THALES

HashiCorp Vault: Integration Guide

THALES LUNA HSM AND DPOD LUNA CLOUD HSM

Document Information

Document Part Number	007-000264-001
Revision	E
Release Date	13 June 2022

Trademarks, Copyrights, and Third-Party Software

CONTENTS

Overview

This document describes various use cases to integrate HashiCorp Vault with Thales Luna HSM or Luna Cloud HSM services, and also covers leveraging Luna HSM for entropy augmentation. HashiCorp Vault Enterprise uses HSM for the following features:

- **Master Key Wrapping**: HashiCorp Vault protects its master key by transiting it through the HSM for encryption rather than using key splitting.
- **Automatic Unsealing**: HashiCorp Vault stores wrapped master key in storage and unseals it automatically when HSM is available.
- **Seal Wrapping**: Provides FIPS key storage conforming functionality for critical security parameters.
- **Entropy Augmentation:** HashiCorp Vault leverages HSM for augmenting system entropy via the PKCS#11 protocol.
- **Managed Key HSM:** Managed Key APIs delegates operations that requires private key material on HSM. Keys stored on HSM are used for PKI Operations.

The benefits of securing the keys with Luna HSMs include:

- Secure generation, storage and protection of the encryption keys on FIPS 140-2 level 3 validated hardware
- Full life cycle management of the keys
- Access to secure audit trail

  NOTE: Access to secure audit trail is available only for Luna HSM.

- Using cloud services with confidence

Certified Platforms

This integration is certified on the following platforms:

- Certified platforms on Luna HSM
- Certified platforms on Luna Cloud HSM

Certified platforms on Luna HSM

HSM Type	Platforms Tested
Luna HSM	RHEL

Luna HSM: Luna HSM appliances are purposefully designed to provide a balance of security, high performance, and usability that makes them an ideal choice for enterprise, financial, and government organizations. Luna HSMs physically and logically secure cryptographic keys and accelerate cryptographic processing. The Luna HSM on premise offerings include the Luna Network HSM, Luna PCIe HSM, and Luna USB HSMs. Luna HSMs are also available for access as an offering from cloud service providers such as IBM Cloud HSM and AWS CloudHSM Classic.

Certified platforms on Luna Cloud HSM

HSM Type	Platforms Tested
Luna Cloud HSM	RHEL

Luna Cloud HSM: Luna Cloud HSM services provide on-demand, cloud-based storage, management, and generation of cryptographic keys through a simple graphical user interface. With Luna Cloud HSM, security is simple, cost effective, and easy to manage because there is no hardware to buy, deploy, and maintain. As an Application Owner, you click and deploy services, generate usage reports, and maintain only those services that you need.

Prerequisites

Before you proceed with the any of the integrations described in this document, complete the following tasks:

Configure Luna HSM

To configure Luna HSM with HashiCorp Vault:

1. Verify that the HSM is set up, initialized, provisioned, and ready for deployment.
2. Create a partition on the HSM that will be later used by HashiCorp Vault.
3. If using a Luna Network HSM, register a client for the system and assign the client to the partition to create an NTLS connection. Initialize the Crypto Officer and Crypto User roles for the registered partition.
4. Ensure that the partition is successfully registered and configured. The command to see the registered partition is:

```
# /usr/safenet/lunaclient/bin/lunacm

lunacm (64-bit) v10.2.0-111. Copyright (c) 2020 SafeNet. All rights reserved.

Available HSMs:

Slot Id ->              0

Label ->                HashiCorp Vault

Serial Number ->        1280780175943

Model ->                LunaSA 7.4.0

Firmware Version ->     7.4.0

Configuration ->        Luna User Partition With SO (PW) Key Export With
                        Cloning Mode

Slot Description ->     Net Token Slot

FM HW Status ->         FM Ready

Current Slot Id: 0
```

5. For PED-authenticated HSM, enable partition policies 22 and 23 to allow activation and auto-activation.

> **NOTE:** Refer to Luna HSM documentation for detailed steps on creating NTLS connection, initializing the partitions, and assigning various user roles.

Set up Luna HSM High-Availability Group

Refer to the Luna HSM documentation for HA steps and details regarding configuring and setting up two or more HSM boxes on host systems. You must enable the HAOnly setting in HA for failover to work so that if the primary goes down due to any reason, all calls get automatically routed to the secondary until the primary recovers and starts up.

Set up Luna HSM in FIPS Mode

> **NOTE:** This setting is not required for Luna HSM Universal Client. This setting is applicable only for Luna HSM Client 7.x.

Under FIPS 186-3/4, the RSA methods permitted for generating keys are 186-3 with primes and 186-3 with aux primes. This means that RSA PKCS and X9.31 key generation is no longer approved for operation in a FIPS-compliant HSM. If you are using Luna HSM in FIPS mode, you have to make the following change in the configuration file:

```
Misc = {
RSAKeyGenMechRemap = 1;
}
```

The above setting redirects the older calling mechanism to a new approved mechanism when Luna HSM is in FIPS mode.

Configure Luna Cloud HSM Service

If you are using Luna Cloud HSM, you have the following configuration options:

- **Standalone Cloud HSM service using minimum client package**
- **Standalone Cloud HSM service using full Luna client package**
- **Luna HSM and Luna Cloud HSM service in hybrid mode**

> **NOTE:** Luna Client v10.x or higher is required for configuring Luna HSM device and Luna Cloud HSM service in hybrid mode.

Standalone Cloud HSM service using minimum client package

To configure Luna Cloud HSM service using minimum client package:

1. Transfer the downloaded .zip file to your Client workstation using pscp, scp, or other secure means.
2. Extract the .zip file into a directory on your client workstation.
3. Extract or untar the appropriate client package for your operating system. Do not extract to a new subdirectory; place the files in the client install directory.

   ```
   [Linux]

   cvclient-min.tar

   # tar -xvf cvclient-min.tar
   ```

4. Run the **setenv** script to create a new configuration file containing information required by the Luna Cloud HSM service.

   ```
   [Linux]

   Source the setenv script.

   # source ./setenv
   ```

5. Run the **LunaCM** utility and verify the Cloud HSM service is listed.

Standalone Cloud HSM service using full Luna client package

To configure Luna Cloud HSM service using full Luna client package:

1. Transfer the downloaded .zip file to your Client workstation using pscp, scp, or other secure means.
2. Extract the .zip file into a directory on your client workstation.
3. Extract or untar the appropriate client package for your operating system. Do not extract to a new subdirectory; place the files in the client install directory.

   ```
   [Linux]

   cvclient-min.tar

   # tar -xvf cvclient-min.tar
   ```

4. Run the **setenv** script to create a new configuration file containing information required by the Luna Cloud HSM service.

   ```
   [Linux]

   Source the setenv script.

   # source ./setenv
   ```

5. Copy the server and partition certificates from the Cloud HSM service client directory to Luna client certificates directory:

   Cloud HSM Certificates:

   ```
   server-certificate.pem

   partition-ca-certificate.pem

   partition-certificate.pem
   ```

   LunaClient Certificate Directory:

   ```
   [Linux default location for Luna Client]

   /usr/safenet/lunaclient/cert/
   ```

   > **NOTE:** Skip this step for Luna Client v10.2 or higher.

6. Open the configuration file from the Cloud HSM service client directory and copy the **XTC** and **REST** section.

   ```
   [Linux]

   Chrystoki.conf
   ```

7. Edit the Luna Client configuration file and add the **XTC** and **REST** sections copied from Cloud HSM service client configuration file.

8. Change server and partition certificates path from step 5 in **XTC** and **REST** sections. Do not change any other entries provided in these sections.

   [XTC]

```
. . .
PartitionCAPath=<LunaClient_cert_directory>\partition-ca-certificate.pem
PartitionCertPath00=<LunaClient_cert_directory>\partition-certificate.pem
. . .

[REST]
. . .
SSLClientSideVerifyFile=<LunaClient_cert_directory>\server-certificate.pem
. . .
```

   NOTE: Skip this step for Luna Client v10.2 or higher.

