Cloudera Manager 7.13.2

Cloudera Navigator Encrypt

Date published: 2020-11-30
Date modified: 2026-02-27

CLOUD∃RA

Legal Notice

Contents

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 Ranger KMS 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 Ranger KMS 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 dmcrypt 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 Ranger KMS and cached locally. This key is used when registering with Ranger KMS 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 Ranger KMS.
- dmcrypt Device Encryption Key (DEK): This key is not managed by Navigator Encrypt or Key Manager Service. It is managed locally by dmcrypt 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:
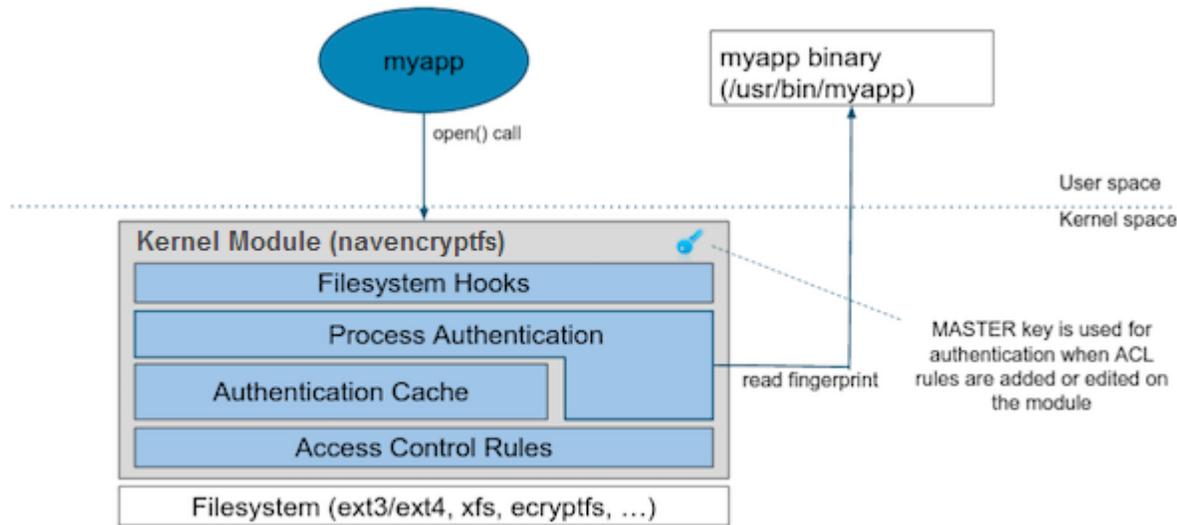
The kernel module calculates the process fingerprint. If the authentication cache already has the fingerprint, the process is allowed to access the data. If the fingerprint is not in the cache, the fingerprint is checked against the ACL. If the ACL grants access, the fingerprint is added to the authentication 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:
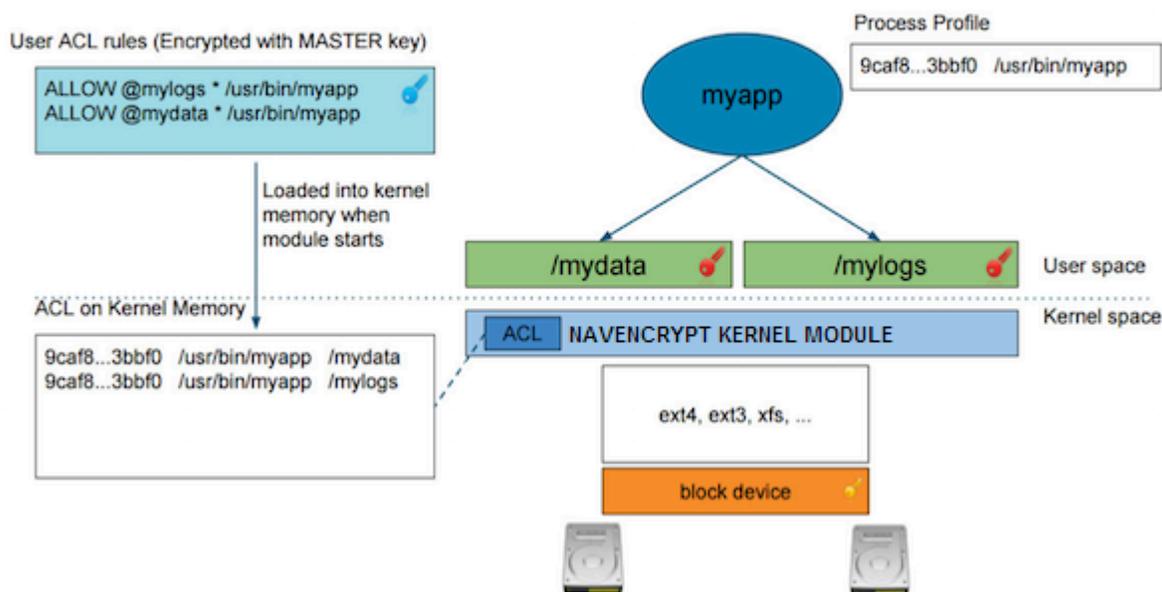
The user adds a rule to allow /usr/bin/myapp to access the encrypted data in the category @mylogs, and adds another rule to allow /usr/bin/myapp to access encrypted data in the category @mydata. These two rules are loaded into the navencrypt-kernel-module cache after restarting the kernel module.

