HashiCorp Vault Enterprise

nShield® HSM Integration Guide

2025-04-23

Table of Contents

Chapter 1. Introduction

HashiCorp Vault Enterprise (referred to as Vault in this guide) supports the creation/storage of keys within Hardware Security Modules (HSMs). Entrust nShield HSMs (referred to as HSM in this guide) provide FIPS or Common Criteria certified solutions to securely generate, encrypt, and decrypt the keys which provide the root of trust for the Vault protection mechanism.

This guide describes how to integrate Vault with an HSM to:

- Offload select PKI operations to the HSM.
- Generate new PKI key pairs and certificates.
- Verify and sign certificate workflows.

1.1. Product configurations

Entrust has successfully tested nShield HSM integration with Vault in the following configurations:

Product	Version
HashiCorp Vault Enterprise	v1.19.0 Enterprise HSM
Base OS	Red Hat Enterprise 9.5

1.1.1. Supported nShield features

Entrust has successfully tested nShield HSM integration with the following features:

Feature	Support
Softcards	Yes
Module Only Key	Yes
OCS cards	Yes
nSaaS	Supported but not tested

1.1.2. Supported nShield hardware and software versions

Entrust has successfully tested with the following nShield HSM hardware and software versions:

1.1.2.1. nShield 5c

Security World Software	Firmware	Netimage	OCS	Softcard	Module
13.6.8	13.4.5 (FIPS 140-3 certified)	13.6.7	✓	✓	✓

1.1.2.2. Connect XC

Security World Software	Firmware	Netimage	OCS	Softcard	Module
13.6.8	12.72.3 (FIPS 140-2 certified)	13.6.7	✓	✓	✓
13.4.8	12.60.15 (CC certified)	13.3.2	✓	✓	✓

1.1.3. Supported nShield key types

Entrust has successfully tested with the following Vault managed keys:

- RSA
- ECDSA

1.2. Requirements

- Access to the Entrust TrustedCare Portal.
- Access to HashiCorp Vault Enterprise Module license from your HashiCorp sales representative.
- A dedicated Linux server.

- Network environment with usable port 9004 for the HSM and 8200 for Vault.

Familiarize yourself with the nShield Documentation.

- The importance of a correct quorum for the Administrator Card Set (ACS).
- Whether Operator Card Set (OCS) protection or Softcard protection is required.
- If OCS protection is to be used, a 1-of-N quorum must be used.
- Whether your Security World must comply with FIPS 140 Level 3 or Common Criteria standards. If using FIPS 140 Level 3, it is advisable to create an OCS for FIPS authorization. The OCS can also provide key protection for the Vault master key. For more information see FIPS 140 Level 3 compliance.
- Whether to instantiate the Security World as recoverable or not.

Chapter 2. Install and configure the Entrust nShield HSM

2.1. Install the Entrust nShield HSM

Install the nShield Connect HSM locally, remotely, or remotely via the serial console. Condensed instructions are available in the following Entrust nShield Support articles.

- How To: Locally Set up a new or replacement nShield Connect.
- How To: Remotely Setup a new or replacement nShield Connect.
- How To: Remotely Setup a new or replacement nShield Connect XC Serial Console Model.

For detailed instructions see the nShield v13.6.8 Hardware Install and Setup Guides.

2.2. Install the nShield Security World Software and create the Security World

1. Install the Security World software. For detailed instructions see the nShield Security World Software v13.6.8 Installation Guide.

   

   If using the older Security World version v12.81.2, install TAC-955 hot fix.

2. Add the Security World utilities path to the system path. This path is typically `/opt/nfast/bin`:

   ```
   # sudo vi /etc/profile.d/nfast.sh
   ```

   Add the following info to `nfast.sh` and save:

   ```
   # Entrust Security World path variable
   export PATH=$PATH:/opt/nfast/bin
   ```

3. Open firewall port 9004 for the Entrust nShield HSM connections:

   ```
   # sudo firewall-cmd --permanent --add-port=9004/tcp
   # sudo firewall-cmd --reload
   ```

4. If using remote administration, open firewall port 9005 for the Entrust nShield Trusted Verification Device (TVD):
5. Open a command window and run the following to confirm the HSM is **operational**:

```
# enquiry
Server:
 enquiry reply flags  none
 enquiry reply level  Six
 serial number        8FE1-B519-C5AA
 mode                 operational
...
Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        8FE1-B519-C5AA
 mode                 operational
 ...
```

6. Create your Security World if one does not already exist or copy an existing one. Follow your organization's security policy for this. For more information see Create a new Security World.

