HASHICORP VAULT

Integration Guide

Applicable Devices:
Vectera Plus

TABLE OF CONTENTS

[1] DOCUMENT INFORMATION

[1.1] DOCUMENT OVERVIEW

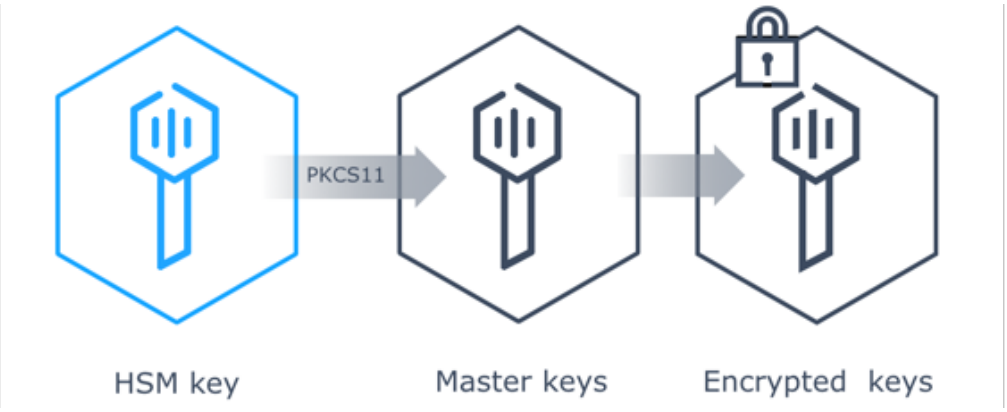
The purpose of this document is to provide information regarding the configuration of Futurex HSMs with HashiCorp Vault using PKCS #11 libraries. For additional questions related to your HSM, see the relevant administrator's guide.

[1.2] APPLICATION DESCRIPTION

Vault Enterprise integrates with Hardware Security Module (HSM) platforms to provide four pieces of special functionality:

- Master Key Wrapping: Vault protects its master key by transiting it through the HSM for encryption rather than splitting into key shares
- Automatic Unsealing: Vault stores its encrypted master key in storage, allowing for automatic unsealing
- Seal Wrapping to provide FIPS KeyStorage-conforming functionality for Critical Security Parameters
- Entropy Augmentation: Allows Vault to leverage the HSM for augmenting system entropy

[1.2.1] Master Key Wrapping and Automatic Unsealing



In some large organizations, there is a fair amount of complexity in designating key officers, who might be available to unseal Vault installations as the most common pattern is to deploy Vault immutably. As such automating unseal using an HSM provides a simplified yet secure way of unsealing Vault nodes as they get deployed.

Vault pulls its encrypted master key from storage and transits it through the HSM for decryption via PKCS #11 API. Once the master key is decrypted, Vault uses the master key to decrypt the encryption key to resume with Vault operations.

[1.2.2] Seal Wrapping

Vault encrypts secrets using 256-bit AES in GCM mode with a randomly generated nonce prior to writing them to its persistent storage. By enabling seal wrap, Vault wraps your secrets with an extra layer of encryption leveraging the HSM encryption and decryption.

Benefits of the Seal Wrap

- Conformance with FIPS 140-2 directives on Key Storage and Key Transport as certified by Leidos
- Supports FIPS level of security equal to HSM
    - For example, if you use Level 3 hardware encryption on an HSM, Vault will be using FIPS 140-2 Level 3 cryptography
- Allows Vault to be deployed in high security GRC environments (e.g. PCI-DSS, HIPAA) where FIPS guidelines important for external audits
- Pathway for Vault's use in managing Department of Defense's (DOD) or North Atlantic Treaty Organization (NATO) military secrets

[1.2.3] Entropy Augmentation

Entropy Augmentation allows Vault to leverage the HSM for augmenting system entropy.

With Entropy Augmentation enabled, the following keys and tokens leverage the configured external entropy source.

Operation	Description
Master Key	AES key that is encrypted by the seal mechanism. This encrypts the key ring.
Key Ring Encryption Keys	The keys embedded in Vault's keyring which encrypt all of Vault's storage.
Recovery Key	With auto-unseal, use the recovery keys to regenerate root token, key rotation, etc.
TLS Private Keys	For HA leader, Raft and Enterprise Replications.
MFA TOTP Keys	The keys used for TOTP in Vault Enterprise MFA
JWT Signing Keys	The keys used to sign wrapping token JWTs.
Root Tokens	Superuser tokens granting access to all operations in Vault.
DR Operation Tokens	Token that allows certain actions to be performed on a DR secondary.

The *transit* secrets engine manages a number of different key types and leverages the keysutil package to generate keys. It will use the external entropy source for key generation.