9. Edit the following entry from the **Misc** section and update the correct path for the **plugins** directory:

```
Misc]
PluginModuleDir=<LunaClient_plugins_directory>

[Linux Default]

/usr/safenet/lunaclient/plugins/
```

   Save the configuration file. If you wish, you can now safely delete the extracted Cloud HSM service client directory.

10. Reset the **ChrystokiConfigurationPath** environment variable and point back to the location of the Luna Client configuration file.

    [Linux]

    Either open a new shell session, or export the environment variable for the current session pointing to the location of the Chrystoki.conf file:

```
# export ChrystokiConfigurationPath=/etc/
```

11. Run the **LunaCM** utility and verify that the Cloud HSM service is listed. In hybrid mode, both Luna and Cloud HSM service will be listed.

    NOTE: Follow the Luna Cloud HSM documentation for detailed steps for creating service, client, and initializing various user roles.

Luna HSM and Luna Cloud HSM service in hybrid mode

To configure Luna HSM and Luna Cloud HSM service in hybrid mode, follow the steps mentioned under the Standalone Cloud HSM service using full Luna client package section above.

NOTE: Luna Client v10.x or higher is required for configuring Luna HSM device and Luna Cloud HSM service in hybrid mode.

Luna Cloud HSM Service in FIPS mode

Luna Cloud HSM service operates in both FIPS and non-FIPS mode. If your organization requires non-FIPS algorithms for your operations, ensure you enable the **Allow non-FIPS approved algorithms** check box when

configuring your Cloud HSM service. The FIPS mode is enabled by default. Refer to the Mechanism List in the SDK Reference Guide for more information about available FIPS and non-FIPS algorithms.

Set up HashiCorp Vault

To download and install HashiCorp Vault and set up the system environment:

> **NOTE:** HashiCorp Vault is packaged as a zip archive and distributed as a binary package for all supported platforms. For more details, refer to HashiCorp Documentation.

1. Download the HashiCorp Vault package from HashiCorp.

   https://www.hashicorp.com/products/vault/trial

2. Unzip the package in the working directory on the host machine. HashiCorp Vault runs as a single binary named `vault`.
3. Add the current working directory to the PATH so that `vault` is executable from any directory.
4. After installing `vault`, verify the installation by opening a new terminal session and checking that the vault binary is available. By executing `vault`, you should be able to see help output similar to what's depicted in the following example:

```
Usage: vault <command> [args]

Common commands:
    read        Read data and retrieves secrets
    write       Write data, configuration, and secrets
    delete      Delete secrets and configuration
    list        List data or secrets
    login       Authenticate locally
    agent       Start a Vault agent
    server      Start a Vault server
    status      Print seal and HA status
    unwrap      Unwrap a wrapped secret

Other commands:
    audit           Interact with audit devices
    auth            Interact with auth methods
    kv              Interact with Vault's Key-Value storage
    lease           Interact with leases
    namespace       Interact with namespaces
    operator        Perform operator-specific tasks
    path-help       Retrieve API help for paths
    plugin          Interact with Vault plugins and catalog
    policy          Interact with policies
    secrets         Interact with secrets engines
    ssh             Initiate an SSH session
    token           Interact with tokens
```

Integrating HashiCorp Vault with Luna HSM

To integrate HashiCorp Vault with a Luna HSM, complete the following tasks:

> Enable PKCS11 seal
> Configure Entropy Augmentation
> Start the Vault
> Initialize the Vault
> Log into the Vault
> Use the Secret Engine
> Enable Entropy Augmentation
> Rotating HashiCorp Vault Keys

Enable the PKCS11 seal

The PKCS11 seal configures HashiCorp Vault to use an HSM with PKCS11 as the seal wrapping mechanism. To enable the PKCS11 seal, create the HashiCorp Vault's configuration file named `config.json` and specify the **seal**,**storage** and **listener** stanzas, as indicated:

```
# PKCS11 seal
seal "pkcs11" {
  lib = "<path to cryptoki library>"
  slot = "<slot number>"
  pin = "<partition password>"
  key_label = "HashiCorp"
  hmac_key_label = "HashiCorp_hmac"
  generate_key = "true"
}

storage "file" {
  path = "/tmp/vault"
}

# Addresses and ports on which Vault will respond to requests
listener "tcp" {
  address     = "127.0.0.1:8200"
  tls_disable = "true"
}
ui = true
```

Here:

`lib` is the path to the PKCS#11 library shared object file.

`slot` is the HSM partition slot number.

`pin` is the HSM partition password.

`generate_key` triggers Vault to generate a key if no existing key with the label specified by `key_label` can be found at Vault initialization time.

`hmac_key_label` is the label of the key for HMACing.

`mechanism` is the encryption/decryption mechanism, as a decimal or hexadecimal (prefixed by 0x) string.

Currently supported mechanisms (in order of precedence) include:

`0x1085 CKM_AES_CBC_PAD (HMAC mechanism required)`

`0x1082 CKM_AES_CBC (HMAC mechanism required)`

`0x1087 CKM_AES_GCM`

`0x0009 CKM_RSA_PKCS_OAEP`

`0x0001 CKM_RSA_PKCS`

This example defines the RSA_PKCS_OEAP mechanism for Master Key Wrapping in `config.json` file:

```
# PKCS11 seal
seal "pkcs11" {
  lib = "<path to cryptoki library>"
  slot = "<slot number>"
  pin = "<partition password>"
  key_label = "rsa_oaep_key"
  mechanism = "0x0009"
  rsa_oaep_hash = "sha256"
  generate_key = "true"
}

storage "file" {
  path = "/tmp/vault"
}

# Addresses and ports on which Vault will respond to requests
listener "tcp" {
  address     = "127.0.0.1:8200"
  tls_disable = "true"
}
ui = true
```

NOTE: The HSM seal can also be activated by providing the following environment variables:

VAULT_HSM_LIB, VAULT_HSM_SLOT, VAULT_HSM_PIN, VAULT_HSM_KEY_LABEL, VAULT_HSM_HMAC_KEY_LABEL VAULT_HSM_MECHANISM and VAULT_HSM_GENERATE_KEY

Although the configuration file enables you to pass in VAULT_HSM_PIN as part of the seal's parameters, it is strongly recommended to set this value through environment variables.

Configure Entropy Augmentation

Vault Enterprise version 1.3 introduced the Entropy Augmentation function to leverage HSM for augmenting system entropy through the PKCS#11 protocol. To configure Entropy Augmentation for Vault version 1.3 and above, define the **entropy** stanza in server configuration file `config.json`, as indicated here:

```
# Entropy
entropy "seal" {
mode = "augmentation"
}
```

NOTE: Since Vault will delegate the random number generation to the HSM, you must set the seal stanza with HSM cluster connection information.

Start the Vault

Start the Vault server using the following configuration file:

```
# ./vault server -config config.json
```

```
[root@localhost home]# ./vault server -config config.json
==> Vault server configuration:

     HSM PKCS#11 Version: 2.20
            HSM Library: Chrystoki
     HSM Library Version: 10.1
     HSM Manufacturer ID: SafeNet
               HSM Type: pkcs11
                    Cgo: enabled
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201",
               Log Level: info
                  Mlock: supported: true, enabled: true
           Recovery Mode: false
                 Storage: file
                 Version: Vault v1.3.0+ent.hsm

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

Initialize the Vault

You must initialize the Vault before you begin configuring and managing secrets. When Vault is initialized while using an HSM, rather than unseal keys being returned to the operator, recovery keys are returned. Some Vault operations such as generation of a root token require these recovery keys. To initialize the HashiCorp Vault:

1. Launch a new terminal session and execute the following command:

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

2. Check the status of Vault by executing the following command:

```
# ./vault status
```

```
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
```

3. Initialize the Vault by executing the following command:

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

This will generate a recovery key and initial root token. Copy this key and keep it in safe place.