The /mydata directory is encrypted under the @mydata category and /mylogs is encrypted under the @mylogs category using dmcrypt (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 with Ranger KMS

The master key and mount encryption keys are securely deposited in Ranger KMS. 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 Ranger KMS can be secured with TLS/SSL certificates, it is recommended that TLS/SSL is used.

The following diagram demonstrates the communication process between Navigator Encrypt and Ranger KMS:

The master key is encrypted by Navigator Encrypt using a local GPG key and stored in the Ranger KMS database. When the master key is needed to perform a Navigator Encrypt operation, Navigator Encrypt fetches the key from Ranger KMS and decrypts the key using the local GPG key.

For details about how to generate Kerberos keytab file, see Generating Kerberos keytab file for Navigator Encrypt.

Registering Cloudera Navigator Encrypt

Prerequisites

Functioning Ranger KMS

After installing Navigator Encrypt on a host, you must register the host with Ranger KMS.

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 Ranger KMS

After installing Navigator Encrypt on a host, you must register the host with Ranger KMS 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://RANGERKMS01.EXAMPLE.COM:9494 --p
assive-server=https://RANGERKMS02.EXAMPLE.COM:9494  --skip-ssl-check --kms -
m 'https
```

Table 1: Registration Options - Ranger KMS

Command Option	Explanation
--server=URL	Specify the Key Manager active server url.
--passive-server=URL	Specify the Key Manager passive server url.
--skip-ssl-check	Skip the SSL certificate check if this option is passed.
--kms	The Key Manager Server is Ranger KMS.
--scheme "<SCHEME>"	Ranger KMS scheme that Navigator Encrypt uses for public key operations. Specify "http" or "https".

Updating Ranger KMS Ports

The navencrypt register command does not provide the ability to change the ports for existing registrations. If the Ranger KMS ports are changed, you must update ztrustee.conf with the new port and scheme parameters (HKP_ PORT and HKP_SCHEME, respectively).

If the Ranger KMS is updated to use port 9494 for both key storage and public key operations, you must update ztru stee.conf as follows :

```
Example :
            {
            "LOCAL_FINGERPRINT":      "2048R/3C818490979EDED9915AEC34E03D413CB
2ABE631",
            "REMOTES":        {
            "dktsmig-2.vpc.cloudera.com":  {
            "REMOTE_FINGERPRINT":     "4096R/391A464B50C40BDFAE7C70D3975DD0B1
49B2680E",
            "REMOTE_SERVER":            "https://dktsmig-2.vpc.cloudera.com:9
494",
            "HKP_PORT":       9494,
            "HKP_SCHEME":     "https",
            "DEFAULT":        true,
            "HKP_TIMEOUT":    60,
            "REMOTE_SERVERS":          ["https://dktsmig-2.vpc.cloudera.com:
9494", "https://dktsmig-1.vpc.cloudera.com:9494"],
            "SSL_INSECURE": true,
            "PROTOCOL":       "json-cleartext",
            "IS_KMS":         true
            }
            }
            }
```

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 14 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
navencrypt	Manages, updates, and verifies your data.
navencrypt-prepare	Prepares your system for encryption by creating mount-points and specifying storage.
navencrypt-prepare --undo	Removes a mountpoint that is no longer in use.
navencrypt-move	Encrypts/decrypts your data to/from the encrypted file system.
navencrypt-profile	Generates process profile information in JSON format.
navencrypt-module-setup	Builds or rebuilds the Navigator Encrypt kernel module.
navencrypt-gen-keytab	Generates keytab file used with Ranger KMS.
navencrypt-convert <devices> Where <devices> refers to a list of block devices under Navigator Encrypt control.	Ensures that the device supplied is eligible to be converted to LUKS 2, and then converts it.

LUKS2 support

LUKS implements a platform-independent standard on-disk format for use in various tools. Version 2 of LUKS supports various features. For details, see the LUKS documentation. Navigator Encrypt defaults to use whichever version of LUKS the platform uses, for example, on RHEL 9 it is LUKS2. To view which drives are using which version of LUKS, execute the following command:

navencrypt-mount status

When upgrading Navigator Encrypt, Navigator Encrypt does nothing to the LUKS version, but supplies a tool, called navencrypt-convert, to upgrade a LUKS1 version to LUKS2 version. This command ensures that the device supplied is eligible to be converted to LUKS2. This command also ensures that Navigator Encrypt is not running, as drives cannot be mounted while being converted.
Usage:

navencrypt-convert <devices>

Where, <devices> refers to a list of block devices under Navigator Encrypt control. Device names should be in the form /dev/sdx, /dev/sdyy, and so on. Loop devices are not supported.

Preparing for encryption

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 with the Key Manager Service
• 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 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 run 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 12 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 nave ncrypt-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 10), 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.

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 Key Manager Service(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/sda1                                          ... OK
```

