NCIPHER

www.ncipher.com

# HashiCorp Vault Enterprise

Integration Guide

**Version:** 1.10

**Date: Thursday, March 12, 2020**

# Contents

# 1 Introduction

HashiCorp Vault (referred to as *Vault* in this guide) protects your organization's credentials and confidential assets and provides secure access control to them through a process of secret leasing, renewal, and revocation. nCipher Hardware Security Modules (HSMs) provide unrestricted, FIPS, or Common Criteria certified solutions to securely generate, encrypt, and decrypt the keys which form the foundation of the HashiCorp Vault protection mechanism. The nShield HSM secures the key used to seal or unseal a Vault instance. The HSM wraps the Vault master key and provides the means for automatically sealing (encrypting) and unsealing (decrypting) it. The HSM is responsible for generating three cryptographic keys:

- An initial integrity key of type HMACSHA256.

  This key will be replaced by an independently generated HMACSHA256 using the nCipher **generatekey** command see *Procedures* on page 7.

- A revised integrity key of type HMACSHA256.
- A wrapping key for the Vault master key of type Rijndael symmetric 256-bit.

## 1.1 Product configurations

We have successfully tested nCipher HSM integration with HashiCorp Vault in the following configurations:

| Operating System | HashiCorp Vault Version | nShield Support | nShield Firmware Version | Security World Software Version |
|---|---|---|---|---|
| Red Hat Enterprise Linux 7 64-bit | 1.3.2+ent.hsm [1] | Connect+<br>Connect XC<br>Solo XC | 12.60.2 | 12.60.5 [2] |

[1] The Vault application is provided either with or without HSM enhancement. You must ensure that the version of Vault used for the deployment is Enterprise Modules. This is the only version that allows the use of HSM protection.

[2] In unrestricted, in FIPS 140-2 L3 restricted, or in Common Criteria CMTS mode

## 1.2 Supported nCipher functionality

| Feature | Support | Feature | Support | Feature | Support |
|---|---|---|---|---|---|
| Key generation | Yes | 1-of-N Operator Card Set | Yes | Strict FIPS support | Yes |
| Key management | Yes | k-of-N Operator Card Set | No | Common Criteria support | Yes |

| Feature | Support | Feature | Support | Feature | Support |
|---|---|---|---|---|---|
| Key import | Yes | Softcards | Yes | Load sharing | Yes |
| Key recovery | Yes | Module-only key | No | Fail over | Yes |

## 1.3 Requirements

Before installing these products, read the associated nShield HSM *Installation Guide*, *User Guide*, and the HashiCorp Vault documentation. This guide assumes you are familiar with the following:

- 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 140-2 Level 3 or Common Criteria standards. If using FIPS Restricted mode, it is advisable to create an OCS for FIPS authorization. The OCS can also provide key protection for the Vault master key. For information about limitations on FIPS authorization, see the *Installation Guide* of the nShield HSM.
- Whether to instantiate the Security World as recoverable or not.
- Network environment setup, via correct firewall configuration with usable ports: 9004 for the HSM and 8200 for Vault.
- HashiCorp Enterprise Modules license, which is required for using Vault with Hardware Security Modules.

## 1.4 This guide

This guide describes how to integrate an nCipher HSM with HashiCorp Vault Enterprise.

## 1.5 More information

For more information about OS support, contact your HashiCorp Vault sales representative or nCipher Support.

For more information about contacting nCipher, see *Contact Us* at the end of this guide.

# 2 Procedures

To install and configure Vault with a single HSM, integration procedures include:

- Installing and configuring the HSM, Security World, and Vault.
- Configuring Vault with an OCS or with a Softcard.
- Initializing Vault.

## 2.1 Optional: Create a Vault-specific user and group

This guide has been written assuming that the integration is performed by an Administrator with **root** privileges. You may want to run Vault as a service using an unprivileged user and group. If so, create them before installing any software and log in to the system using those credentials.