```
Recovery Key 1: vSz2okfEnhyEqXT4gXTUJDTVo5bLa7qrYSVtZEhi5ZA=

Initial Root Token: 5tvLldtsxhKxVAGenD5abbHI

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.
```

Note the following logs in the first terminal where Vault Server is running:

```
2018-11-16T06:15:48.859-0500 [INFO]  core: loaded wrapping token key
2018-11-16T06:15:48.860-0500 [INFO]  core: successfully mounted backend:
type=kv path=secret/
…
2018-11-16T06:15:48.952-0500 [INFO]  core: root token generated
…
2018-11-16T06:15:49.031-0500 [INFO]  core: vault is unsealed
2018-11-16T06:15:49.032-0500 [INFO]  core: post-unseal setup starting
2018-11-16T06:15:49.153-0500 [INFO]  core: loaded wrapping token key
…
2018-11-16T06:15:49.157-0500 [INFO]  core: successfully unsealed with
stored key(s): stored_keys_used=1
2018-11-16T06:15:49.157-0500 [INFO]  expiration: lease restore complete
```

4. Verify the keys generated on the partition by executing the `partition contents` command in lunacm.

```
lunacm:>partition contents

       The 'Crypto Officer' is currently logged in.  Looking for objects
       accessible to the 'Crypto Officer'.

       Object list:

       Label:         hashicorp_hmac
       Handle:        97
       Object Type:   Symmetric Key
       Object UID:    35000000180000025b990800

       Label:         hashicorp
       Handle:        85
       Object Type:   Symmetric Key
       Object UID:    34000000180000025b990800


       Number of objects:  2
```

Log into the Vault

You must log in to the Vault to begin configuring and managing the secrets engine. To log in to the HashiCorp Vault:

1. Run the `vault login` command:

   `# ./vault login <VAULT-TOKEN>`

   Here, `<VAULT-TOKEN>` is the initial root token generated during Vault initialization.

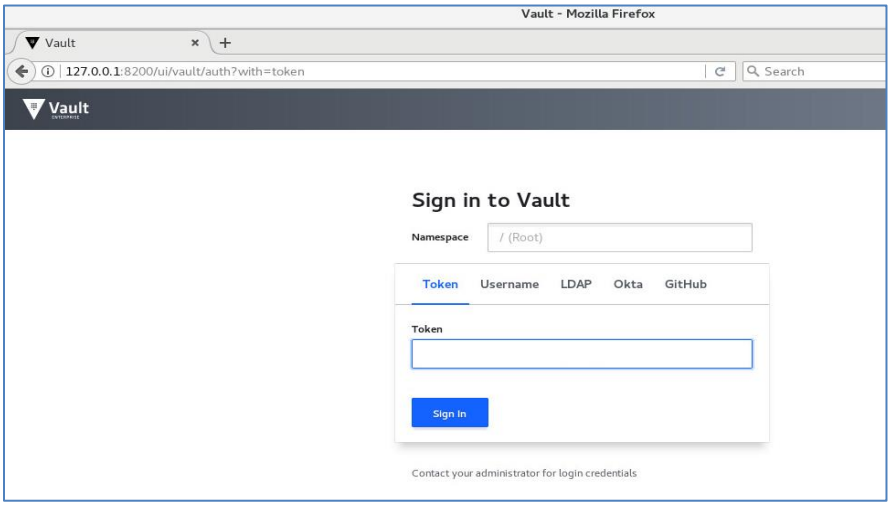
```
[root@hashicorp]# ./vault login 5tvLldtsxhKxVAGenD5abbHI
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                  Value
---                  -----
token                5tvLldtsxhKxVAGenD5abbHI
token_accessor       94w73PrwFKjk8b0MyaMmPRay
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

2. Verify that the Vault server is now initialized and auto-unsealed.

```
[root@localhost home]#  ./vault status
Key                      Value
---                      -----
Recovery Seal Type       shamir
Initialized              true
Sealed                   false
Total Recovery Shares    1
Threshold                1
Version                  1.3.0+ent.hsm
Cluster Name             vault-cluster-f52d12c3
Cluster ID               3085e6c5-c2b7-fdd0-8871-a826bbf86916
HA Enabled               false
```

3. Access the Vault console through browser using the URL: http://127.0.0.1:8200/ui, if required, as UI is enabled in configuration file (config.json).



4. Log in to Vault console by entering the **Token** that is the initial root token generated during Vault initialization.

Use the Secrets Engine

Secrets engines are components that store, generate, or encrypt data. The `kv` secrets engine is used to store arbitrary secrets within the configured physical storage for Vault. Versioning can be enabled and a configurable number of versions for each key can be stored. To use the Secrets Engine:

1. Execute the following command to view the secrets:

```
# ./vault secrets list
```

```
[root@hashicorp]# ./vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_16b25559    per-token private secret storage
identity/     identity     identity_a6ee82e3     identity store
secret/       kv           kv_70be3e30           key/value secret storage
sys/          system       system_494262c6       system endpoints used for control, policy and debugging
```

2. Enable the kv engine:

```
# ./vault secrets enable -version=1 kv

Success! Enabled the kv secrets engine at: kv/
```

3. Write arbitrary secret data:

```
# ./vault kv put kv/my-secret my-value=s3cr3t

Success! Data written to: kv/my-secret
```

4. Display the secret value by executing command:

```
 # ./vault kv get kv/my-secret

====== Data ======
Key          Value
---          -----
my-value     s3cr3t
```

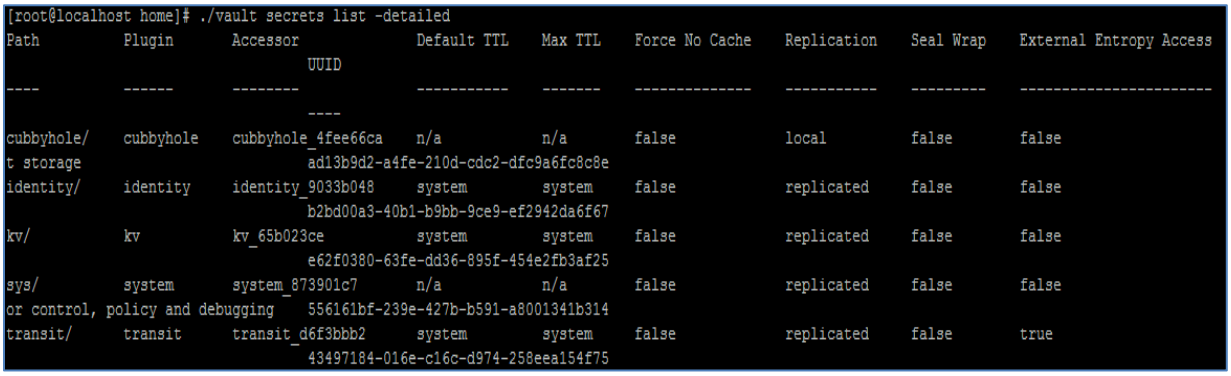
Enable Entropy Augmentation

To leverage the external entropy source, set the `external_entropy_access` parameter to true while enabling a `secrets` engine or `auth` method. To enable external entropy source on a transit secrets engine:

1. Enable `transit` secrets engine with external entropy source.

```
# ./vault secrets enable -external-entropy-access transit
Success! Enabled the transit secrets engine at: transit/
```

2. List the enabled secrets engine with `-detailed` flag.



```
[root@localhost home]# ./vault secrets list -detailed
Path          Plugin       Accessor             Default TTL    Max TTL    Force No Cache    Replication    Seal Wrap    External Entropy Access
                           UUID
----          ------       --------             -----------    -------    --------------    -----------    ---------    ----------------------
                           ----
cubbyhole/    cubbyhole    cubbyhole_4fee66ca    n/a            n/a        false             local          false        false
t storage                  ad13b9d2-a4fe-210d-cdc2-dfc9a6fc8c8e
identity/     identity     identity_9033b048     system         system     false             replicated     false        false
                           b2bd00a3-40b1-b9bb-9ce9-ef2942da6f67
kv/           kv           kv_65b023ce           system         system     false             replicated     false        false
                           e62f0380-63fe-dd36-895f-454e2fb3af25
sys/          system       system_873901c7       n/a            n/a        false             replicated     false        false
or control, policy and debugging    556161bf-239e-427b-b591-a8001341b314
transit/      transit      transit_d6f3bbb2      system         system     false             replicated     false        true
                           43497184-016e-c16c-d974-258eea154f75
```