After you have successfully prepared a client for encryption, you can encrypt and decrypt data using the commands described in

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=/DMCRYPT/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-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 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_optio
ns.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 (/dmcrypt/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 /dmcrypt/storage /dev/loop0 /dmcrypt/mountpoint
Type MASTER passphrase:

Encryption Type:  dmCrypt (LUKS)
Cipher:           aes
Key Size:         256
Random Interface: OpenSSL
Filesystem:       ext4
Options:

Verifying MASTER key against Key Manager Service(wait a moment)   ... OK
Generation Encryption Keys with OpenSSL                   ... OK
Assigning '/dev/loop0'->'/dmcrypt/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>
/dmcrypt/mountpoint      /dev/loop0        luks     key=keytrustee,nav_datastor
e='/dmcrypt/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 11.

Pass-through mount options for navencrypt-prepare

Navigator Encrypt 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). 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 Key Manager Service(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 dmcrypt 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 dmcrypt (rw,dmcrypt_sig=6de3db1e87077adb,ecryptfs_un
link_sigs,noauto,\
```

```
dmcrypt_cipher=aes,dmcrypt_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

Learn how to encrypt and decrypt data using 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 9 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.
- 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-m ovecommand 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> <enc
rypted_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
navencrypt-move	Main command interface for all actions that require moving data either to or from the encrypted file system. For more information see the navencrypt-move man page (man navencrypt-move).
encrypt	Identifies the cryptographic operation, in this case, encrypting data. The decrypt option is described later in *Decrypting Data*. **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 --per-file 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.
@<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 25 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
```

Managing Navigator Encrypt 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 /var/lib/mysql, you could use the category @mysql to indicate that this rule is granting access to a process on the MySQL data. See Listing Categories on page 25 for instructions on viewing existing categories.
PATH	Specifies the rights permissions of a specific path. For example: *, www/*.htaccess. Omit the leading slash (/).
PROCESS	Specifies the process or command name for the rule.
PARAMETERS	Tells the process the parent-child process to be executed: --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. --children defines for Navigator Encrypt which child processes to allow that are executed by a process/script. Example: --shell=/bin/bash, --children=/bin/df,/bin/ls

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.

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 5: 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 /var/lib/mysql, you could use the category @mysql to indicate that this rule is granting access to a process on the MySQL data. See Listing Categories on page 25 for instructions on viewing existing categories.
PATH	Specifies the rights permissions of a specific path. For example: *, www/*.htaccess. Omit the leading slash (/).
PROCESS	Specifies the process or command name for the rule.
PARAMETERS	Tells the process the parent-child process to be executed:
	--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.
	--children defines for Navigator Encrypt which child processes to allow that are executed by a process/script.
	Example: --shell=/bin/bash, --children=/bin/df,/bin/ls

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/ls
```

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 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*. 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* for information on determining line numbers.

Deleting ACL Rules by Profile

See *ACL Profile Rules* 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/m
ysqld
```

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 run the script. Supported shells are /usr/bin/bash, /bin/bash, /usr/bin/dash, and /bin/bash

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, run 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/NetwrkManager/NetworkManager.
pid",
"gid":"0"
"cwd":"/",
"fd0":"/dev/null",
"fd1":"/dev/null",
"fd2":"/dev/null"
}
```

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

```
ps -p <pid_of_process> -o euid,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"}
```

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 filesystem 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 `navencrypt acl --update` command. 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 OSs, 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.

