Fortanix®

Integration Guide

USING FORTANIX SELF-DEFENDING KMS FOR HASHICORP VAULT ENTERPRISE

Fortanix Inc. | 800 West El Camino Real | Suite 180 | Mountain View, CA 94040 | United States of America |
+1 (628) 400-2043 | info@fortanix.com | www.fortanix.com

1.0 INTRODUCTION

This article describes how to integrate **Fortanix Self-Defending Key Management Service (KMS) with HashiCorp Vault Enterprise** and secure it by protecting its master key through an additional HSM key. It also contains the information that a user requires to:

- Deploy the Fortanix PKCS#11 library for Vault
- Enable Vault seal wrapping with an Hardware Security Module (HSM) PKCS11 token
- Test the Vault server
- Rotate the Vault master key

1.1 WHY USE FORTANIX SELF-DEFENDING KMS WITH HASHICORP VAULT ENTERPRISE

The Fortanix Self-Defending KMS with HashiCorp Vault Enterprise integration greatly simplifies the Vault administration. It is a must-have for any HashiCorp Vault Enterprise deployment within an organization storing its sensitive information inside Vault.

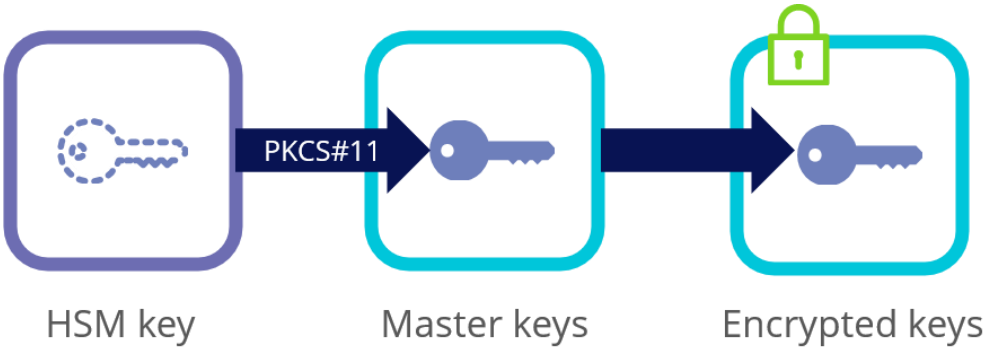

FIGURE 1: FORTANIX SELF-DEFENDING KMS SOLUTION FOR HASHICORP VAULT

In summary, Fortanix **Self-Defending KMS** can harden and secure HashiCorp Vault by:

1. **Master Key Wrapping**: The Vault master key is protected by transiting it through the Fortanix HSM for encryption rather than having it split into key shares.
2. **Automatic Unsealing**: Vault stores its encrypted master key in storage, allowing for automatic unsealing, which is decrypted only through authorized HSM access.
3. **Seal Wrapping:** This provides **FIPS 140-2 Level 2** secret storage conforming functionality for Critical Security Parameters. Note, that the Fortanix **Self-Defending KMS** itself is classified by **NIST** as a **FIPS 140-2 Level 3** HSM.
4. **Entropy Augmentation:** Vault Enterprise features a mechanism to sample entropy (or randomness for cryptographic operations) from external cryptographic modules through

the Seals interface. While the system entropy used by Vault is more than capable of operating in most threat models, there are some situations where additional entropy from hardware-based random number generators are desirable.

Fortanix Self-Defending KMS can harden existing as well as new HashiCorp Vault Enterprise deployments regardless of their current seal configuration.

2.0 PREREQUISITES

- HashiCorp Vault Enterprise HSM
- Fortanix **Self-Defending KMS**
- Network route between Vault Enterprise and Fortanix Self-Defending KMS

3.0 SETUP

3.1 DEPLOY THE FORTANIX PKCS#11 LIBRARY FOR VAULT

1. Install the PKCS#11 library by following the instructions provided in the following URL:

https://support.fortanix.com/hc/en-us/articles/360016160451-Clients-PKCS-11-Library

NOTE: HashiCorp Vault Enterprise HSM is only supported on Linux.