1. Create the Vault-specific user, for example:

```
# useradd -r vaultusr
```

2. Supply the user account with privileges on the **Bin** directory created as part of the Vault install process:

```
# groupadd -r vault
# usermod -a -G vault vaultusr
# chown -Rv vaultusr:vault /opt/vault
# passwd vaultusr
```

3. Confirm the changes:

```
# cat /etc/group
```

4. Add this user to the **nfast** group:

```
# usermod -a -G nfast vaultusr
```

## 2.2 HSM configuration

### 2.2.1 Install the Security World software and create a Security World

1. Install and configure the Security World software. For instructions, see the *Installation Guide* and the *User Guide* for the HSM.
2. When the Software has been installed, ensure that *$NFAST_HOME* is on the *PATH*:

```
# export PATH=$PATH:/opt/nfast/bin/
```

3. Confirm that the HSM is available:

```
# enquiry
```

4. Create your Security World if one does not already exist.
5. Confirm that the Security World is **operational** and **usable** in the output of the **nfkminfo** command:

```
# nfkminfo
```

6. Create a **cknfastrc** file in **/opt/nfast/**.

```
# vi /opt/nfast/cknfastrc
```

7. Save and close the **cknfastrc** file.

## 2.2.2 Create the OCS or the Softcard

The Vault seal key can be protected with an OCS or a Softcard:

- Operator Cards are smartcards that are presented to the physical smartcard reader of a HSM. If an OCS is used, k must = 1 whereas N can be up to, but not exceed, 64. For more information on OCS use, properties, and k-of-N values, see the *User Guide* for your HSM.
- Softcards are logical tokens (passphrases) that protect they key and authorize its use.

### 2.2.2.1 Example for key protection using an OCS

This command is an example to create the OCS with example name and attributes, for 1-of-N (**1/2** for a single-HSM configuration):

```
# createocs -m1 -N HashiCorp -Q 1/2
```

### 2.2.2.2 Example for key protection using a Softcard

This procedure is an example to create a Softcard with example name and attributes.

1. Edit the **cknfastrc** file for Softcard use. Set the **CKNFAST_LOADSHARING** environment variable:

```
# vi /opt/nfast/cknfastrc
CKNFAST_LOADSHARING=1
```

2. Save and close the **cknfastrc** file.
3. Create the Softcard using the **ppmk** command. Enter a passphrase or password at the prompt.

```
# ppmk -n vaultsc
Enter new pass phrase:
Enter new pass phrase again:
New softcard created: HKLTU Softcard_ID
```

### 2.2.3 Confirm that the PKCS#11 library is available

In the example below an Operator Card Set has been created with the name **HashiCorp**. The card is present in the physical slot (card reader) of the HSM, and is loaded to **slot #1**.

1. Execute the **ckcheckinst** command to test the library:

```
# ckcheckinst
PKCS#11 library interface version 2.01
flags 0
manufacturerID "nCipher Corp. Ltd"
libraryDescription "nCipher PKCS#11 12.60.5-345-50b3"
implementation version 12.60
Slot  Status            Label
====  ======            =====
0  Fixed token       "accelerator"
1  Operator card     "Hashicorp"
2  No token present
3  No token present
```

2. Select the slot number of the Operator card and press the Enter key.

```
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).
```

3. Enter the passphrase for the OCS.

```
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.
```

## 2.3 Install Vault

1. If you have to use a Vault-specific user, log in as that user. The examples in this guide were executed as **root**. For instructions to create a Vault-specific user, see *Optional: Create a Vault-specific user and group* on page 7.
2. Create the directories for Vault installation. This command shows the recommended directory names and location. You can create the directories elsewhere but they must be available system-wide.

```
mkdir -p /opt/vault/{logs,bin,data}
```

3. Download the Vault package from HashiCorp, ensuring that it is the binary file for Enterprise with HSM support. For example:

```
wget https://releases.hashicorp.com/vault/1.3.2+ent.hsm/vault_1.3.2+ent.hsm_linux_
amd64.zip
```

4. Unzip the binary file and extract it to the working directory on the host machine, for example **/opt/vault/bin**. There should only be a single binary file named **vault_**.

```
# unzip vault_1.3.2+ent.hsm_linux_amd64.zip -d /opt/vault/bin
```

5. If the Vault binary has been copied between servers rather than downloaded directly with **wget** as in step 3, check that the directory has the correct permissions. If necessary, correct the permission settings:

```
# chmod a+x vault
# ls -la vault
-rwxr-xr-x. 1 root root 137112792 Dec 18 09:06 vault
```

6. Add the Vault binary file to the *PATH* :

```
# export PATH=$PATH:/opt/vault/bin
# echo "export PATH=$PATH:/opt/vault/bin" >> ~/.bashrc
```

7. Confirm that the binary file is available:

```
# vault
```

   A list of available features offered by Vault should be printed.