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 Ranger KMS. If these files are accidentally deleted, they can be restored from Ranger KMS 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 versions, 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 Key Manager Service, 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
```

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:
  /dmcrypt-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 versions, use systemctl [start|stop|status|restart]   navencrypt-mount.

> **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 run navencrypt-mount   status with some inactive mount points:

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

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

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

The following example shows how to run 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 run 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.

When upgrading to a new version of the Linux Kernel, the kernel-headers and kernel-devel packages need to be updated and the navencrypt-module-setup command needs to be run.

Navigator Encrypt Configuration Directory Structure

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

```
$ tree /etc/navencrypt/
/etc/navencrypt/
```

```
### control -> /etc/navencrypt/jSpi9SM65xUIIhrau1Nn8ZXmQhrrQ9e363EUz8HKiRs
### jSpi9SM65xUIIhrau1Nn8ZXmQhrrQ9e363EUz8HKiRs
### 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 Key Manager Service 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 Key Manager Service 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 the Key Manager Service 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.

Generating Kerberos keytab file for Navigator Encrypt

Learn how to generate Navigator Encrypt keytab file by using the navencrypt-gen-keytab script.

About this task

In environments utilizing Navigator Encrypt alongside Ranger KMS, a script named navencrypt-gen-keytab facilitates the creation of Kerberos keytab files. This script interacts with Cloudera Manager to generate the necessary keytab file, which is pivotal for secure authentication within the Navigator Encrypt framework.

The navencrypt-gen-keytab script is designed to integrate seamlessly with Cloudera Manager, leveraging its capabilities to streamline the keytab file creation process.

Before you begin

- You must have access to Cloudera Manager instance (<cm-url>).

  Example Cloudera Manager URL format: https://xxyyz-zyyx8-1.vpc.cloudera.com:XXXX
- You must have valid credentials (<user:pw>) with appropriate permissions to interact with Cloudera Manager.

  Example credential formats: example-username:example-password

Procedure

1. SSH to one of the hosts where Navigator Encrypt is deployed.
2. Execute the following script:

```
navencrypt-gen-keytab <cm-url> <user:pw>
```

Replace <cm-url> with the URL of your Cloudera Manager instance and <user:pw> with the appropriate username and password combination.

> **Note:** If you choose not to supply the password as part of the <user:pw> argument, the script prompts you twice to enter it securely.

Upon successful execution, the navencrypt.keytab keytab file is created and stored in the /etc/navencrypt/ directory.

3. To verify the creation and content of the keytab file, use the `cat` command:

```
cat /etc/navencrypt/navencrypt.keytab
```

This command displays the content of the generated keytab file, confirming its successful creation.

Navigator Encrypt installed using Cloudera Manager (Tech Preview)

In Cloudera Runtime 7.3.2.0 and higher releases, the new Navigator Encrypt parcel centralizes the entire lifecycle management within Cloudera Manager. Previously, Navigator Encrypt required manual, host-by-host RPM installation and configuration. Administrators can now manage the installation, configuration, and upgrade of Navigator Encrypt across all cluster hosts directly from the Cloudera Manager interface. This new approach provides a significant improvement in usability, streamlining ongoing maintenance and lifecycle management.

Downloading, distributing, and activating Navigator Encrypt parcel

Supported OSs

> Red Hat, Ubuntu, SLES

Prerequisites on required and all hosts

> The following table describes the platforms and required commands to install different parcels needed for Navigator Encrypt:

Platform	Commands

RHEL	• To install the dkms component:
	`sudo yum install dkms`
	• To install the cryptsetup:
	`sudo yum install cryptsetup`
	• To install headers from the kernel:
	`sudo yum install kernel-devel-"$(uname -r)"`

SLES	• To install the cryptsetup: `sudo zypper install cryptsetup`

Ubuntu	• To install the dkms component:
	`sudo apt install dkms`
	• To install the cryptsetup:
	`sudo apt install cryptsetup`
	• To install headers from the kernel:
	`sudo apt install linux-headers-"$(uname -r)"`

Installing Navigator Encrypt using Cloudera Manager

Review the prerequisites and learn how to install Navigator Encrypt using Cloudera Manager.

Before you begin

- Ranger KMS must be installed and running.
- Navigator Encrypt RPM files must not be installed.

Procedure

1. In Cloudera Manager, select the Add Service option from the Actions drop-down menu for your cluster.

   The **Add Service to Cluster** wizard appears.

2. Select NavEncrypt, and click Continue.

   The **Assign Roles** page appears.

3. Select hosts, and click OK.

4. Click Continue.

   The **Review Changes** page appears.