ACS cards cannot be duplicated after the Security World is created. You may want to create extras in case of a card failure or a lost card.

7. Confirm the Security World is **Usable**:

```
# nfkminfo
World
 generation  2
 state       0x3737000c Initialised Usable ...
 ...
Module #1
 generation 2
 state      0x2 Usable
 ...
```

2.3. Select the protection method

OCS, Softcard, or Module protection can be used to authorize access to the keys protected by the HSM. Typically, an organization's security policies dictate the use of one or the others.

- Operator Cards Set (OCS) are smartcards that are presented to the physical smartcard reader of an HSM. For more information on OCS use, properties, and k-of-N values, see Operator Card Sets (OCS).

- Softcards are logical tokens (passphrases) that protect the key and authorize its use. For more information on Softcards use see Softcards.
- Module protection has no passphrase.

Follow your organization's security policy to select an authorization access method.

1. Create file `/opt/nfast/cknfastrc` containing the nShield PKCS #11 library environment variables per the selection above.

   For example:

   ```
   # Enable Softcard protection
   CKNFAST_LOADSHARING=1

   # Enable Module protection
   CKNFAST_FAKE_ACCELERATOR_LOGIN=1

   # Needed for managed key
   CKNFAST_OVERRIDE_SECURITY_ASSURANCES=wrapping_crypt

   # OCS Preload file location and card set state
   NFAST_NFKM_TOKENSFILE=/opt/nfast/preloadtoken
   CKNFAST_NONREMOVABLE=1

   # PKCS #11 log level and file location
   CKNFAST_DEBUG=10
   CKNFAST_DEBUGFILE=/opt/nfast/log/pkcs11.log
   ```

2. Change ownership of `/opt/nfast/cknfastrc` to `nfast`.

   ```
   # ls -al /opt/nfast/cknfastrc
   -rw-rw-rw-. 1 root root 324 Apr  3 16:12 /opt/nfast/cknfastrc

   # chown nfast:nfast /opt/nfast/cknfastrc

   # ls -al /opt/nfast/cknfastrc
   -rw-rw-rw-. 1 nfast nfast 324 Apr  3 16:12 /opt/nfast/cknfastrc
   ```

2.4. Create the OCS

1. Edit file `/opt/nfast/kmdata/config/cardlist` adding the serial number of the card(s) to be presented, or the wildcard "*".
2. Open a command window as an administrator.
3. Run the `createocs` command as described below, entering a passphrase at the prompt.

   Follow your organization's security policy for the values K/N. Use the same passphrase for all the OCS cards in the set (one for each person with access

privilege, plus the spares). In this example note that `slot 2`, remote via a TVD, is used to present the card.

Vault requires k = 1 whereas N can be up to, but not exceeding, 64.

After an OCS card set has been created, the cards cannot be duplicated. You may want to create extras in case of a card failure or a lost card.

The preload utility loads OCS onto the HSM. This feature makes the OCS available for use after been physically removed from the HSM for safe storage or other reasons. Add the **-p** (persistent) option to the command below to have authentication after the OCS card has been removed from the HSM front panel slot, or from the TVD.

```
# createocs -m1 -s2 -N testOCS -Q 1/1

FIPS 140-2 level 3 auth obtained.

Creating Cardset:
 Module 1: 0 cards of 1 written
 Module 1 slot 0: Admin Card #1
 Module 1 slot 2: empty
 Module 1 slot 3: empty
 Module 1 slot 2: blank cardSteps:

 Module 1 slot 2:- passphrase specified - writing card
Card writing complete.

cardset created; hkltu = a165a26f929841fe9ff2acdf4bb6141c1f1a2eed
```

The authentication provided by the OCS as shown in the command line above is non-persistent and only available while the OCS card is inserted in the HSM front panel slot, or the TVD.