2. Configure the Fortanix PKCS#11 library using the following commands:

```
export FORTANIX_API_ENDPOINT = "https://sdkms.fortanix.com"
export FORTANIX_P11LIB =/opt/fortanix/pkcs11/fortanix_pkcs11.so
```

3. [**OPTIONAL**] - Configure extra variables to test the Fortanix library using the following commands:

```
export FORTANIX_API_KEY = "<<FORTNAIX_API_KEY>>"
export FORTANIX_PKCS11_NUM_SLOTS = 5
export FORTANIX_PKCS11_LOG_LEVEL = “debug”
```

4. [**OPTIONAL**] - Validate that the Fortanix PKCS#11 library is functional using the following commands:

```
$> pkcs11-tool --module /path/to/fortanix_pkcs11.so --show-info

  Cryptoki version 2.40
  Manufacturer     Fortanix
```

```
 Library          Fortanix PKCS#11 3.23.1408 (ver 3.23)
Using slot 0 with a present token (0x0)
```

3.2 ENABLE VAULT SEAL WRAPPING WITH AN HSM PKCS11 TOKEN

1. Create a Vault configuration file called `cfg-vault-p11.hcl`:

```
 seal "pkcs11" {
   lib            = "/opt/fortanix/pkcs11/fortanix_pkcs11.so"
   slot           = "0"
   pin            = "<<FORTANIX_API_KEY>>"
   key_label      = " Vault-Master-Key-RSA-OAEP-SHA256"
   mechanism      = "0x0009"
   rsa_oaep_hash  = "sha256"
   generate_key   = "true"
}
```

NOTE:

- Create an App in **Self-Defending KMS** dedicated to the Vault Enterprise deployment.
- Use different Fortanix Apps corresponding to each Vault server, to streamline the audit reporting for later compliance.
- Specify the `FORTANIX_API_KEY` in the Vault config file shown above.
- Additional Vault configuration parameters are available at – https://www.vaultproject.io/docs/configuration/seal/pkcs11
- Additional master key encryption mechanisms supported are:
    - `0x0009` corresponds to `CKM_RSA_PKCS_OAEP`
    - `0x1087` corresponds to `CKM_AES_GCM`
    - `0x1082` corresponds to `CKM_AES_CBC` – This mode is not supported for Fortanix HSM Gateway.
- Vault creates non-exportable keys if **`generate_key="true"`**. Otherwise, please create the Fortanix Self-Defending KMS and specify the correct key label with **`generate_key="false"`**.

2. Vault Enterprise is now ready for initialization and testing. Verify the log:

```
$> vault server -config=cfg-vault-p11.hcl
```

```
==> Vault server configuration:

     HSM PKCS#11 Version: 2.40
             HSM Library: Fortanix PKCS#11 3.23.1408
     HSM Library Version: 3.23
     HSM Manufacturer ID: Fortanix
               HSM Type: pkcs11
                    Cgo: enabled
             Go Version: go1.14.4
                 Listener 1: tcp (addr: "0.0.0.0:8200", cluster
 address: "0.0.0.0:8201", max_request_duration: "1m30s",
 max_request_size: "33554432", tls: "disabled")
              Log Level: trace
                  Mlock: supported: true, enabled: false
        Recovery Mode: false
              Storage: file
              Version: Vault v1.6.0+ent.hsm

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

2020-07-30T13:17:15.405-0400 [INFO]  proxy environment: http_proxy=
https_proxy= no_proxy=
2020-07-30T13:17:16.988-0400 [INFO]  core: stored unseal keys
supported, attempting fetch
2020-07-30T13:17:16.988-0400 [WARN]  failed to unseal core:
error="stored unseal keys are supported, but none were found"
```