8. Add the **autocomplete** operation to make command-line use easier:

```
# vault -autocomplete-install
```

9. Restart the terminal:

```
# exec $SHELL
```

## 2.4 Configure Vault to run as a service

### 2.4.1 Create a config.json file

Set up a configuration file to enable Vault to be run as a service. See also *Vault commands* on page 24.

1. Open a new terminal and create a directory for the Vault configuration file:

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

2. Create a **config.json** file.

```
# vi /etc/vault/config.json
```

#### 2.4.1.1 Example configuration file for using Vault with OCS protection

The highlighted attributes below are to be defined by the user. The values are for example only.

```
# PKCS11 seal
seal "pkcs11" {
lib = "/opt/nfast/toolkits/pkcs11/libcknfast.so"
slot = "492971158"
pin = "pin"
key_label = "HashiCorp"
hmac_key_label = "hsm-hmac-key"
generate_key = "true"
}
storage "file" {
path = "/opt/vault/data/test.data"
}
# Addresses and ports on which Vault will respond to requests
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = "true"
"api_addr" = "http://vault_server_IP_address:8200"
"storage" = {}
"file" = {}
"path" = "/opt/vault/data"
}
ui = "true"
disable_mlock = "true"
```

#### 2.4.1.2 Example configuration file for using Vault with Softcard protection

Softcards are loaded to a virtual slot with a base decimal value of 761406613. Specifically, in PKCS11 configurations, it may be necessary to increase the base value to locate the virtual slot where your Softcard is loaded. Increase the final integer in increments until successful. In the example below, we can see that the slot ID is 761406615.

The highlighted attributes below are to be defined by the user. The values are for example only.

```
# PKCS11 seal
seal "pkcs11" {
lib = "/opt/nfast/toolkits/pkcs11/libcknfast.so"
slot = "761406615"
pin = "scpin"
key_label = "HashiCorp"
hmac_key_label = "hsm-hmac-key"
generate_key = "true"
}
storage "file" {
path = "/opt/vault/data/test.data"
}
# Addresses and ports on which Vault will respond to requests
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = "true"
"api_addr" = "http://vault_server_IP_address:8200"
"storage" = {}
"file" = {}
"path" = "/opt/vault/data"
}
ui = "true"
disable_mlock = "true"
```

### 2.4.2 Create and configure Vault directories

1. Create a vault file in **sysconfig**:

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

2. Create a service file in **/etc/systemd/system/**:

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

3. Add the following information to the file.

   In the example both **User** and **Group** are root as can be seen under **[Service]**. **User** and **Group** reflect the **Vault User / Group**, see *Optional: Create a Vault-specific user and group* on page 7.

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

   ```
   [Unit]
   Description=vault service
   ```

```
Requires=network-online.target
After=network-online.target
ConditionFileNotEmpty=/etc/vault/config.json
[Service]
User=root
Group=root
EnvironmentFile=-/etc/sysconfig/vault
Environment=GOMAXPROCS=2
Restart=on-failure
ExecStart=/opt/vault/bin/vault server -config=/etc/vault/config.json
StandardOutput=/opt/vault/logs/output.log
StandardError=/opt/vault/logs/error.log
LimitMEMLOCK=infinity
ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGTERM
[Install]
WantedBy=multi-user.target
```

4. If you are setting paths different from the default, you must edit the following lines as well in the configuration file:

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

## 2.5 Enable Vault

1. Set the following environment variable to allow Vault to be accessed from a web browser via the web user interface (web UI):

```
export VAULT_ADDRESS=http://vault_server_IP_address:8200
echo "export VAULT_ADDR=http://vault_server_IP_address:8200" >> ~/.bashrc
```