4. Verify the OCS created:

```
# nfkminfo -c
Cardset list - 2 cardsets:  (P)ersistent/(N)ot, (R)emoteable/(L)ocal-only
 Operator logical token hash               k/n timeout  name
 edb3d45a28e5a6b22b033684ce589d9e198272c2  1/5  none-NL testOCS
```

The `rocs` utility also shows the OCS created:

```
# rocs
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
```

```
rocs> list cardset
No. Name                    Keys (recov) Sharing
  1 testOCS                 0 (0)        1 of 5
rocs> quit
```

2.5. Create the Softcard

1. Enable Softcard protection as described in Select the protection method.
2. Open a command window as an administrator.
3. Run the following command, and enter a passphrase at the prompt:

```
# ppmk -n testSC

Enter new pass phrase:
Enter new pass phrase again:
New softcard created: HKLTU d9414ed688c6405aab675471d3722f8c70f5d864
```

4. Verify the Softcard created:

```
# nfkminfo -s
SoftCard summary - 1 softcards:
 Operator logical token hash               name
 925f67e72ea3c354cae4e6797bde3753d24e7744  testSC
```

The `rocs` utility also shows the OCS and Softcard created:

```
# rocs
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs> list cards
No. Name                    Keys (recov) Sharing
  1 testOCS                 0 (0)        1 of 5
  2 testSC                  0 (0)        (softcard)
rocs> quit
```

Chapter 3. Create the Vault encryption and HMAC keys

The Vault encryption and HMAC keys can be protected with an OCS, Softcard, or Module. Key generation with all three protection methods is shown below. Choose the one that applies to you.

3.1. Verify the PKCS #11 library is available

1. Present the OCS if using OCS protection.
2. Execute the `ckcheckinst` command to test the library. Enter the slot number corresponding to the protection method used. Enter the OCS or Softcard passphrase when prompted.

```
# ckcheckinst
PKCS#11 library interface version 2.40
                           flags 0
                  manufacturerID "nCipher Corp. Ltd               "
               libraryDescription "nCipher PKCS#11 13.6.8-209-a5bd9"
           implementation version 13.06
         Loadsharing and Failover enabled

Slot  Status            Label
====  ======            =====
   0  Fixed token       "loadshared accelerator          "
   1  Operator card     "testOCS                         "
   2  No token present
   3  Soft token        "testSC                          "

Select slot number to run library test or 'R'etry or to 'E'xit: 1
Using slot number 1.

Please enter the passphrase for this token (No echo set).
Passphrase:

Test                      Pass/Failed
----                      -----------

1 Generate RSA key pair   Pass
2 Generate DSA key pair   Pass
3 Encryption/Decryption   Pass
4 Signing/Verification    Pass

Deleting test keys         ok

PKCS#11 library test successful.
```

3.2. Create the keys using OCS protection

To create OCS protected keys in a FIPS 140-3 world, the OCS must be presented

via the card reader in the HSM front panel. An alternative is to present the OCS remotely via the TVD while mapping the TVD slot to slot 0.

For example:

```
# cat /opt/nfast/kmdata/hsm-8FE1-B519-C5AA/config/config
syntax-version=1
...
[slot_mapping]
...
#
# ESN of the module on which slot 0 will be remapped with another.
#  esn=ESN
#
# Slot to exchange with slot 0. Setting this value to 0 means do
# nothing.(default=0)
#  slot=INT
esn=8FE1-B519-C5AA
slot=2
...
```

1. Create the Vault encryption key `vault_v1_ocs`:

```
# generatekey --generate --batch -m1 -s0 pkcs11 protect=token cardset=testOCS plainname=vault_v1_ocs
type=AES size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
 slot         Slot to read cards from    0
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v1_ocs
 nvram        Blob in NVRAM (needs ACS)  no

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: `testOCS' #2
 Module 1 slot 2: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-
d8ed4d73cfd90e4bf5c41c63459019e3382c1bbc
```

2. Create the Vault HMAC key `vault_hmac_v1_ocs`:

```
# generatekey --generate --batch -m1 -s0 pkcs11 protect=token cardset=testOCS plainname=vault_hmac_v1_ocs
type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               token
```

```
 slot         Slot to read cards from    0
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v1_ocs
 nvram        Blob in NVRAM (needs ACS)  no

Loading `testOCS':
 Module 1: 0 cards of 1 read
 Module 1 slot 0: `testOCS' #2
 Module 1 slot 2: empty
 Module 1 slot 3: empty
 Module 1 slot 4: empty
 Module 1 slot 5: empty
 Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-
3787adc4f5b499040058dcfdc0ad643e43817024
```

3.3. Create the keys using Softcard protection

1. Create the Vault encryption key `vault_v1_sc`:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=testSC plainname=vault_v1_sc type=AES
size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
 softcard     Soft card to protect key   testSC
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v1_sc
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `testSC':

Please wait........