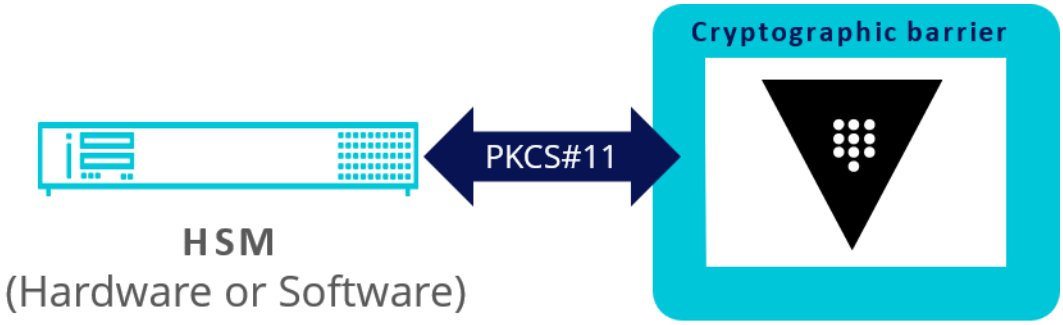

FIGURE 2: VAULT CONFIGURATION

3. Verify the status of Vault using 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
```

4.0 TEST THE VAULT SERVER

Once the Vault server is running, without an Enterprise or a Trial License, it terminates in 30 minutes. To continue testing, please restart the Vault server.

4.1 INITIALIZE AND UNSEAL THE VAULT

With configuration done, Vault needs to be initialized and unsealed before use using the following command:

```
$> vault operator init -recovery-shares=3 -recovery-threshold=2

Recovery Key 1: /Ui9pazFsRI/ObpGAietRaakNiM8nj6s9SgCTDs2fVOT
Recovery Key 2: yJUsNBL4bjWeQidHIRJBE+tB+WObbTZTlzzIRN3AsfQ+
Recovery Key 3: URVrc2l7ZROFl3ePCotVceSHxrMGGSwwpAUyXlhxzWky

Initial Root Token: s.sb11N0r9HJrCyZHNrfb0YIti

Success! Vault is initialized

Recovery key initialized with 3 key shares and a key threshold of 2.
Please securely distribute the key shares printed above.
```

The previous step connects to Fortanix Self-Defending KMS and auto-unseals Vault by having the master key decrypted inside the HSM.

NOTE:

- If there are problems with the previous command, please ensure that `Rsyslogd` is enabled by verifying `/var/log/syslog` (Debian) or `/var/log/messages` (RHEL).
- Start the Vault server with additional debugging as follows:

```
$> FORTANIX_PKCS11_LOG_LEVEL=debug vault server -log-level=trace -
config=cfg-vault-p11.hcl

==> Vault server configuration:

     HSM PKCS#11 Version: 2.40
             HSM Library: Fortanix PKCS#11 3.23.1408
     HSM Library Version: 3.23
     HSM Manufacturer ID: Fortanix
                HSM Type: pkcs11
                     Cgo: enabled
              Go Version: go1.14.4
                 Listener 1: tcp (addr: "0.0.0.0:8200", cluster
 address: "0.0.0.0:8201", max_request_duration: "1m30s",
 max_request_size: "33554432", tls: "disabled")
              Log Level: trace
                  Mlock: supported: true, enabled: false
          Recovery Mode: false
                Storage: file
                Version: Vault v1.6.0+ent.hsm

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

2020-07-30T13:17:15.405-0400 [INFO]  proxy environment: http_proxy=
https_proxy= no_proxy=
2020-07-30T13:17:15.408-0400 [TRACE] seal.pkcs11: pkcs11 mechanism
selected: mechanism=0x1087 name=aes-gcm
2020-07-30T13:17:16.959-0400 [TRACE] seal.pkcs11: key successfully
found
```

```
2020-07-30T13:17:16.975-0400 [WARN]  no `api_addr` value specified in
config or in VAULT_API_ADDR; falling back to detection if possible,
but this value should be manually set
2020-07-30T13:17:16.976-0400 [TRACE] core: activating sealwrap
capability
2020-07-30T13:17:16.976-0400 [TRACE] core: initializing licensing
2020-07-30T13:17:16.982-0400 [DEBUG] storage.cache: creating LRU
cache: size=0
2020-07-30T13:17:16.985-0400 [DEBUG] cluster listener addresses
synthesized: cluster_addresses=[0.0.0.0:8201]
2020-07-30T13:17:16.988-0400 [INFO]  core: stored unseal keys
supported, attempting fetch
2020-07-30T13:17:16.988-0400 [WARN]  failed to unseal core:
error="stored unseal keys are supported, but none were found"
```

In addition to the Vault server transcript, the Syslog or messages log should reveal more details pertaining to connection to the Fortanix Self-Defending KMS or other PKCS#11 errors, if any.

4.2 VERIFY THE STATUS OF THE VAULT

Verify the status of the vault using the following command:

```
$> vault status

Key                     Value
---                     -----
Recovery Seal Type      shamir
Initialized             true
Sealed                  false
Total Recovery Shares   3
Threshold               2
Version                 1.6.0+ent.hsm
Cluster Name            vault-cluster-dfcace02
Cluster ID              0fe5ab51-ff92-fdb0-9c37-f739679554f8
HA Enabled              false
```

4.3 RESTART VAULT

Restart Vault and check the auto-unseal status before continuing using the following command:

```
$> vault login
Token (will be hidden):
Success! You are now authenticated...

$> vault secrets enable kv -path=morekv
$> vault kv put morekv/hello foo=world

Success! Data written to: morekv/hello

$> vault kv get morekv/hello

=== Data ===
Key    Value
---    -----
foo    world
```

4.4 TEST SEAL WRAPPING

1. Apply the Enterprise license.

```
vault write sys/license text=<license>
```

NOTE: For FIPS 140-2 compliance, seal wrap requires FIPS 140-2 Certified HSM which is supported by Vault Enterprise.

For some values, seal wrapping is always enabled including the recovery key, any stored key shares, the master key, the keyring, and more. When working with the key/value secrets engine, you can enable seal wrap to wrap all data.

To compare seal wrapped data against unwrapped data, enable key/value v1 secrets engine at two different paths: `kv-unwrapped` and `kv-seal-wrapped`.

2. Enable k/v v1 without seal wrap at `kv-unwrapped`.

```
vault secrets enable -path=kv-unwrapped kv
```

3. Enable k/v v1 with seal wrap. To do so, use the '`-seal-wrap`' flag when you enable the KV workflow.

```
vault secrets enable -path=kv-seal-wrapped -seal-wrap kv
```

4. List the enabled secrets engines with details.

```
vault secrets list -detailed
Path                Plugin      Accessor              Seal Wrap
----                ------      --------              ---------
cubbyhole/          cubbyhole   cubbyhole_3a415617    false
identity/           identity    identity_b544f633     false
kv-seal-wrapped/    kv          kv_72d69c39           true
kv-unwrapped/       kv          kv_2e60002d           false
```

NOTE: Notice that the `Seal Wrap` parameter value is `true` for `kv-seal-wrapped`.

4.4.1 EXAMPLE TO TEST THE SEAL WRAP FEATURE

1. Write a secret at `kv-unwrapped/unwrapped` for testing.

```
vault kv put kv-unwrapped/unwrapped password="my-long-
password"
```

2. Read the path to verify.

```
vault kv get kv-unwrapped/unwrapped


====== Data ======
Key          Value
---          -----
password     my-long-password
```

3. Write the same secret at `kv-seal-wrapped/wrapped` for testing.

```
vault kv put kv-seal-wrapped/wrapped password="my-long-
password"
```

4. Read the path to verify.

```
vault kv get kv-seal-wrapped/wrapped


====== Data ======
Key          Value
---          -----
password     my-long-password
```

Using a valid token, you can write and read secrets the same way regardless of the seal wrap.

NOTE: Remember that the Vault server was configured to use the local file system (`/tmp/vault`) as its storage backend in this example.

5. SSH into the machine where the Vault server is running and check the stored values in the `/tmp/vault` directory.

```
cd /tmp/vault/logical
```

6. Under the `/tmp/vault/logical` directory, there are two sub-directories. One maps to `kv-unwrapped/` and another maps to `kv-seal-wrapped/` although you cannot tell by the folder names.
7. View the secret at rest. One of the directories maps to `kv-unwrapped/unwrapped`.

