup

If the kernel headers were not installed on your host, or if the wrong version of the kernel headers were installed, the Navigator Encrypt module was not built at installation time. To avoid reinstalling the system, install the correct headers and execute the `navencrypt-module-setup` command. This attempts to build the module and install it.

This method is also an efficient way to install any new Navigator Encrypt module feature or fix without otherwise modifying your current Navigator Encrypt environment.


Navigator Encrypt Configuration Directory Structure

The file and directory structure of `/etc/navencrypt` is as follows:

```
$ tree /etc/navencrypt/
/etc/navencrypt/
### control -> /etc/navencrypt/jSpi9SM65xUIIhraulNn8ZXmQhrrQ9e363EUz8HKiRs
### jSpi9SM65xUIIhraulNn8ZXmQhrrQ9e363EUz8HKiRs
### rules
### ztab
locust
### keytrustee
### clientname
```

```
### deposits
#   ### dev.loop0
#   ### media.31E5-79B9locustlocust[system ~]# . /etc/*release[system ~]#
. /etc/*release
#   ### mnt.a
#   ### mnt.encrypted
#   ### mnt.tomount
### pubring.gpg
### pubring.gpg~
### random_seed
### secring.gpg
### trustdb.gpg
### ztrustee.conf
```

The following files and folders are part of the created file structure:

- control
File that saves information about the mount points and corresponding Navigator Key Trustee keys. If this file is accidentally deleted, you can restore it using the `navencrypt control --restore-control-file` command.
- rules
File that contains the ACL rules. It is encrypted with the user-provided master key.
- ztab
File that contains information about all the mount points and their encryption type.
 **Important:** Use caution when editing the `/etc/navencrypt/ztab` file. Entries are tab-separated (not space-separated). The `ztab` file must not contain empty lines.
- keytrustee
Directory where Navigator Key Trustee GPG keys are stored. These are generated during `navencrypt register` operations.
- keytrustee/deposits
Directory where the Navigator Encrypt mount encryption keys (MEKs) are saved. These are encrypted with the user-provided master key. If these are accidentally deleted, you can restore them from Key Trustee Server using the `navencrypt key --restore-keys` command.

Every mount point has an internal randomly-generated encryption passphrase.

Collecting Navigator Encrypt Environment Information

When troubleshooting problems with Navigator Encrypt, it is helpful to gather information about the installation and environment. Navigator Encrypt provides a command to facilitate this:

```
sudo navencrypt-collect
```

This command collects and outputs to the console the following information:

- Information about the system on which Navigator Encrypt is installed
- Entries from `/etc/navencrypt/ztab`
- The contents of the `keytrustee.conf` file
- Recent entries from the Navigator Encrypt log file
- Configured software repositories
- Checksums of all `/usr/src/navencrypt*` and `/usr/sbin/navencrypt*` files

You can use this information to compare Navigator Encrypt installations, and you can also provide this information to Cloudera Support for troubleshooting. The `navencrypt-collect` command only outputs this information on the console, and does not generate any files or upload to Cloudera.

To save the information to a file, use the redirect operator (>). For example:

```
sudo navencrypt-collect > navencrypt.info
```

Upgrading Navigator Encrypt Hosts

See "Best Practices for Upgrading Navigator Encrypt Hosts" in the Upgrading Cloudera Navigator Encrypt guide for information about upgrading operating systems (OS) and kernels on hosts that have Navigator Encrypt installed.