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-
408577c897946d66019d59cff232f989c57d6600
```

2. Create the Vault HMAC key `vault_hmac_v1_sc`:

```
# generatekey --generate --batch -m1 pkcs11 protect=softcard softcard=testSC plainname=vault_hmac_v1_sc
type=HMACSHA256 size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               softcard
 softcard     Soft card to protect key   testSC
 recovery     Key recovery               yes
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
```

```
 size         Key size                  256
 plainname    Key name                  vault_hmac_v1_sc
 nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `testSC':

Please wait........

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-
624d9f8e4e56dc281ce58b3b53dfb88772c9e88d
```

3.4. Create the keys using Module protection

1. Create the Vault encryption key `vault_v1_m`:

```
# generatekey --generate --batch -m1 pkcs11 protect=module plainname=vault_v1_m type=AES size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               module
 verify       Verify security of key     yes
 type         Key type                   AES
 size         Key size                   256
 plainname    Key name                   vault_v1_m
 nvram        Blob in NVRAM (needs ACS)  no

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ua96a3d9e6dd69fba7c5a4df8a26f5dc4ccb2f5f79
```

2. Create the Vault HMAC key `vault_hmac_v1_m`:

```
# generatekey --generate --batch -m1 pkcs11 protect=module plainname=vault_hmac_v1_m type=HMACSHA256
size=256
key generation parameters:
 operation    Operation to perform       generate
 application  Application                pkcs11
 protect      Protected by               module
 verify       Verify security of key     yes
 type         Key type                   HMACSHA256
 size         Key size                   256
 plainname    Key name                   vault_hmac_v1_m
 nvram        Blob in NVRAM (needs ACS)  no

Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_ua55376b64e163268e15e25670b0bab7f595d7a7c3
```

3.5. Verify the keys created

1. Verify the keys created using the `rocs` utility:

```
# rocs
`rocs' key recovery tool
Useful commands: `help', `help intro', `quit'.
rocs> list keys
```

```
  No. Name                    App        Protected by
    1 vault_v1_m              pkcs11     module
    2 vault_v1_ocs            pkcs11     testOCS
    3 vault_hmac_v1_ocs       pkcs11     testOCS
    4 vault_v1_sc             pkcs11     testSC (testSC)
    5 vault_hmac_v1_sc        pkcs11     testSC (testSC)
    6 vault_hmac_v1_m         pkcs11     module
rocs> exit
```

2. Verify the keys created using the `nfkminfo` utility:

```
# nfkminfo -l

Keys with module protection:
 key_pkcs11_ua55376b64e163268e15e25670b0bab7f595d7a7c3 `vault_hmac_v1_m'
 key_pkcs11_ua96a3d9e6dd69fba7c5a4df8a26f5dc4ccb2f5f79 `vault_v1_m'

Keys protected by softcards:
 key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-408577c897946d66019d59cff232f989c57d6600
`vault_v1_sc'
 key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-624d9f8e4e56dc281ce58b3b53dfb88772c9e88d
`vault_hmac_v1_sc'

Keys protected by cardsets:
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-3787adc4f5b499040058dcfdc0ad643e43817024
`vault_hmac_v1_ocs'
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-d8ed4d73cfd90e4bf5c41c63459019e3382c1bbc
`vault_v1_ocs'
```

3.6. Find the slot value for each protection method

Each protection method is loaded to a virtual slot of the HSM. The decimal value of this slot will be needed further down to configure Vault.

1. Run the `cklist` utility. Notice the lines below.