```
cd 2da357cd-55f2-7eed-c46e-c477b70bed18
```

8. View its content. The password value is encrypted.

```
cat _unwrapped
{"Value":"AAAAAQICk547prhuhMiBXLq2lx8ZkMpSB3p+GKHAwuMhKrZGS
eqsFevMS6YoqTVlbvpU9B4zWPZ2HA
SeNZ3YMw=="}
```

9. Another directory maps to `kv-seal-wrapped/wrapped`.

```
cd ../5bcea44d-28a3-87af-393b-c6d398fe41d8
```

10. View its content. The password value is encrypted.

```
cat _wrapped
{"Value":"ClBAg9oN7zBBaDBZcsilDAyGkL7soPe7vBA5+ADADuyzo8GuH
ZHb9UFN2nF1h0OpKEgCIkG3JNHcXt
tZqCi6szcuNBgF3pwhWGwB4FREM3b5CRIQYK7239Q92gRGrcBBeZD6ghogE
tSBDmZJBahk7n4lIYF3X4iBqmwZgH
Vo4lzWur7rzncgASofCIIhENEEGghoc21fZGVtbyINaHNtX2htYWNfZGVtb
3M="}
```

NOTE: Secrets are encrypted regardless; however, the seal-wrapped value is significantly longer despite the fact that both values are the same, `my-long-password`.

Looking closely at the values in the last two outputs, you will see a slight difference.

- The first output shows the truncated secret encrypted with Vault's encryption key which is a standard across anything that passes Vault's barrier and is stored in its storage backend.
- In the second output, the value looks different, as expected. This is because Vault has applied a Seal Wrap to it, which means that it was both encrypted by Vault and encrypted by the HSM before being stored in Vault's storage backend.

4.5 ENTROPY AUGMENTATION

1. Apply the Enterprise license.

```
vault write sys/license text=<license>
```

2. Entropy augmentation is disabled by default. To enable entropy augmentation, Vault's configuration file must include a properly configured entropy and seal stanza for a supported seal type. For example:

```
seal "pkcs11" {
    ...
}

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

3. Enable Entropy Augmentation.

   To leverage the external entropy source, set the `external_entropy_access` parameter to `true` when you enable a secrets engine or auth method.

   In this step, you are going to enable an external entropy source on a transit secrets engine. Execute the following command to enable transit secrets engine with external entropy source using the `-external-entropy-access` flag.

```
vault secrets enable -external-entropy-access transit
```

4. List the enabled secrets engines with `-detailed` flag.

```
vault secrets list -detailed
Path         Plugin     Accessor            External Entropy Access
----         ------     --------            -----------------------
cubbyhole/   cubbyhole  cubbyhole_b8e95763  false
identity/    identity   identity_b629a5f8   false
sys/         system     system_45f38555     false
transit/     transit    transit_b93cf7a7    true
```

NOTE: Notice that the External Entropy Access is set to `true` for `transit/`.

4.5.1 EXAMPLE TO TEST THE ENTROPY AUGMENTATION FEATURE

You can start using the transit secrets engine to encrypt your sensitive data which leverages the HSM as its external entropy source. Regardless, the user experience remains the same as before.

1. Create a new encryption key named, "`orders`".

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

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

2. 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:v1:4oQdMopYk805wCww5t+5mwn6hmTy1FvfMiGHfsftc8lxD
             4YdkgW3RZUNymISzhCE

key_version 1
```

3. Now, test to verify that you can decrypt.

```
/vault write transit/decrypt/orders
ciphertext="vault:v1:4oQdMopYk805wCww5t+5mwn6hmTy1FvfMiGHfs
ftc8lxD4YdkgW3RZUNymISzhCE"

Key          Value
---          -----
plaintext    NDExMSAxMTExIDExMTEgMTExMQo=
```

4. Decode to get the original data.

```
base64 --decode <<< NDExMSAxMTExIDExMTEgMTExMQo=
4111 1111 1111 1111
```

NOTE: When the external entropy access is enabled, connectivity to the HSM is required. If the HSM becomes unreachable for any reason, the transit secrets engine will fail to generate new keys or rotate the existing keys.