3. Notice that the **External Entropy Access** is set to **true** for **transit/**.

4. Use the `transit` secrets engine to encrypt sensitive data that leverages the HSM as its external entropy source. Create a new encryption key named "orders".

```
# ./vault write -f transit/keys/orders
```

```
Success! Data written to: transit/keys/orders
```

5. Send a base64-encoded string to be encrypted by Vault.

```
# ./vault write transit/encrypt/orders plaintext=$(base64 <<< "4111 1111 1111
1111")
```

```
Key            Value
---            -----
```

```
ciphertext
vault:v1:ZXKU9Yc8+BefMCkPJVUksh5y0NlTymeToyTKl7NzdE5I4CpRtcjjPnUsvVmwPpQ/
```

6. Verify that ciphertext can be decrypted.

```
# ./vault write transit/decrypt/orders
\ciphertext="vault:v1:ZXKU9Yc8+BefMCkPJVUksh5y0NlTymeToyTKl7NzdE5I4CpRtcjjPnUsv
VmwPpQ/"
```

```
Key           Value
---           -----
```

```
plaintext     NDExMSAxMTExIDExMTEgMTExMQo=
```

7. Decode to get the original plaintext string.

```
# base64 --decode <<< NDExMSAxMTExIDExMTEgMTExMQo=
```

```
4111 1111 1111 1111
```

NOTE: When the external entropy access is enabled, the connectivity to the HSM is required. If the HSM becomes unreachable for any reason, the transit secrets engine returns an error and data cannot be encrypted or decrypted until the HSM connection gets restored.

This completes the integration of HashiCorp Vault with Luna HSM or DPoD Luna Cloud HSM service.

Rotating HashiCorp Vault Keys

The PKCS11 seal supports rotating keys by using different key labels to track key versions. To rotate the key value, generate a new key in a different key label in the HSM and update Vault's configuration with the new key label value. Restart the Vault instance to pick up the new key label and all new encryption operations will use the updated key label. Old keys must not be disabled or deleted since they are used to decrypt older data. If rotation is desired for data that was seal wrapped prior to this version, set `default_key_label` and `hmac_default_key_label` to enable decryption of older values.

To rotate HashiCorp Vault Keys:

1. Stop the Vault server from the terminal, if running.
2. Change the configuration file `config.json` as follows:

```
#Entropy
entropy "seal" {
mode = "augmentation"
}

# PKCS11 seal
seal "pkcs11" {
  lib = "<path to cryptoki library>"
  slot = "<slot number>"
  pin = "<partition password>"
  default_key_label="HashiCorp"
  key_label = "HashiCorp_rot"
  default_hmac_key_label = "HashiCorp_hmac"
  hmac_key_label = "HashiCorp_hmac_rot"
  generate_key = "true"
}
storage "file" {
  path = "/tmp/vault"
}
```

3. Start the Vault using the updated configuration file.

```
# ./vault server -config config.json
```

4. Launch a new terminal session and execute the following command :

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

5. Verify the Vault status and list the secrets by executing the following command:

```
# ./vault secrets list
```

```
[root@hashicorp]# ./vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_16b25559    per-token private secret storage
identity/     identity     identity_a6ee82e3     identity store
kv/           kv           kv_ae934885           n/a
secret/       kv           kv_70be3e30           key/value secret storage
sys/          system       system_494262c6       system endpoints used for control, policy and debugging
```

6. Verify the keys generated in the partition by executing the `partition contents` command in lunacm utility on host.

```
lunacm:>partition contents

        The 'Crypto Officer' is currently logged in.  Looking for objects
        accessible to the 'Crypto Officer'.

        Object list:

        Label:          hashicorp_hmac_rot
        Handle:         102
        Object Type:    Symmetric Key
        Object UID:     37000000180000025b990800

        Label:          hashicorp_rot
        Handle:         99
        Object Type:    Symmetric Key
        Object UID:     36000000180000025b990800

        Label:          hashicorp_hmac
        Handle:         97
        Object Type:    Symmetric Key
        Object UID:     35000000180000025b990800

        Label:          hashicorp
        Handle:         85
        Object Type:    Symmetric Key
        Object UID:     34000000180000025b990800


        Number of objects:  4


Command Result : No Error
```

This completes the rotation of the HashiCorp Vault keys using Luna HSM or DPoD Luna Cloud HSM service.

Integrating HashiCorp Vault as a Docker Container with Luna HSMs

If you are integrating HashiCorp Vault as a Docker Container with a Luna HSM, complete the following steps:

> Set up Docker

> Set up Docker-Compose

> Enabling Host to use Luna HSM in Container

> Creating a Docker Container with HashiCorp Vault and Luna HSM

Set up Docker

Refer to Docker Documentation for installing Docker and running Docker service. After installation, ensure that the Docker service is up and running successfully.

```
[root@localhost ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-03-16 20:02:18 IST; 1 weeks 5 days ago
     Docs: https://docs.docker.com
 Main PID: 36659 (dockerd)
    Tasks: 8
   Memory: 38.6M
   CGroup: /system.slice/docker.service
           └─36659 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 16 20:02:17 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:17.801410394+05:30" level=error msg="Failed to built-in GetDriver graph btrfs /var/lib/docker"
Mar 16 20:02:17 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:17.865637726+05:30" level=warning msg="Your kernel does not support cgroup blkio weight"
Mar 16 20:02:17 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:17.865715653+05:30" level=warning msg="Your kernel does not support cgroup blkio weight_device"
Mar 16 20:02:17 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:17.866150886+05:30" level=info msg="Loading containers: start."
Mar 16 20:02:18 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:18.177926256+05:30" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daem
Mar 16 20:02:18 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:18.448992988+05:30" level=info msg="Loading containers: done."
Mar 16 20:02:18 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:18.515718260+05:30" level=info msg="Docker daemon" commit=906f57f graphdriver(s)=overlay2 version=20.10.13
Mar 16 20:02:18 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:18.516663431+05:30" level=info msg="Daemon has completed initialization"
Mar 16 20:02:18 localhost.localdomain systemd[1]: Started Docker Application Container Engine.
Mar 16 20:02:18 localhost.localdomain dockerd[36659]: time="2022-03-16T20:02:18.589284345+05:30" level=info msg="API listen on /var/run/docker.sock"
[root@localhost ~]#
```

Set up Docker-Compose

Refer to Docker Compose Documentation for installing Docker Compose. After installation, ensure that the Docker Compose is installed successfully.

```
[root@localhost ~]# docker-compose --version
docker-compose version 1.23.2, build 1110ad01
[root@localhost ~]#
```

Enabling Host to use Luna HSM in Container

Refer to the Configure Luna HSM section for setting up NTL connection with Luna HSM. After setting up the NTL connection, copy the Chrystoki.conf, Client and Server certificates to the host where you want to create the container running Vault inside.

> **Note:** For Docker environment, please disable IP Check setting on Luna. To disable run the "**ntls ipcheck disable**" on Luna SA console. Skip this step if you are using public IPs for containers.

Creating a Docker Container with HashiCorp Vault and Luna HSM

Follow these steps to integrate HashiCorp Vault with Luna HSM as a Docker Container:

> Set up Vault in Container

> Log in to the Vault on host

> Use the secrets engine on host

> Use Vault from one container to another

Set up Vault in Container

1. Connect to the host system as root or as a user with administrative privileges.
2. Create a working directory to place all the necessary binaries and files required for the integration.

```
# mkdir /opt/luna-docker
# cd /opt/luna-docker
```

3. Copy the Luna minimal client, Vault binary, Luna configuration file and Luna client/server certificates in the working directory. Following is the list of required files to proceed further.

```
[root@localhost ~]# ls -ltr /opt/luna-docker/
total 109284
-rw-r--r--. 1 root root 46080000 Mar 16 19:06 LunaClient-Minimal-10.4.0-417.x86_64.tar
-rw-r--r--. 1 root root 65806944 Mar 16 19:07 vault_1.6.2+ent.hsm_linux_amd64.zip
-rw-r--r--. 1 root root     1168 Mar 16 19:09 dockerclient.pem
-rw-r--r--. 1 root root     1743 Mar 16 19:09 dockerclientKey.pem
-rw-r--r--. 1 root root     1172 Mar 16 19:09 CAFile.pem
-rw-r-----. 1 root root     1366 Mar 16 19:09 Chrystoki.conf
[root@localhost ~]#
```

> **Note:** You can use any supported version of Luna Minimal Client and Vault binary as per your requirement. All other files are copied from the workstation where you have already registered a Luna Partition and created a NTL connection.