```
# cklist
Listing contents of slot 0
 (token label "loadshared accelerator          ")
...
Listing contents of slot 1
 (token label "testOCS                         ")
...
Skipping slot 2 (not present)
...
Listing contents of slot 3
 (token label "testSC                          ")
```

`loadshared accelerator`

: Module protection.

`testOCS`

: The name given to the OCS created in section Create the OCS.

`testSC`

The name given to the Softcard token created in section Create the Softcard.

2. Search file `/opt/nfast/log/pkcs11.log` for **pSlotList**. Notice the hex value for each slot. For example:

```
...
2025-04-03 17:23:26 [6544]: pkcs11: 00000000 <    pSlotList[0] 0x2D622495
2025-04-03 17:23:26 [6544]: pkcs11: 00000000 <    pSlotList[1] 0x2D622496
2025-04-03 17:23:26 [6544]: pkcs11: 00000000 <    pSlotList[2] 0x2D622497
2025-04-03 17:23:26 [6544]: pkcs11: 00000000 <    pSlotList[3] 0x2D622498
...
```

3. Convert the **pSlotList** values to decimal:

Protection Method	Slot Number	Value (Hex)	Value (Decimal)
Module	0	0x2D622495	761406613
OCS	1	0x2D622496	761406614
	2	0x2D622497	761406615
Softcards	3	0x2D622498	761406616

Save the decimal values.

Adding or deleting Softcard tokens, or adding or deleting OCS, or adding or deleting Modules keys will change the values above. Redo the step to find the new values if necessary.

Chapter 4. Install Vault

4.1. System preparation

1. Open the following firewall ports for incoming Vault connections:

```
# sudo firewall-cmd --permanent --add-port=8200/tcp
# sudo firewall-cmd --permanent --add-port=8201/tcp
# sudo firewall-cmd --reload
```

2. Install `open-vm-tools`:

```
# sudo yum install open-vm-tools unzip opensc
```

4.2. Create Vault user and group

1. Create the Vault group:

```
# sudo groupadd --system vault
```

2. Create the Vault user:

```
# sudo useradd --system --shell /sbin/nologin --gid vault vault
```

3. Add the Vault user to the nShield **nfast** group:

```
# sudo usermod --append --groups nfast vault
```

4.3. Install Vault

1. Download the Vault package from HashiCorp at https://releases.hashicorp.com/vault/, ensuring that it is the binary file for Enterprise with HSM support:

```
# cd Downloads

# wget https://releases.hashicorp.com/vault/1.19.0+ent.hsm/vault_1.19.0+ent.hsm_linux_amd64.zip
--2025-04-04 10:56:18--
https://releases.hashicorp.com/vault/1.19.0+ent.hsm/vault_1.19.0+ent.hsm_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 18.160.181.50, 18.160.181.55, 18.160.181.20,
...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|18.160.181.50|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 165452931 (158M) [application/zip]
Saving to: 'vault_1.19.0+ent.hsm_linux_amd64.zip'

vault_1.19.0+ent.hsm_linux_amd6 100%[=================================================>] 157.79M
27.7MB/s    in 6.8s

2025-04-04 10:56:34 (23.3 MB/s) - 'vault_1.19.0+ent.hsm_linux_amd64.zip' saved [165452931/165452931]
```

2. Unzip the binary file and extract it to the working directory on the host machine, for example `/usr/local/bin`. There should only be a single binary file named `vault`.

```
# unzip vault_1.19.0+ent.hsm_linux_amd64.zip -d /usr/local/bin
Archive:  vault_1.19.0+ent.hsm_linux_amd64.zip
replace /usr/local/bin/EULA.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: /usr/local/bin/EULA.txt
replace /usr/local/bin/vault? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: /usr/local/bin/vault
replace /usr/local/bin/TermsOfEvaluation.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: /usr/local/bin/TermsOfEvaluation.txt
```

3. Set Vault permissions:

```
# sudo chmod 755 /usr/local/bin/vault
# sudo setcap cap_ipc_lock=+ep /usr/local/bin/vault
# ls -la /usr/local/bin/vault
-rwxr-xr-x. 1 root root 397524552 Sep 22 17:22 /usr/local/bin/vault
```

4. Add the Vault binary file to the path:

```
# sudo vi /etc/profile.d/vault.sh
```

   Add the following information to `vault.sh` and restart the system. The `VAULT_ADDR` variable allows Vault to be accessed from a web browser via the web user interface (web UI).