2. Enable Vault.

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

## 2.6 Start Vault

1. Start the vault service using the **config.json** file:

```
# start vault /opt/vault/bin/vault server -config=/etc/vault/config.json
```

You should receive an output similar to the following:

```
==> Vault server configuration:
HSM PKCS#11 Version: 2.1
HSM Library: nCipher PKCS#11 12.60.5-345-50b3
HSM Library Version: 12.60
HSM Manufacturer ID: nCipher Corp. Ltd
HSM Type: pkcs11
Cgo: enabled
Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201", max_
request_duration: "1m30s", max_request_size: "33554432", tls: "disabled")
Log Level: info
Mlock: supported: true, enabled: false
Recovery Mode: false
Storage: file
Version: Vault v1.3.2+ent.hsm
==> Vault server started! Log data will stream in below:
2020-02-03T12:21:45.677Z [INFO]  proxy environment: http_proxy= https_proxy= no_proxy=
```

The HSM library should report the nCipher PKCS#11 library version, and the HSM type as PKCS#11.

2. Confirm that there are two keys in the **/opt/nfast/kmdata/local** directory. Open a new terminal and execute the **nfkimfo** command:

```
# nfkminfo -l
```

An example of the expected output:

```
Keys protected by cardsets:
key_pkcs11_uc4415...-4324c7... 'nshield-hsm-hmac-key'
key_pkcs11_uc4415...-e26a66... 'HashiCorp'
```

3. Confirm the state of the vault service by running the **vault status** command.

   You may need to restart the terminal with `# exec $SHELL` or open a new terminal.

```
# vault status
```

4. Confirm the value of the **Initialized** and **Sealed** properties. The expected values are: **Initialised =**

**false**, **Sealed = true**.

```
Key                     Value
---                     -----
Recovery Seal Type      pkcs11
Initialized             false
Sealed                  true
Total Recovery Shares   0
Threshold               0
Unseal Progress         0/0
Unseal Nonce            n/a
Version                 n/a
HA Enabled              false
```

## 2.7 Generate the HSM-protected Vault HMAC key

### 2.7.1 Generate the Vault HMAC key with OCS protection

1. Stop Vault before generating the new HMAC key.

   Use Ctrl+C in the terminal window.

2. Using the nShield **generatekey** command, create an OCS-protected HMACSHA256 key.

   The required attributes of the key are shown in **bold**. The *plainname* can be user-defined but should be meaningful.

   The Operator card must be present in the HSM card reader to generate an OCS-protected key.

```
# generatekey pkcs11 protect=token
slot: Slot to read cards from? (0-3) [0] > 0
recovery: Key recovery? (yes/no) [yes] >
type: Key type? (DES3, DH, DHEx, DSA, HMACSHA1, HMACSHA256, HMACSHA384,
HMACSHA512, RSA, DES2, AES, Rijndael, ECDSA, ECDH, Ed25519,
X25519) [RSA] > HMACSHA256
size: Key size? (bits, 80-2048) [] > 256
plainname: Key name? [] > nshield-hsm-hmac-key
nvram: Blob in NVRAM (needs ACS)? (yes/no) [no] >
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                    nshield-hsm-hmac-key
nvram        Blob in NVRAM (needs ACS)   no
Loading `Hashicorp':
Module 1: 0 cards of 1 read
Module 1 slot 0: `Hashicorp' #1
Module 1 slot 2: empty
Module 1 slot 3: empty
Module 1 slot 0:- passphrase supplied - reading card
Card reading complete.
Key successfully generated.
```

3. When the key has been generated, confirm that the key has the correct permissions for use within Vault, that is, it can be used for generating and verifying MACs.

   a. Find the ID of the key, use the **nfkminfo** command with the **-l** parameter (for **--name-list**):

   ```
   # nfkminfo -l
   ```

   The output lists the keys that are protected by cardsets:

   ```
   key_pkcs11_uc4415...-e306385...      beb2db...   'Hashicorp'
   key_pkcs11_uc4415...-f6e23...        7fbdb9...   'nshield-hsm-hmac-key'
   ```

   b. Verify the key with the **nfkmverify** command with the following information:

   - Application name: *PKCS11*.
   - ID, in the example: the long string beginning with *uc* (*uc* indicates a token-protected key).