```
Error writing data to transit/encrypt/orders: Error making API
request.

URL: PUT http://127.0.0.1:8200/v1/transit/encrypt/orders
```

```
Code: 400. Errors:

* error performing token check: failed to read entry: error
initializing session
for decryption: error logging in to HSM: pkcs11: 0xE0:
CKR_TOKEN_NOT_PRESENT
```

5.0 ROTATE MASTER KEY

For existing **Vault** Enterprise deployments where the Vault server is running with PKCS#11 auto-unseal with another HSM vendor's PKCS#11 library, this section may be useful to migrate to the Fortanix **HSM Gateway**.

In addition, this section is also relevant to new or existing HashiCorp Vault deployments, wherein the master key needs to be rotated simply inside the Fortanix **Self-Defending KMS**.

5.1 ROTATE MASTER KEY

With configuration and end-to-end testing complete, Vault can have its master key rotated by specifying a default key that is used as a fallback.

1. Shutdown the Vault server.
2. Update the Vault configuration file called `cfg-vault-p11.hcl` using the following command:

```
seal "pkcs11" {
   lib             = "/opt/fortanix/pkcs11/fortanix_pkcs11.so"
   slot            = "0"
   pin             = "<<FORTANIX_API_KEY>>"
   key_label          = "New-Vault-Master-Key-RSA-OAEP-SHA256"
   default_key_label = "Vault-Master-Key-RSA-OAEP-SHA256"
   mechanism       = "0x0009"
   rsa_oaep_hash   = "sha256"
   generate_key    = "true"
}
```

In this example:

- `Vault-Master-Key-RSA-OAEP-SHA256` is the existing key.
- `New-Vault-Master-Key-RSA-OAEP-SHA256` is the new key.

5.2 RESTART VAULT

Restart vault using the following command:

```
$> vault server -config=cfg-vault-p11.hcl

==> Vault server configuration:

     HSM PKCS#11 Version: 2.40
             HSM Library: Fortanix PKCS#11 3.23.1408
     HSM Library Version: 3.23
     HSM Manufacturer ID: Fortanix
                HSM Type: pkcs11
                     Cgo: enabled
              Go Version: go1.14.4
                  Listener 1: tcp (addr: "0.0.0.0:8200", cluster
 address: "0.0.0.0:8201", max_request_duration: "1m30s",
 max_request_size: "33554432", tls: "disabled")
               Log Level: trace
                   Mlock: supported: true, enabled: false
           Recovery Mode: false
                 Storage: file
                 Version: Vault v1.6.0+ent.hsm

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

2020-07-30T14:23:55.605-0400 [INFO]  proxy environment: http_proxy=
https_proxy= no_proxy=
2020-07-30T14:23:57.414-0400 [WARN]  no `api_addr` value specified in
config or in VAULT_API_ADDR; falling back to detection if possible,
but this value should be manually set
2020-07-30T14:23:57.447-0400 [INFO]  core: stored unseal keys
supported, attempting fetch
2020-07-30T14:24:00.524-0400 [INFO]  core.cluster-listener.tcp:
starting listener: listener_address=0.0.0.0:8201
2020-07-30T14:24:00.526-0400 [INFO]  core.cluster-listener: serving
cluster requests: cluster_listen_address=[::]:8201
2020-07-30T14:24:00.527-0400 [INFO]  core: post-unseal setup starting
2020-07-30T14:24:08.122-0400 [INFO]  core: loaded wrapping token key
```