```
# HashiCorp Vault Enterprise path variable
export PATH="$PATH:/usr/local/bin"
export VAULT_ADDR=http://127.0.0.1:8200
```

5. Create the Vault data directories:

```
# sudo mkdir --parents /opt/vault/data
# sudo mkdir --parents /opt/vault/data/hsm
# sudo mkdir --parents /opt/vault/logs
# sudo chmod --recursive 750 /opt/vault
# sudo chown --recursive vault:vault /opt/vault
```

6. Reboot the server:

```
# reboot
```

7. Confirm the binary file is available:

```
# vault version
Vault v1.19.0+ent.hsm (838f2be3861a749e52d97bbacf275a472bec8ccb), built 2025-03-04T12:36:26Z (cgo)
```

4.4. Install Vault license

1. Create a directory for the Vault license and configuration files:

```
# sudo mkdir /etc/vault
```

2. Three options are given in the Install a HashiCorp Enterprise License page of the online documentation for enabling an enterprise license, as well as a procedure to request a trail license. For this guide, create a file containing the enterprise license key:

```
# cat /etc/vault/license.hclic
02MV4UU43B...
```

4.5. Create a configuration file

Create a `/etc/vault/config.hcl` configuration file to enable Vault to be run as a service. See also Vault commands.

An example configuration file with OCS protection is shown below. The **pin** is the **passphrase** entered when the OCS was created in section Create the OCS.

```
# PKCS#11 Seal, Entrust nShield Integration
seal "pkcs11" {
lib = "/opt/nfast/toolkits/pkcs11/libcknfast.so"
slot = "761406614"
pin = "xxxxxxx"
key_label = "vault_v1_ocs"
hmac_key_label = "vault_hmac_v1_ocs"
# Vault is commanding HSM to generate keys if these don't already exists
generate_key = true
}

# Vault listener with TLS disabled
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = true
}

# Storage
```

```
storage "raft" {
   path = "/opt/vault/data/hsm"
   node_id = "vault"
}

ui = true

# License file
license_path = "/etc/vault/license.hclic"

disable_mlock = false
api_addr = "http://127.0.0.1:8200"
cluster_addr = "https://127.0.0.1:8201"

# Managed Key Library
kms_library "pkcs11" {
name = "hsm1" # This can be re-named to anything you like
library = "/opt/nfast/toolkits/pkcs11/libcknfast.so" #PKCS11 Library Location
}
```

In this example:

- The **slot** and **pin** parameters will change according to the protection selected. See section Find the slot value for each protection method.
- The entropy seal mode is set to augmentation. This leverages the HSM for augmenting system entropy via the PKCS #11 protocol.
- The seal wrap is enabled. By enabling seal wrap, Vault wraps your secrets with an extra layer of encryption leveraging the HSM encryption and decryption.
- Notice the path to the license file.

4.6. Create and configure Vault directories

1. Create file `/etc/sysconfig/vault`:

```
# sudo touch /etc/sysconfig/vault
```

2. Create a service file:

```
# vi /etc/systemd/system/vault.service
```

3. Add the following information to the service file:

If deploying on a server with more than two CPUs, you may increase the value of `Environment=GOMAXPROCS` accordingly.

```
[Unit]
Description="HashiCorp Vault"
Requires=network-online.target
After=network-online.target nc_hardserver.service
```

```
ConditionFileNotEmpty=/etc/vault/config.hcl
[Service]
User=vault
Group=vault
EnvironmentFile=/etc/sysconfig/vault
ExecStart=/usr/local/bin/vault server -config=/etc/vault/config.hcl
StandardOutput=/opt/vault/logs/output.log
StandardError=/opt/vault/logs/error.log
ExecReload=/bin/kill --signal -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=5
TimeoutStopSec=30
StartLimitInterval=60
StartLimitBurst=3
AmbientCapabilities=CAP_IPC_LOCK
LimitNOFILE=65536
LimitMEMLOCK=infinity
[Install]
WantedBy=multi-user.target
```

4. If you are setting paths different from the default, edit the following lines in `/etc/systemd/system/vault.service`. Also change the location of the configuration file `config.hcl` accordingly:

```
ConditionFileNotEmpty=/etc/vault/config.hcl
EnvironmentFile=-/etc/sysconfig/vault
ExecStart=/opt/vault/bin/vault server -config=/etc/vault/config.hcl
StandardOutput=/opt/vault/logs/output.log
StandardError=/opt/vault/logs/error.log
```

4.7. Enable Vault

Enable Vault:

```
# systemctl enable vault.service
```

Chapter 5. Test the integration

5.1. Start Vault

The HSM will be accessed when starting Vault. Therefore, the OCS or Softcard is needed.

1. Start the Vault in a separate window.

   If the protection method defined in `/etc/vault/config.hcl` is OCS protection, the OCS card created in Create the OCS must be inserted in the HSM slot. Otherwise the Vault will fail to start.