   ```
   # nfkmverify pkcs11 uc4415...-f6e235...
   ```

   In the output, look for the following message: Key may be used for: GENERATING or verifying message authentication codes

   ```
   ** [Application key pkcs11 uc4415...-f6e2357...] **
   [Named `nshield-hsm-hmac-key']
   Useable by HOST applications
   Cardset protected: 1/2 ephemeral [0s `Hashicorp']
   Cardset hash hash
   (Currently in Module #1 Slot #0: Card #1)
   Key useable INDEFINITELY (after card loading)
   Recovery ENABLED
   Type HMACSHA256 256 bits
   ```

```
Key may be used for: GENERATING or verifying message authentication codes
Generating module esn_number #1 (in same incarnation)
nCore hash hash
Public half is ABSENT
Verification successful, confirm details above.  1 key verified.
```

4. Update the **config.json** file with the new HMAC key name, see *Update the config.json file* on page 17.

### 2.7.2 Generate the Vault HMAC key with Softcard protection

1. Stop Vault before generating the new HMAC key.

   Use Ctrl+C in the terminal window.

2. Open a terminal and run the **generatekey** command to create a Softcard-protected HMACSHA256 key. *plainname* is user-defined.

```
# generatekey pkcs11 protect=softcard
recovery: Key recovery? (yes/no) [yes] >
type: Key type? (DES3, DH, DHEx, DSA, HMACSHA1, HMACSHA256, HMACSHA384,
HMACSHA512, RSA, DES2, AES, Rijndael, ECDSA, ECDH, Ed25519,
X25519) [RSA] > HMACSHA256
size: Key size? (bits, 80-2048) [] > 256
plainname: Key name? [] > nshield-hsm-hmac-key
nvram: Blob in NVRAM (needs ACS)? (yes/no) [no] >
key generation parameters:
operation    Operation to perform       generate
application  Application                pkcs11
protect      Protected by               softcard
softcard     Soft card to protect key   vaultsc
recovery     Key recovery               yes
verify       Verify security of key     yes
type         Key type                   HMACSHA256
size         Key size                   256
plainname    Key name                   nshield-hsm-hmac-key
nvram        Blob in NVRAM (needs ACS)  no
Please enter the pass phrase for softcard `vaultsc':
Please wait........
Key successfully generated.
Path to key: /opt/nfast/kmdata/local/key_pkcs11_uc0efd...
```

## 2.8 Update the config.json file

Edit the **config.json** file to name the OCS or the Softcard that you set up to protect the Vault HMAC key.

1. If the service is still running or has been restarted since the HMAC key was generated, stop it with Ctrl+C in the terminal window.
2. Edit the **config.json file** and change the value of **hmac-key-label** from the original to the new generated key name.

```
# vi /etc/vault/config.json
```

OCS:

```
pin = "pin"
key_label = "Hashicorp"
hmac_key_label = "nshield-hsm-hmac-key"
generate_key = "true"
```

Softcard:

```
pin = "scpin"
key_label = "Hashicorp"
hmac_key_label = "nshield-hsm-hmac-key"
generate_key = "true"
```

3. Save the **config.json** file and exit.
4. Re-enable **vault.service** so that it reads in the parameters from the modified **config.json** file.

```
# systemctl disable vault.service
# systemctl enable vault.service
# start vault /opt/vault/bin/vault server -config=/etc/vault/config.json
```

## 2.9 Initialize Vault

### 2.9.1 Create Vault Recovery Key and Initial Root Token

1. Ensure that Vault is both enabled and running.

```
# vault status
```

2. Open a new terminal and execute:

```
# vault operator init -recovery-shares=1 -recovery-threshold=1
```

You should see an output similar to the example below:

```
Recovery Key 1: RecoveryKey
Initial Root Token: s.InitialRootToken
Success! Vault is initialized
Recovery key initialized with 1 key shares and a key threshold of 1. Please securely
distribute the key shares printed above.
```

The Recovery Key and the Initial Root Token are not recoverable. Copy them to a secure place because they will be required later, for example when you are installing the Vault license in the next section.

### 2.9.2 Install the Vault license

To enable HSM functionality you must have a Vault Enterprise license.