5. Enter the following details, and click Continue:

Property name	Description
Master Password	Set a master password. The password must be between 15 to 32 characters.
Ranger KMS Host	The host where Ranger KMS is running.

   The installation process starts and displays its progress.

6. After the installation process completes, click Continue.

7. Click Finish.

   Navigator Encrypt is installed, and you can view it under Cloudera Manager Cluster .

8. Click the Actions option beside Navigator Encrypt service, and click Start.

   The **Start** wizard displays the progress.

   The start process initiates the Navigator Encrypt Mount service, Navigator Encrypt Health Monitor, along with Navigator Encrypt.

9. After the start process completes, click Close.

Monitoring Navigator Encrypt health using Cloudera Manager

Verify whether Navigator Encrypt is healthy using Cloudera Manager.

Procedure

Select the Instances option from the Actions drop-down menu in the Navigator Encrypt service.

You can verify whether Navigator Encrypt is running on all the hosts it is installed.

Configuring Navigator Encrypt using Cloudera Manager

Learn about configuring Navigator Encrypt using Cloudera Manager, including the mandatory configurations of mount points, encryption rules, and ACL rules.

Adding Navigator Encrypt mount points

Learn about configuring Navigator Encrypt mount points using Cloudera Manager.

Procedure

1. Go to Cloudera Manager Navigator Encrypt Configuration .

2. Search for the NavEncrypt Mount Points property.

3. Add the block device and mount point, separated by a space.

> **Note:** The path must be absolute and the device and mount point must exist. Otherwise, the operation fails.

4. Click Save Changes.

5. Select Actions Create the NavEncrypt Mount Points .

The **Create the NavEncrypt Mount Points** wizard appears.

6. Confirm the action by clicking Create the NavEncrypt Mount Points.

The process starts.

7. Click Close after the process is complete.

Adding encryption rules

Learn about configuring encryption rules for Navigator Encrypt using Cloudera Manager.

Procedure

1. Go to Cloudera Manager Navigator Encrypt Configuration .

2. Search for the Encryption Rules property.

3. Add the category, directory, or file, and the encrypted mount point, separated by spaces.

> **Note:** The directory or file must be absolute. Otherwise, the operation fails.

4. Click Save Changes.

5. Select Actions Add NavEncrypt encrypt rules and encrypt files .

6. Confirm the action by clicking Add NavEncrypt encrypt rules and encrypt files.

The process starts.

7. Click Close after the process is complete.

Adding ACL rules

Learn about configuring ACL rules for Navigator Encrypt using Cloudera Manager.

Procedure

1. Go to Cloudera Manager Navigator Encrypt Configuration .

2. Search for the ACL Rules property.

3. Add the type of rule (either ALLOW or DENY), @<category>, the access path, and the absolute path of a process or command.
navencrypt acl --add --rule="ALLOW @kudu *   /var/lib/kudu/tserver"

4. Click Save Changes.

5. Select Actions Add NavEncrypt ACL Rules .

6. Confirm the action by clicking Add NavEncrypt ACL Rules.

The process starts.

7. Click Close after the process is complete.

Stopping Navigator Encrypt using Cloudera Manager

Learn about stopping Navigator Encrypt using Cloudera Manager.

Procedure

1. Go to Cloudera Manager Cluster Navigator Encrypt .
2. Click Actions Stop .

   The **Stop NavEncrypt** wizard appears.
3. Confirm the action by clicking Stop.

   The process starts.
4. Click Close after the process is complete.

Troubleshooting Navigator Encrypt issues

Check the Navigator Encrypt log files for errors by using either the UI or the CLI. View some error message examples from the log files.

You can check the log files for errors and troubleshoot the issues.

Checking log files using the UI

1. Go to Cloudera Manager Navigator Encrypt .
2. Go to the Instances tab.
3. Select the NavEncrypt instance whose log file you want to check.
4. Click Log Files Role Log File .

   The **Log Details** page is displayed.
5. Check for error messages and troubleshoot accordingly.

Check log files using the CLI

1. Bring the log files in the command prompt by running the following command:

   ```
   cd /var/log/navencrypt/
   ls
   cat navencrypt.log
   ```
2. Check for error messages and troubleshoot accordingly.

Example error messages

-
  ```
  E20250917 10.34.53.555096 0 NA] app=navencrypt-manager action=management
   Cannot contact any
  KDC for realm 'VPC.CLOUDERA.COM'
  ```

-
  ```
  E20250923 17:22:50.359286 0 N/A] app=navencrypt action=register Cannot s
  end deposit to key manager: Unauthorized
  ```