```
# vault server -config=/etc/vault/config.hcl
WARNING: Request Limiter configuration is no longer supported; overriding server configuration to disable

==> Vault server configuration:

Administrative Namespace:
             Api Address: http://127.0.0.1:8200
                     Cgo: enabled
         Cluster Address: https://127.0.0.1:8201
   Environment Variables: BASH_FUNC_which%%, DBUS_SESSION_BUS_ADDRESS, DISPLAY, HISTCONTROL, HISTSIZE,
HOME, HOSTNAME, LANG, LESSOPEN, LOGNAME, LS_COLORS, MAIL, MOTD_SHOWN, OLDPWD, PATH, PWD,
SELINUX_LEVEL_REQUESTED, SELINUX_ROLE_REQUESTED, SELINUX_USE_CURRENT_RANGE, SHELL, SHLVL, SSH_AUTH_SOCK,
SSH_CLIENT, SSH_CONNECTION, SSH_TTY, TERM, USER, VAULT_ADDR, XDG_DATA_DIRS, XDG_RUNTIME_DIR,
XDG_SESSION_CLASS, XDG_SESSION_ID, XDG_SESSION_TYPE, _, which_declare
              Go Version: go1.23.6
              Listener 1: tcp (addr: "0.0.0.0:8200", cluster address: "0.0.0.0:8201",
disable_request_limiter: "false", max_request_duration: "1m30s", max_request_size: "33554432", tls:
"disabled")
               Log Level:
                   Mlock: supported: true, enabled: false
           Recovery Mode: false
                 Storage: raft (HA available)
                 Version: Vault v1.19.0+ent.hsm, built 2025-03-04T12:36:26Z
             Version Sha: 838f2be3861a749e52d97bbacf275a472bec8ccb

==> Vault server started! Log data will stream in below:
...
```

2. Initialize the Vault back in the original window:

   The `vault operator init` command returns the Recovery Key(s) and Initial Root Token. Save these.

```
# vault operator init
Recovery Key 1: L2DHbGYEvhlNHQ9rp6BIkIAHZoYtO2b6OU+wyef5Q/N0
Recovery Key 2: VEB57DW1G6hwEj+LWLdY2jQ7Ila6g8TpFZuqRB2cdUt/
Recovery Key 3: LLCx/CG40R0uFHpogmOxijaN2QwXB9ptyfA+C8Dl31YE
Recovery Key 4: 7JilYckIDit+IHgPlmPY21H5IyB1rpvCQAF4C45/9g+v
Recovery Key 5: G0OgT/LBWbAkPEadhyQUBNM7RcL6h4rwBmOVXccG4Bk8

Initial Root Token: hvs.QiiNikZWKf9gNMVMVRfTEQLO
```

```
Success! Vault is initialized

Recovery key initialized with 5 key shares and a key threshold of 3. Please
securely distribute the key shares printed above.
```

5.2. Log in from the command line

Log in to Vault using the Initial Root Token saved above:

```
# vault login hvs.QiiNikZWKf9gNMVMVRfTEQLO
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                hvs.QiiNikZWKf9gNMVMVRfTEQLO
token_accessor       yi2yH6h7oE0U5U8sQJ2jXpXd
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

5.3. Create Managed Key In Vault

1. Create an RSA-managed key **hsm-key-ocs-rsa** in Vault **VaultKeyOCSRSA**, protected by the OCS **testOCS** in the HSM:

   ```
   # vault write /sys/managed-keys/pkcs11/hsm-key-ocs-rsa library=hsm1 slot=761406614 pin=ncipher
   key_label="VaultKeyOCSRSA" allow_generate_key=true allow_store_key=true mechanism=0x0001 key_bits=2048
   Success! Data written to: sys/managed-keys/pkcs11/hsm-key-ocs-rsa
   ```