1. Once you have the license file, give it a suitable name, for example **license-file**, and convert it to JSON format. If necessary, modify the file so that it resembles the following example:

```
{
   "text":
   "01MV4UU43BK5HGYYTOJZWFQMTMNNEWU33JJYZFS6CZPJIXSWSHKV2E46SNGVHUGMDZJ5LVM2
    KMKRAXQWTKNN2E4MLJVEE2COKRJGWSLJO5UVSM2WPJSEOOLULJMEUZTBK5IWST3JJF5E46S"
}
```

2. Copy the file to a convenient location, for example **/opt/vault/**.
3. Present the license file, using the Initial Root Token that was defined in *Create Vault Recovery Key and Initial Root Token* on page 18.

```
# curl --header "X-Vault-Token: s.Initial.RootToken" --request PUT --data @/path_to_license_
file/license_file_name http://127.0.0.1:8200/v1/sys/license
```

4. Verify that the license is installed:

```
# curl --header "X-Vault-Token: s.InitialRootToken" http://127.0.0.1:8200/v1/sys/license
```

### 2.9.3 Unseal Vault

Vault must be unsealed to make it ready for use.

1. Ensure that you have the Recovery key, see *Create Vault Recovery Key and Initial Root Token* on page 18.
2. Execute the **unseal** command:

```
# vault operator unseal -address=http://127.0.0.1:8200
Unseal Key (will be hidden):
```

3. Copy and paste the key to the command line.

   When you are copying the key, it will not be revealed. Click in the terminal only once to copy it to the command line.

If the seal remains in place this is probably due to the Recovery key being presented twice in the paste process. Try again by executing the **vault operator unseal** command.

4. Confirm the status of Vault with the **vault status** command. Enter the Recovery Key value supplied with the Initial Root Token at the prompt. The value of **Sealed** should be **false**:

```
# vault status
Key                     Value
---                     -----
Recovery Seal Type      shamir
Initialized             true
Sealed                  false
Total Recovery Shares   1
Threshold               1
Version                 vault_version
Cluster Name            cluster_name
Cluster ID              cluster_id
HA Enabled              false
```

## 2.10 Log in to Vault

Now that Vault is unsealed, you should be able to log in.

### 2.10.1 Log in from the command line

1. Run the **export** command:

```
# export VAULT_ADDR='http://127.0.0.1:8200'
```

2. Log in to Vault using the Initial Root Token:

```
# vault login s.InitialRootToken
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               s.InitialRootToken
token_accessor      3LrJOVCbBtpBgHWGeu70WAhu
token_duration      ∞
token_renewable     false
token_policies      ["root"]
identity_policies   []
policies            ["root"]
```

### 2.10.2 Log in from the web UI

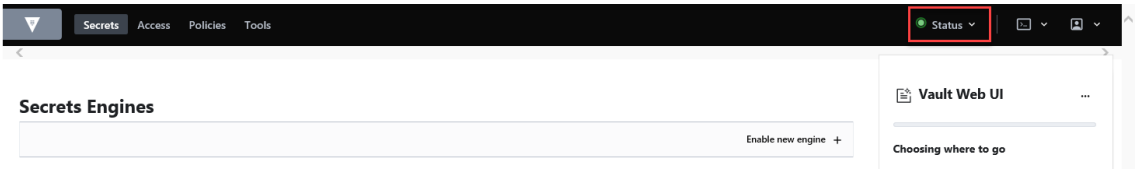
1. Open a browser and enter the IP address of the Vault Host Server:

   ```
   http://vault_server_IP_address:8200/ui/vault/auth
   ```

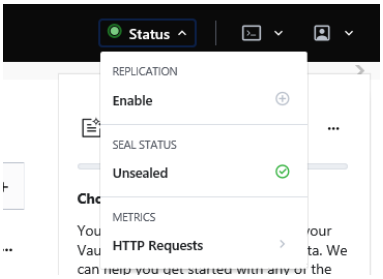
   The Vault sign-in page loads.

2. Enter the sign-in credentials:
   a. From the drop-down list, select the user name.
   b. For the Method, select Token.
   c. In the Token field, enter the Root Token key.
   d. Click Sign In.