4. Update the Luna configuration file **Chrystoki.conf** in the current directory and edit the following sections of the **Chrystoki.conf** file to make it compatible with Luna Minimal Client library paths.

```
Chrystoki2 = {
   LibUNIX = /usr/safenet/lunaclient/libs/64/libCryptoki2.so;
   LibUNIX64 = /usr/safenet/lunaclient/libs/64/libCryptoki2_64.so;
}
```

5. Remove the “Secure Trusted Channel” section from the **Chrystoki.conf** file.

```
Secure Trusted Channel = {
ClientTokenLib = /usr/safenet/lunaclient/libs/64/libSoftToken.so;
SoftTokenDir = /usr/safenet/lunaclient/configData/token;
ClientIdentitiesDir = /usr/safenet/lunaclient/data/client_identities;
PartitionIdentitiesDir = /usr/safenet/lunaclient/data/partition_identities;
}
```

6. Create a HashiCorp Vault configuration file **config.json** for enabling the PKCS11 seal in the current working directory, as indicated here:

```
# PKCS11 seal
seal "pkcs11" {
lib = "/usr/safenet/lunaclient/libs/64/libCryptoki2_64.so"
slot = "0"
pin = "userpin1"
key_label = "HashiCorpRSA"
hmac_key_label = "HashiCorpHMac"
generate_key = "true"
mechanism = "0x0009"
rsa_oaep_hash = "sha256"
}
storage "file" {
path = "/tmp/vault"
}
# Addresses and ports on which Vault will respond to requests
listener "tcp" {
address = "0.0.0.0:8200"
tls_disable = "true"
}
# Entropy
entropy "seal" {
mode = "augmentation"
}
disable_mlock = true
ui = true
```

7. Create a shell script **vault_start.sh** containing minimum commands to up and run HashiCorp Vault inside ihe Docker container. Keep the script in current working directory, as indicated here:

```
#! /bin/bash
# REMOVING VAULT DIRECTORY IF ALREADY PRESENT
rm -rf /tmp/vault > /dev/null
# GO TO / DIRECTORY WHERE VAULT BINARY IS PRESENT
cd /hashi_vault
# START VAULT IN THE BACKGROUND
./vault server -config config.json &
# SLEEPING FOR 10 SECONDS, FOR VAULT TO START UP AND LOAD
sleep 10s
# SETTING THE VAULT_ADDR PARAMETER
export VAULT_ADDR='http://0.0.0.0:8200'
# LISTING VAULT STATUS
./vault status
# INITIALIZING THE VAULT
./vault operator init -recovery-shares=1 -recovery-threshold=1
# SLEEPING FOR 10 SECONDS, FOR VAULT TO GET INITIALIZED
sleep 10s
# LISTING VAULT STATUS
./vault status
# RESTARTING THE VAULT
kill -9 `pgrep vault`
sleep 10s
./vault server -config config.json
sleep 10s
```

8. Create a **Dockerfile** in the current directory that will create a Docker image containing **Vault** and the associated resources for Luna Minimal Client to communicate with the Luna HSM. Check the file names and paths for all the resources.

```
### luna-docker image
FROM centos:centos7 as vault-server
WORKDIR /hashi_vault

COPY vault_1.6.2+ent.hsm_linux_amd64.zip /tmp/
RUN yum install -y unzip
RUN unzip /tmp/vault_1.6.2+ent.hsm_linux_amd64.zip
COPY config.json /hashi_vault/config.json
COPY vault_start.sh /vault_start.sh

COPY LunaClient-Minimal-10.4.0-417.x86_64.tar /tmp/
RUN mkdir -p /usr/safenet/lunaclient
RUN mkdir -p /usr/safenet/lunaclient/cert
RUN mkdir -p /usr/safenet/lunaclient/cert/client
RUN mkdir -p /usr/safenet/lunaclient/cert/server
RUN tar -xvf /tmp/LunaClient-Minimal-10.4.0-417.x86_64.tar --strip 1 -C
/usr/safenet/lunaclient
RUN cp /usr/safenet/lunaclient/openssl.cnf /usr/safenet/lunaclient/bin
ENV ChrystokiConfigurationPath=/etc
COPY Chrystoki.conf /etc/Chrystoki.conf
COPY CAFile.pem /usr/safenet/lunaclient/cert/server
COPY dockerclientKey.pem /usr/safenet/lunaclient/cert/client
COPY dockerclient.pem /usr/safenet/lunaclient/cert/client

RUN chmod +x /vault_start.sh
ENTRYPOINT ["/vault_start.sh"]
```

9. Create an image with `docker build` command and **Dockerfile**:

```
# docker build . -t vault-server
```

You will see a confirmation message similar to the following when the image is built successfully.

```
Removing intermediate container edfc2ebabe76
 ---> cebba8ffbc60
Successfully built cebba8ffbc60
Successfully tagged vault-server:latest
[root@localhost luna-docker]#
```

The image **vault-server** will be listed along with other images:

```
# docker images
```

The created image will be listed along with other images, if present. The output will be similar to the following example:

```
[root@localhost luna-docker]# docker images
REPOSITORY     TAG       IMAGE ID       CREATED              SIZE
vault-server   latest    cebba8ffbc60   About a minute ago   716MB
centos         centos7   eeb6ee3f44bd   6 months ago         204MB
[root@localhost luna-docker]#
```

10. Create the yaml file **docker-compose.yaml** and ensure that the file contains the following information:

```
version: "3"
volumes:
  vault:
services:
  HashiCorpVaultContainer:
    container_name: hashicorp_vault_container
    image: vault-server:latest
    ports:
      - 8200:8200
    volumes:
      - vault:/hashi_vault
```

11. Create and start the container using **docker-compose up** command:

```
# docker-compose up
```

When the Vault initializes successfully, you will see output similar to the following example:

```
hashicorp_vault_container  | 2022-03-29T14:07:19.321Z [INFO]  core: post-unseal setup complete
hashicorp_vault_container  | 2022-03-29T14:07:19.321Z [INFO]  core: vault is unsealed
hashicorp_vault_container  | 2022-03-29T14:07:19.321Z [INFO]  core: unsealed with stored key
```

12. Open a new terminal and ensure that the Docker container by the name **hashicorp_vault_container** is running using the command:

```
# docker ps -a
```

```
[root@localhost luna-docker]# docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS                      PORTS                                       NAMES
40fa8fb1dde9   vault-server:latest   "/vault_start.sh"        3 minutes ago   Up 3 minutes                0.0.0.0:8200->8200/tcp, :::8200->8200/tcp   hashicorp_vault_container
a27dcb05bd1a   bb1b6515cc60         "/bin/sh -c 'unzip /…"   20 minutes ago  Exited (11) 20 minutes ago                                              heuristic_fermat
[root@localhost luna-docker]#
```

13. Run the docker logs command to get the root token which will be required when logging in to the Vault.

```
# docker logs hashicorp_vault_container > logs.txt
```

In the logs.txt, you will see the “Initial Root Token”, as depicted below:

```
HA Enabled                false
Recovery Key 1: X6wq/vanVrP5S1Z9PYUa7Y445HsI6/NZjeM6KT5p7Oc=

Initial Root Token: s.DpxDy6PvMUw5th7uJyTqsntf
```

14. Create a soft link for the vault binary, mounted from container to host.

```
# ln -sf /var/lib/docker/volumes/luna-docker_vault/_data/vault
```

```
[root@localhost luna-docker]# ls -ltr vault
lrwxrwxrwx. 1 root root 53 Mar 29 20:06 vault -> /var/lib/docker/volumes/luna-docker_vault/_data/vault
[root@localhost luna-docker]#
```

15. Set the VAULT_ADDR variable on host to begin using the vault service on host system.

```
# export VAULT_ADDR='http://0.0.0.0:8200'
```

Log in to the Vault on host

You must log in to the Vault to begin configuring and managing the secrets engine. To log in to the HashiCorp Vault:

1. Log in to the Vault.

```
# ./vault login <VAULT-TOKEN>
```

Here, <VAULT-TOKEN> is the initial root token generated during Vault initialization.

```
[root@localhost luna-docker]# ./vault login s.YcX2cMGvNOOY4nwRFbiOGAfD
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.YcX2cMGvNOOY4nwRFbiOGAfD
token_accessor       1jNfWxKsx17hbj940Hi3Y8xQ
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
[root@localhost luna-docker]#
```

2. Check to verify that the Vault server is initialized and auto-unsealed.

```
[root@localhost luna-docker]# ./vault status
Key                      Value
---                      -----
Recovery Seal Type       shamir
Initialized              true
Sealed                   false
Total Recovery Shares    1
Threshold                1
Version                  1.6.2+ent.hsm
Storage Type             file
Cluster Name             vault-cluster-ffa6970c
Cluster ID               1e7d4a5b-3b59-d67f-b659-b6e4c74d1f0a
HA Enabled               false
[root@localhost luna-docker]#
```

Use the secrets engine on host

Secrets engines are components that store, generate, or encrypt data. The KV secrets engine is used to store arbitrary secrets within the configured physical storage for Vault. Versioning can be enabled and a configurable number of versions for each key can be stored. To use the Secrets Engine:

1. Enable the kv engine.

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

2. Execute the following command to view the secrets:

```
# ./vault secrets list
```

```
[root@localhost luna-docker]# ./vault secrets list
Path          Type         Accessor              Description
----          ----         --------              -----------
cubbyhole/    cubbyhole    cubbyhole_a61ae7d0    per-token private secret storage
identity/     identity     identity_5c23a4d0     identity store
kv/           kv           kv_ec32d6f0           n/a
sys/          system       system_5afdb183       system endpoints used for control, policy and debugging
[root@localhost luna-docker]#
```

3. Write arbitrary secret data.

```
# ./vault kv put kv/my-secret my-value=s3cr3t
Success! Data written to: kv/my-secret
```

4. Display the secret value by executing command:

```
# ./vault kv get kv/my-secret
```

```
[root@localhost luna-docker]# ./vault kv get kv/my-secret
====== Data ======
Key          Value
---          -----
my-value     s3cr3t
[root@localhost luna-docker]#
```

You should now be able to successfully run the vault in the container integrated with Luna HSM and also mount the vault binary at host system working without the Luna Client on the host system. You can also mount this vault binary from host to container in any application that requires the vault binary for encrypting/decrypting the secrets.

Use Vault from one container to another

1. Create another container without installing Luna Client and Vault binary using the command below. You had already mounted the vault binary from a container to host and the same binary will now be mounted in the new container.

```
# docker run -it --name vault-service -v /opt/luna-docker/vault:/usr/bin/vault
--net=host centos:centos7
```

Or

```
# docker run -it --name vault-service -v /opt/luna-docker/vault:/usr/bin/vault
--env VAULT_ADDR=http://0.0.0.0:8200 --net=host centos:centos7
```

> **Note:** This is just an example to run the vault service in multiple container without installing the Luna Client and Vault. You can use this as a reference and use the vault integrated with Luna HSM in multiple container.

2. Export the VAULT_ADDR variable in a new container.

```
# export VAULT_ADDR='http://0.0.0.0:8200'
```

> **Note:** Skip this step if you have already set the VAULT_ADDR environment in “docker run” command in step 1.

3. Log in to the Vault in new container.

```
# vault login <VAULT-TOKEN>
```

4. Write arbitrary secret data from the new container.

```
# vault kv put kv/my-secret my-value=C0ntainer
Success! Data written to: kv/my-secret
```

5. Display the secret value by executing the following command in the new container:

```
# vault kv get kv/my-secret
```

```
[root@localhost /]# export VAULT_ADDR='http://0.0.0.0:8200'
[root@localhost /]#
[root@localhost /]# vault login s.KUvvTVHFCLXIqIJCUPUGUeta
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.KUvvTVHFCLXIqIJCUPUGUeta
token_accessor       xpj6dv0AVlcyFJUYEKtvRA1i
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
[root@localhost /]#
[root@localhost /]# vault kv put kv/my-secret my-value=C0ntainer
Success! Data written to: kv/my-secret
[root@localhost /]#
[root@localhost /]# vault kv get kv/my-secret
====== Data ======
Key         Value
---         -----
my-value    C0ntainer
[root@localhost /]#
```

This completes the integration of the HashiCorp Vault runinng inside a Docker container with Luna HSM, using Luna Minimal Client package. The vault service is running in one container and the service at host as well as another container has been exposed that is required to use vault integrated with Luna HSM.

Integrating Vault PKI with Luna HSM using Managed Key

You can integrate Vault with Luna HSM to enable PKI secrets engine to use HSM through managed keys system for key generation, certificate signing, and various other private key related operations.

> **NOTE:** Managed Key with HSM is supported only in vault_1.10.0+ent.hsm or above version.

Vault's managed key system can be used with Luna HSM in both auto-unseal disabled and auto-unseal enabled modes. The managed keys system also works with both vault generated keys and manually generated keys (keys that already exist on the HSM). The steps to integrate Luna HSM with Vault are:

- Configuring Managed Keys on Luna HSM with Auto-Unseal disabled
- Configuring Managed Keys on Luna HSM with Auto-Unseal enabled
- Generating Managed Key within Luna HSM
- Validating the PKI functionality by using the keys generated on Luna HSM

Configuring Managed Keys on Luna HSM with Auto-Unseal disabled

1. Create a Vault configuration file named **managed-key-config.hcl** with the following contents.

```
kms_library "pkcs11" {
  name = "luna"
  library = "/usr/safenet/lunaclient/lib/libCryptoki2_64.so"
}

storage "file" {
  path    = "./vault/data"
}

listener "tcp" {
  address     = "0.0.0.0:8200"
  tls_disable = "true"
}

disable_mlock = true
license_path = "License.txt"

api_addr = "http://127.0.0.1:8200"
cluster_addr = "https://127.0.0.1:8201"
ui = true
```

> **Note**: Vault license must be saved in file let say “License.txt” and the license file path must be provided in “license_path”.

2. Start the Vault using the configuration file.

```
# vault server -config=managed-key-config.hcl
```

3. Open a new terminal window and export the following variables.

```
# export VAULT_ADDR="http://127.0.0.1:8200"
# export PIN=<partition_co_password>
# export LABEL=<partition_label>
```

4. Initialize the Vault if you are running it for the first time.

> **Note:** Ensure to make note of the unseal key and root token which will be use later.

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

```
[root@localhost hashicorp]# vault operator init -key-shares=1 -key-threshold=1
Unseal Key 1: a92ZDTtOkiBCYvoc+FI7dHgQvumMyIYs1UFOtRXQYBw=

Initial Root Token: hvs.DMoxp7O7SdBm7gcrR9YM68n2

Vault initialized with 1 key shares and a key threshold of 1. Please securely
distribute the key shares printed above. When the Vault is re-sealed,
restarted, or stopped, you must supply at least 1 of these keys to unseal it
before it can start servicing requests.
```

5. Unseal the Vault using unseal key, generated when Vault is initialized.

```
# vault operator unseal <unseal key>
```

```
[root@localhost hashicorp]# vault operator unseal a92ZDTtOkiBCYvoc+FI7dHgQvumMyIYs1UFOtRXQYB
w=
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.10.0+ent.hsm
Storage Type    file
Cluster Name    vault-cluster-64e7c24f
Cluster ID      a31a3e6b-0359-a3f5-4642-a70278a16b0c
HA Enabled      false
[root@localhost hashicorp]#
```

6. Log in to Vault.

```
# vault login <root token>
```

```
[root@localhost hashicorp]# vault login hvs.DMoxp7O7SdBm7gcrR9YM68n2
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.DMoxp7O7SdBm7gcrR9YM68n2
token_accessor       opRERuVr1xrty7CXUgEkJraX
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
[root@localhost hashicorp]#
```

Vault is now configured to use Luna HSM for Managed Key.