```
2020-07-30T14:24:08.123-0400 [INFO]  core: successfully setup plugin
catalog: plugin-directory=
2020-07-30T14:24:08.128-0400 [INFO]  core: successfully mounted
backend: type=system path=sys/
2020-07-30T14:24:08.129-0400 [INFO]  core: successfully mounted
backend: type=identity path=identity/
2020-07-30T14:24:08.130-0400 [INFO]  core: successfully mounted
backend: type=kv path=morekv/
2020-07-30T14:24:08.130-0400 [INFO]  core: successfully mounted
backend: type=cubbyhole path=cubbyhole/
2020-07-30T14:24:08.156-0400 [INFO]  core: successfully enabled
credential backend: type=token path=token/
2020-07-30T14:24:08.158-0400 [INFO]  core: restoring leases
2020-07-30T14:24:08.158-0400 [INFO]  rollback: starting rollback
manager
2020-07-30T14:24:08.161-0400 [INFO]  expiration: lease restore
complete
2020-07-30T14:24:08.161-0400 [INFO]  identity: entities restored
2020-07-30T14:24:08.162-0400 [INFO]  identity: groups restored
2020-07-30T14:24:08.162-0400 [INFO]  mfa: configurations restored
2020-07-30T14:24:08.170-0400 [INFO]  core: stopping replication
2020-07-30T14:24:08.170-0400 [INFO]  core: closed sync connection
2020-07-30T14:24:08.171-0400 [INFO]  core: replication stopped
2020-07-30T14:24:08.171-0400 [INFO]  core: setting up replication
2020-07-30T14:24:08.172-0400 [INFO]  core: replicated cluster
information not found or disabled, not activating client
2020-07-30T14:24:08.172-0400 [INFO]  core: replication setup finished
2020-07-30T14:24:09.830-0400 [INFO]  core.autoseal: upgrading
recovery key
2020-07-30T14:24:13.799-0400 [INFO]  core.autoseal: upgrading stored
keys
2020-07-30T14:24:17.223-0400 [INFO]  core: usage gauge collection is
disabled
2020-07-30T14:24:17.225-0400 [INFO]  core: post-unseal setup complete
2020-07-30T14:24:17.226-0400 [INFO]  core: vault is unsealed
2020-07-30T14:24:17.226-0400 [INFO]  core: unsealed with stored keys:
stored_keys_used=1
```

5.3 VERIFY THE STATUS OF VAULT

Verify the status of vault using the following command:

```
$> vault status

Key                     Value
---                     -----
Recovery Seal Type      shamir
Initialized             true
Sealed                  false
Total Recovery Shares   3
Threshold               2
Version                 1.6.0+ent.hsm
Cluster Name            vault-cluster-dfcace02
Cluster ID              0fe5ab51-ff92-fdb0-9c37-f739679554f8
HA Enabled              false
```

5.4 CHECK AND UPDATE THE SECRET

Check the secret and update it using the following command:

```
$> vault login
Token (will be hidden):
Success! You are now authenticated...

$> vault kv put morekv/hello foo=new-world
Success! Data written to: morekv/hello

$> vault kv get morekv/hello

=== Data ===
Key    Value
---    -----
foo    new-world
```

6.0 HSM GATEWAY

Notes on testing HashiCorp Vault Enterprise with an existing HSM from Thales, Entrust or Gemalto:

- Ensure in Fortanix Self-Defending KMS that a HSM-enabled Group is successfully connected to a Fortanix HSM Gateway.
- Use an appropriate **`FORTANIX_API_KEY`** corresponding to the external HSM-enabled Group.
- Check the HSM for a list of cryptographic mechanisms supported. Some HSMs have disabled certain mechanisms like **`CKM_AES_GCM`**.
- Fortanix HSM Gateway with Fortanix Self-Defending KMS has been successfully tested with a Vault Enterprise master **`CKM_RSA_PKCS_OAEP`**.

7.0 REFERENCES

- https://support.fortanix.com/hc/en-us/articles/360018312391-PKCS-11
- https://www.vaultproject.io/docs/enterprise/hsm
- https://www.vaultproject.io/docs/configuration/seal/pkcs11
- https://learn.hashicorp.com/vault/operations/ops-seal-wrap
- https://learn.hashicorp.com/vault/getting-started/first-secret

8.0 DOCUMENT INFORMATION

8.1 DOCUMENT LOCATION

The latest published version of this document is located at the URL:

https://support.fortanix.com/hc/en-us/articles/360046779472-Using-Fortanix-Self-Defending-KMS-with-HashiCorp-Vault-Enterprise

8.2 DOCUMENT UPDATES

This document will typically be updated on a periodic review and update cycle.

For any urgent document updates, please send an email to: support@fortanix.com