   The Secrets Engines page is displayed. The Status light should be green:




3. Click the Status bar to check the details: Vault should be Unsealed:




## 2.11 Test the integration

Use the Secrets Engine operations of the Vault to check whether the integration works as intended. You can use these commands to generate, encrypt, and store any secret in a secure storage area, for example in a cubbyhole or in a key/value (KV) storage location.

### 2.11.1 Examine Vault Secrets

View the current secrets and default locations:

```
# vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_611f970b    per-token private secret storage
identity/     identity     identity_e583f78e     identity store
sys/          system       system_89d5c339       system endpoints used for control, policy
and debugging
```

### 2.11.2 Enable the KV engine

```
# vault secrets enable -version=1 kv
Success! Enabled the kv secrets engine at: kv/
```

### 2.11.3 Write secret data

Add data to the key/value storage area of Vault:

```
# vault kv put kv/path-to-secret key=value
Success! Data written to: kv/path-to-secret
```

### 2.11.4 Retrieve secret data

```
# vault kv get kv/path-to-secret
=== Data ===
Key    Value
---    -----
key    value
```

## 2.12 Seal Vault

When you are done, seal Vault:

```
# vault operator seal
Success! Vault is sealed.
```

# 3 Troubleshooting

Use the following table to troubleshoot the error messages shown.

| Error Message | Resolution |
|---|---|
| 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 nShield HSM physical slot. |

# Appendix A Vault commands

## C.1 Vault commands

| Task | Command |
|---|---|
| Check the status of Vault | **# vault status** |
| Log into Vault | **# vault login *s.InitialRootToken*** |
| Unseal Vault | **# vault operator unseal -address=http://127.0.0.1** - requires the Recovery key |
| Seal Vault | **# vault operator seal** |

## C.2 vault.service commands

| Task | Command |
|---|---|
| Disable the service | **# systemctl disable vault.service** |
| Enable the service | **# systemctl enable vault.service** |
| Stop the service | Ctrl+C in the running Vault service terminal |
| Start the service | Use the **config.json** file and location to start the service, for example:<br>**# start vault /opt/vault/bin/vault server -config=/etc/vault/config.json** |

# Contact Us

| | |
|---|---|
| Web site: | https://www.ncipher.com |
| Support: | https://help.ncipher.com |
| Email Support: | support@ncipher.com |
| Online documentation: | Available from the Support site listed above. |

You can also contact our Support teams by telephone, using the following numbers:

### Europe, Middle East, and Africa

| | |
|---|---|
| United Kingdom: | +44 1223 622444<br>One Station Square<br>Cambridge<br>CB1 2GA<br>UK |

### Americas

| | |
|---|---|
| Toll Free: | +1 833 425 1990 |
| Fort Lauderdale: | +1 954 953 5229<br>Sawgrass Commerce Center – A<br>Suite 130,<br>13800 NW 14 Street<br>Sunrise<br>FL 33323 USA |

### Asia Pacific

| | |
|---|---|
| Australia: | +61 8 9126 9070<br>World Trade Centre Northbank Wharf<br>Siddeley St<br>Melbourne VIC 3005<br>Australia |
| Japan: | +81 50 3196 4994 |
| Hong Kong: | +852 3008 3188<br>10/F, V-Point,<br>18 Tang Lung Street<br>Causeway Bay<br>Hong Kong |

www.ncipher.com

## About nCipher Security

nCipher Security, an Entrust Datacard company, is a leader in the general-purpose hardware security module (HSM) market, empowering world-leading organizations by delivering trust, integrity and control to their business critical information and applications. Today's fast-moving digital environment enhances customer satisfaction, gives competitive advantage and improves operational efficiency – it also multiplies the security risks. Our cryptographic solutions secure emerging technologies such as cloud, IoT, blockchain, and digital payments and help meet new compliance mandates. We do this using our same proven technology that global organizations depend on today to protect against threats to their sensitive data, network communications and enterprise infrastructure. We deliver trust for your business critical applications, ensure the integrity of your data and put you in complete control – today, tomorrow, always.
www.ncipher.com

Search: nCipher Security

TRUST. INTEGRITY. CONTROL