Configuring Managed Keys on Luna HSM with Auto-Unseal Enabled

1. Create a Vault configuration file **hsm-config.hcl** with the following contents.

```
seal "pkcs11" {
  lib = "/usr/safenet/lunaclient/lib/libCryptoki2_64.so"
  slot = "<slot_id>"
  pin = "<partition_co_password>"
  key_label="hsm-mkek-key1"
  hmac_key_label="hsm-hmac-key1"
  generate_key ="true"
  mechanism=0x1
}

kms_library "pkcs11" {
  name="luna"
  library = "/usr/safenet/lunaclient/lib/libCryptoki2_64.so"
}

storage "file" {
  path    = "./vault/hsm-data"
}

listener "tcp" {
```

```
    address     = "0.0.0.0:8200"
    tls_disable = "true"
  }

  disable_mlock = true
  license_path = "License.txt"

  api_addr = "http://127.0.0.1:8200"
  cluster_addr = "https://127.0.0.1:8201"
  ui = true
```

Note: Save the Vault license in a file, let say “License.txt”. This license file path must be provided in the “license_path”. Start the Vault using the configuration file.

```
# vault server -config=hsm-config.hcl
```

2. Open a new terminal window and export the following variables.

```
# export VAULT_ADDR="http://127.0.0.1:8200"
# export PIN=<partition_co_password>
# export LABEL=<partition_label>
```

3. Initialize the Vault if you are running it for the first time.

Note: Ensure to make note of the root token that will be use later.

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

```
[root@localhost hashicorp]# vault operator init -recovery-shares=1 -recovery-threshold=1
Recovery Key 1: hBVHNy17ERmWWdYZzQXBopXakh0auxVFSYGnC3L/RPc=

Initial Root Token: hvs.BZr39qnZIaP1cEwTZ7fFdxup

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.
[root@localhost hashicorp]#
```

4. Log in to Vault.

```
# vault login <root token>
```

```
[root@localhost hashicorp]# vault login hvs.BZr39qnZIaP1cEwTZ7fFdxup
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.BZr39qnZIaP1cEwTZ7fFdxup
token_accessor       ZgDHt0Xtwt7t5ZhbqXtAEcq6
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
[root@localhost hashicorp]#
```

Vault is now configured to use Luna HSM for Managed Key.

Generating Managed Keys within Luna HSM

Vault supports managed keys generation on Luna HSM in the following ways:

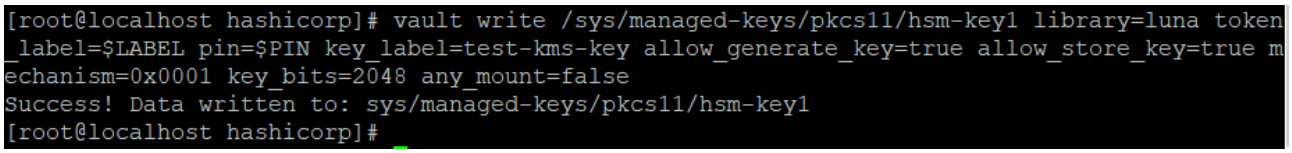
> Generate key directly on Luna HSM

> Use already existing key on Luna HSM

Generating key directly on Luna HSM

To generate key directly, execute the command.

```
# vault write /sys/managed-keys/pkcs11/hsm-key1 library=luna token_label=$LABEL
pin=$PIN key_label=test-kms-key allow_generate_key=true allow_store_key=true
mechanism=0x0001 key_bits=2048 any_mount=false
```



Where:

“library” is the name of the KMS Library defined in the kms_library stanza in the Vault config file.

For more information on the values for “mechanism” see:

https://vault-megr1oalr-hashicorp.vercel.app/docs/configuration/seal/pkcs11#mechanism

> **NOTE:** When auto-unseal is enabled, the slot can be the same as the auto-unseal, but the key label needs to be unique. Alternatively, a different slot altogether can be used.

Using already existing key on Luna HSM

To use the already existing key follow thw steps below.

1. Create a RSA-2048 key manually on Luna HSM.

   ```
   # /usr/safenet/lunaclient/bin/cmu generatekeypair -modulusBits=2048 -
   publicExponent=65537 -label=test-hsm-key -sign=1 -verify=1 -encrypt=1 -
   decrypt=1 -wrap=1 -unwrap=1 -id=c50f7b86372b441ba77cb6f8598f1e35
   ```

2. Now when you have an already existing key on Luna HSM execute the command below using either key_label or key_id.

```
# vault write /sys/managed-keys/pkcs11/hsm-key2 library=luna token_label=$LABEL
pin=$PIN key_label="test-hsm-key" allow_generate_key=false
allow_store_key=false mechanism=0x0001 key_bits=2048 any_mount=false
```

```
[root@localhost hashicorp]# vault write /sys/managed-keys/pkcs11/hsm-key2 library=luna token
_label=$LABEL pin=$PIN key_label="test-hsm-key" allow_generate_key=false allow_store_key=fal
se mechanism=0x0001 key_bits=2048 any_mount=false
Success! Data written to: sys/managed-keys/pkcs11/hsm-key2
[root@localhost hashicorp]#
```

NOTE: You only need to use either key_label or key_id in the config. This will depend on how you manually create the key.

```
# vault write /sys/managed-keys/pkcs11/hsm-key3 library=luna token_label=$LABEL
pin=$PIN key_id=c50f7b86372b441ba77cb6f8598f1e35 allow_generate_key=false
allow_store_key=false mechanism=0x0001 key_bits=2048 any_mount=false
```

```
[root@localhost hashicorp]# vault write /sys/managed-keys/pkcs11/hsm-key3 library=luna token
_label=$LABEL pin=$PIN key_id=c50f7b86372b441ba77cb6f8598f1e35 allow_generate_key=false allo
w_store_key=false mechanism=0x0001 key_bits=2048 any_mount=false
Success! Data written to: sys/managed-keys/pkcs11/hsm-key3
[root@localhost hashicorp]#
```

a. Verify the config has been written to Vault.

```
# vault list /sys/managed-keys/pkcs11
```

```
[root@localhost hashicorp]# vault list /sys/managed-keys/pkcs11
Keys
----
hsm-key1
hsm-key2
hsm-key3
[root@localhost hashicorp]#
```

b. Verify that a key config is valid by test signing some data.

```
# vault write -f /sys/managed-keys/pkcs11/hsm-key1/test/sign
```

```
[root@localhost hashicorp]# vault write -f /sys/managed-keys/pkcs11/hsm-key1/test/sign
Success! Data written to: sys/managed-keys/pkcs11/hsm-key1/test/sign
[root@localhost hashicorp]#
```

NOTE: Where hsm-key1 is the key name defined in the Vault Write command.

This completes the generation of Managed Key on Luna HSM which can be used by Vault PKI Engine to perform crypto operations.

Validating PKI Functionality Using Managed Key Generated on Luna HSM

Perform the steps below to generate the Root CA and intermediate CA using managed key on Luna HSM and sign the leaf certificate using the keys created on Luna HSM.