2. Write to the HSM the new managed key **hsm-key-ocs-rsa**:

   ```
   # vault write -f /sys/managed-keys/pkcs11/hsm-key-ocs-rsa/test/sign
   Success! Data written to: sys/managed-keys/pkcs11/hsm-key-ocs-rsa/test/sign
   ```

3. Create a ECDSA managed key **hsm-key-ocs-ecdsa** in Vault labeled **VaultKeyOCSRSA**, and protected by the OCS **testOCS** in the HSM:

   ```
   # vault write /sys/managed-keys/pkcs11/hsm-key-ocs-ecdsa library=hsm1 slot=761406614 pin=ncipher
   key_label="VaultKeyOCSECDSA" allow_generate_key=true allow_store_key=true mechanism=0x1041 curve=P256
   Success! Data written to: sys/managed-keys/pkcs11/hsm-key-ocs-ecdsa
   ```

4. Write to the HSM the new managed key **hsm-key-ocs-ecdsa**:

```
# vault write -f /sys/managed-keys/pkcs11/hsm-key-ocs-ecdsa/test/sign
Success! Data written to: sys/managed-keys/pkcs11/hsm-key-ocs-ecdsa/test/sign
```

5. List all keys created in the HSM. Notice the new keys **VaultKeyOCSRSA** and **VaultKeyOCSECDSA**:

```
# nfkminfo -l

Keys with module protection:
 key_pkcs11_ua55376b64e163268e15e25670b0bab7f595d7a7c3 `vault_hmac_v1_m'
 key_pkcs11_ua96a3d9e6dd69fba7c5a4df8a26f5dc4ccb2f5f79 `vault_v1_m'

Keys protected by softcards:
 key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-408577c897946d66019d59cff232f989c57d6600
`vault_v1_sc'
 key_pkcs11_uc925f67e72ea3c354cae4e6797bde3753d24e7744-624d9f8e4e56dc281ce58b3b53dfb88772c9e88d
`vault_hmac_v1_sc'

Keys protected by cardsets:
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-12b105aa2cbb3c5f152ad8ea78e5d1d770dba9f2
`VaultKeyOCSECDSA'
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-3787adc4f5b499040058dcfdc0ad643e43817024
`vault_hmac_v1_ocs'
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-d8ed4d73cfd90e4bf5c41c63459019e3382c1bbc
`vault_v1_ocs'
 key_pkcs11_ucedb3d45a28e5a6b22b033684ce589d9e198272c2-e0e90329b06569f64a4e5c9922c27063911cda3f
`VaultKeyOCSRSA'
```

6. Enable the PKI secrets engine at the path `pki` and reference a managed key **hsm-key** stored in the HSM:

```
# vault secrets enable -path=pki -allowed-managed-keys=hsm-key pki
Success! Enabled the pki secrets engine at: pki/
```

7. Perform PKI operations as needed. See the PKI Secrets Engine page in the online documentation.

Chapter 6. Troubleshooting

Error Message	Resolution
Vault fails to start. There may not be a log file created if the vault fails to start upon executing `# systemctl start vault.service`.	Execute the following instead to get some debugging information. `# vault server -config=/etc/vault/config.hcl`.
Error: **failed to decrypt encrypted stored keys: error initializing session for decryption: error logging in to HSM: pkcs11: 0xE0: CKR_TOKEN_NOT_PRESENT**	Ensure that the Operator card is inserted in the physical slot of the nShield HSM.

Chapter 7. Vault commands

7.1. Vault commands

Task	Command
Log into Vault	`# vault login s.InitialRootToken`
Check Vault status	`# vault status`
Unseal Vault	`# vault operator unseal -address=http://127.0.0.1:8200`
Seal Vault	`# vault operator seal`

7.2. vault.service commands

Task	Command
Enable Vault Service	`# systemctl enable vault.service`
Disable Vault service	`# systemctl disable vault.service`
Start Vault service	`# systemclt start vault.service`
Stop Vault service	`# systemclt stop vault.service`
Restart Vault service	`# systemctl restart vault.service`

Chapter 8. Additional resources and related products

8.1. nShield Connect

8.2. nShield as a Service

8.3. Entrust products

8.4. nShield product documentation