1. Create a managed key for the Root CA

```
# vault write /sys/managed-keys/pkcs11/hsm-key-root library=luna
token_label=$LABEL pin=$PIN key_label="hsm-key-root" allow_generate_key=true
allow_store_key=true mechanism=0x0001 key_bits=2048 any_mount=false
```

```
[root@localhost hashicorp]# vault write /sys/managed-keys/pkcs11/hsm-key-root library=luna t
oken_label=$LABEL pin=$PIN key_label="hsm-key-root" allow_generate_key=true allow_store_key=
true mechanism=0x0001 key_bits=2048 any_mount=false
Success! Data written to: sys/managed-keys/pkcs11/hsm-key-root
[root@localhost hashicorp]#
```

2. Create a managed key for the Intermediate CA

```
# vault write /sys/managed-keys/pkcs11/hsm-key-int library=luna
token_label=$LABEL pin=$PIN key_label="hsm-key-int" allow_generate_key=true
allow_store_key=true mechanism=0x0001 key_bits=2048 any_mount=false
```

```
[root@localhost hashicorp]# vault write /sys/managed-keys/pkcs11/hsm-key-int library=luna to
ken_label=$LABEL pin=$PIN key_label="hsm-key-int" allow_generate_key=true allow_store_key=tr
ue mechanism=0x0001 key_bits=2048 any_mount=false
Success! Data written to: sys/managed-keys/pkcs11/hsm-key-int
[root@localhost hashicorp]#
```

3. Enable PKI secrets engine for the Root.

```
# vault secrets enable -path=pki -allowed-managed-keys=hsm-key-root pki
```

```
[root@localhost hashicorp]# vault secrets enable -path=pki -allowed-managed-keys=hsm-key-roo
t pki
Success! Enabled the pki secrets engine at: pki/
[root@localhost hashicorp]#
```

4. Enable intermediate secrets engine.

```
# vault secrets enable -path=pki_int -allowed-managed-keys=hsm-key-int pki
```

```
[root@localhost hashicorp]# vault secrets enable -path=pki_int -allowed-managed-keys=hsm-key
-int pki
Success! Enabled the pki secrets engine at: pki_int/
[root@localhost hashicorp]#
```

5. Create a Root CA Certificate with a managed key on Luna HSM.

```
# vault write -field=certificate pki/root/generate/kms managed_key_name=hsm-
key-root common_name=example.com ttl=8760h > /tmp/CA_cert.crt
```

6. Verify the certificate.

```
# cat /tmp/CA_cert.crt
```

```
[root@localhost hashicorp]# cat /tmp/CA_cert.crt
-----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIUclglytUABAd27q3F/4bJ5x4eCd4wDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLZXhhbXBsZS5jb20wHhcNMjIwNTA2MTAzODAyWhcNMjIw
NjA3MTAzODMyWjAWMRQwEgYDVQQDEwtleGFtcGxlLmNvbTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBANpY4WvXFaniO0OzDIJkIspLRfjXYeLKTPtIUBQB
8ZleAeOz2PlSFUZFjAd3m6W9fcAo1k+Rj01nRFWycfj2OwxIAHVOp273GmGxqYss
YO+6UNz0wSi1zAya2k318xlnZphVoDX02EJfEVIXzYU8hWF6xBwknsqZZ2gIqHo5
dZci6bJudoFo4jbeNUSf/cWoOv2jC8gFXqotRJ6oBhSRzxJbes4drfmk3nFAZAj+
UtmZC6J6Vp36lORn9k/ti0WKGOCr9zoLUZSpc0/AlM8RCbcEXBmfrYZ8jGnykohc
5Q1QvT9EjVg0kdOeXSZil1l1VMzsSIdTaC45tek/VqWgSVECAwEAAaN7MHkwDgYD
VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFMdGu9kpz070
rNAH0ng8Za8Ab2vyMB8GA1UdIwQYMBaAFMdGu9kpz070rNAH0ng8Za8Ab2vyMBYG
A1UdEQQPMA2CC2V4YW1wbGUuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQB2vzimF+VA
c0yu6lPgmrcfFnuVeXdsh+kEEeS7pYCz7WtWvvn3uucyGxYnPynKzo3GHJwS4Soz
/H15MpVO7U96FXVZKDNR9eSBNTyWI88vVCjMUcLO6Bw5Tbhn2OOBEp5/LA5ISpH0
1NIGHQ42U+ul/CSKD9X3eYAL2UIDf9yaNBIzYB6BdM1EQ9MLh4x4T39AmzPdHJWq
01YDIverjU7730PiUVPHdI1IdKyiMNguyvBtQjOCMnVEf1crandQeP8t6asQsNqX
HTNJFe0zkybdwSZovSgekzKEH97wmpDA5nEf43d2EpDtdQbeFeA4GNn5fkJ7uG5B
07/HbX4yMOod
-----END CERTIFICATE-----[root@localhost hashicorp]#
```

7. Create an Intermediate CA CSR with the managed key on Luna HSM.

```
# vault write -format=json pki_int/intermediate/generate/kms
managed_key_name=hsm-key-int common_name="example.com" | jq -r '.data.csr' >
/tmp/pki_intermediate.csr
```

8. Verify the CSR looks good.

```
# cat /tmp/pki_intermediate.csr
```

```
[root@localhost hashicorp]# cat /tmp/pki_intermediate.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIChDCCAWwCAQAwFjEUMBIGA1UEAxMLZXhhbXBsZS5jb20wggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDAyRLoAtgKgTHjyoHPahFs3fDsGmBLst/tFSG9
e4NmRk69STcQyoBNmBDwPhMUSrwr5POwjRYLhAVUtiVjalrFoNYdFza3r4evXkOs
qQNsd7dLCnn2qbLvbI3EfWOItui2tNrudi2KQg/JP/+t+KrPuIelgTSbBADcDSKz
MVLqm3kybipegu3JypElupKNkJrqyvEDymQHxDUK30irxEIyg2n1eOMbKi9VPxB7
HI2sHH4Vl7J+VPcbqnUUSZG9+K/dMNKHvr+Twk55p5R4QbJESHlm489314opkBso
OCcoYDGjLPK/7+V3XebLv0HWhoqAKy60IGX4oY/AVXWvxig7AgMBAAGgKTAnBgkq
hkiG9w0BCQ4xGjAYMBYGA1UdEQQPMA2CC2V4YW1wbGUuY29tMA0GCSqGSIb3DQEB
CwUAA4IBAQCQMLA7x7EzDGJBgKYlsfBuQVtjxRvacsI9mML/6ftXejD9b71/cEpj
rBWbk69i8g5G9gY0e1C+jwLXrcyxdtBXIAUBBX/Ph8p3DpsEhTcNS10GQ9pVAnC5
MvalGfO0SrSaNP8hzOF6pesCRayV2QJaUpsPPAZkjsnLfqRFM1nPK1xl+viGacAM
kIgUZt2d6I6sO8+K/hefv5ex6ejaV7PbOqbZXwGyNGXSbwaxD/NzODkZI0Be3531
Gku0M1C1FCLR0+WhrQyQ9NSCOmR9irfZgI83Hc0G/ZE/ZRJS4XbESmJcnx9NV3PX
o9PgZFHJbtFBGNwPObuikCUd16JtSXcH
-----END CERTIFICATE REQUEST-----
[root@localhost hashicorp]#
```

9. Sign the intermediate CA CSR with managed root CA.

```
# vault write -format=json pki/root/sign-intermediate
csr=@/tmp/pki_intermediate.csr format=pem_bundle ttl="43800h" | jq -r
'.data.certificate' > /tmp/intermediate.cert.pem
```

10. Create an Intermediate CA certificate in Managed Key.

```
# vault write pki_int/intermediate/set-signed
certificate=@/tmp/intermediate.cert.pem
```

```
[root@localhost hashicorp]# vault write pki_int/intermediate/set-signed certificate=@/tmp/in
termediate.cert.pem
Success! Data written to: pki_int/intermediate/set-signed
[root@localhost hashicorp]#
```

11. Create a role to Issue a leaf certificate from the intermediate key.

```
# vault write pki_int/roles/example-dot-com allowed_domains="example.com"
allow_subdomains=true max_ttl="720h"
```

```
[root@localhost hashicorp]# vault write pki_int/roles/example-dot-com allowed_domains="examp
le.com" allow_subdomains=true max_ttl="720h"
Success! Data written to: pki_int/roles/example-dot-com
[root@localhost hashicorp]#
```

12. Issue leaf certificate.

```
# vault write -format=json pki_int/issue/example-dot-com
common_name="test.example.com" ttl="24h"
```

This completes the Luna HSM integration with Vault PKI using Managed Key.

Contacting Customer Support

If you encounter a problem while installing, registering, or operating this product, refer to the documentation. If you cannot resolve the issue, contact your supplier or Thales Customer Support. Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Thales and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is a database where you can find solutions for most common problems. The Customer Support Portal is a comprehensive, fully searchable repository of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more. You can also use the portal to create and manage support cases.

> **NOTE:** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone Support

If you have an urgent problem, or cannot access the Customer Support Portal, you can contact Thales Customer Support by telephone at +1 410-931-7520. Additional local telephone support numbers are listed on the support portal.

Email Support

You can also contact technical support by email at technical.support.DIS@thalesgroup.com.