[1.3] COPYRIGHT AND TRADEMARK NOTICES

[1.4] TERMS OF USE

[1.5] GUARDIAN INTEGRATION

The Guardian Series 3 introduces mission-critical viability to core cryptographic infrastructure, including:

- Centralize device management
- Eliminates points of failure
- Distribute transaction loads
- Group-specific function blocking
- User-defined grouping systems

Please see applicable guide for configuring HSMs with the Guardian Series 3.

[2] PREREQUISITES

Supported Hardware:

- Vectera Plus, 6.7.x.x and above

Supported Operating Systems:

- Windows 7 and above
- Linux (Ubuntu, Debian and Red Hat-based distributions)

Other:

- OpenSSL
- Vault Enterprise HSM binary

[3] INSTALL FUTUREX PKCS #11 (FXPKCS11)

In a Windows environment, the easiest way to install the Futurex PKCS #11 (FXPKCS11) module is through installing **FXTools**. FXTools can be downloaded from the Futurex Portal. In a Linux environment, you need to download a tarball of the PKCS #11 binaries from the Futurex Portal. Then, extract the .tar file locally where you want the application to be installed in your file system. Step by step installation instructions for both of these scenarios is provided in the following subsections.

NOTE: The Futurex PKCS #11 module needs to be installed on the server that will be using the HSM.

[3.1] INSTRUCTIONS FOR INSTALLING THE FXPKCS11 MODULE USING FXTOOLS IN WINDOWS

- Run the FXTools installer as an administrator

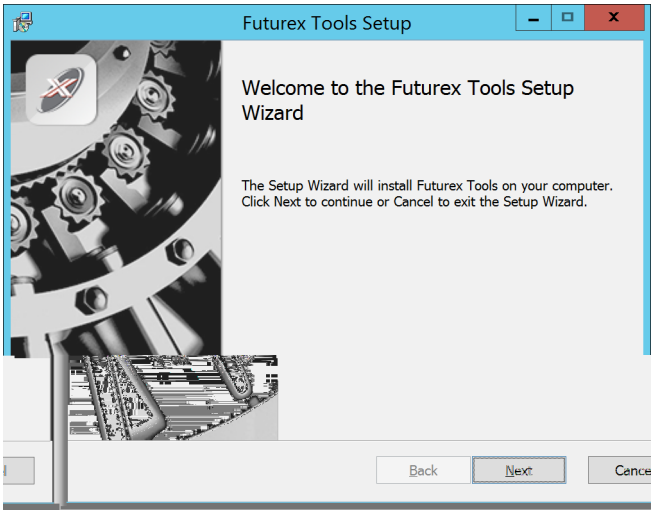


FIGURE: FUTUREX TOOLS SETUP WIZARD

By default, all tools are installed on the system. A user can overwrite and choose not to install certain modules.

- **Futurex Client Tools –**Command Line Interface (CLI) and associated SDK for both Java and C.
- **Futurex CNG Module –**The Microsoft Next Generation Cryptographic Library.
- **Futurex Cryptographic Service Provider (CSP) –**The legacy Microsoft cryptographic library.
- **Futurex EKM Module –**The Microsoft Enterprise Key Management library.
- **Futurex PKCS #11 Module –**The Futurex PKCS #11 library and associated tools.
- **Futurex Secure Access Client –**The client used to connect a Futurex Excrypt Touch to a local laptop, via USB, and a remote Futurex device.

After starting the installation, all noted services are installed. If the Futurex Secure Access Client was selected, the Futurex Excrypt Touch driver will also be installed (Note this sometimes will start minimized or in the background).

After installation is complete, all services are installed in the "*C:\Program Files\Futurex*" directory. The CNG Module, CSP Module, EKM Module, and PKCS #11 Module all require configuration files, located in their

corresponding directory with a *.cfg* extension. In addition, the CNG and CSP Modules are registered in the Windows Registry (*HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider*) and are installed in the "*C:\Windows\System32*" directory.

[3.2] INSTRUCTIONS FOR INSTALLING THE PKCS #11 MODULE IN LINUX

Extract the appropriate tarball file for your specific Linux distribution in the desired working directory.

NOTE: For the Futurex PKCS #11 module to be accessible system-wide, it would need to be placed into */usr/local/bin* by an administrative user. If the module only needs to be utilized by the current user, then installing into *$HOME/bin* would be the appropriate location.

The extracted content of the *.tar* file is a single *fxpkcs11* directory. Inside of the *fxpkcs11* directory are the following files and directories (Only files/folders that are relevant to the installation process are included below):

- *fxpkcs11.cfg* -> PKCS #11 configuration file
- *x86/* - This folder contains the module files for 32-bit architecture
- *x64/* - This folder contains the module files for 64-bit architecture

Within the *x86* and *x64* directories are two directories. One named *OpenSSL-1.0.x* and the other named *OpenSSL-1.1.x*. Both of these OpenSSL directories contain the PKCS #11 module files, built with the respective OpenSSL versions. These files are listed below, with short descriptions of each:

- *configTest* -> Program to test configuration and connection to the HSM
- *libfxpkcs11.so* -> PKCS #11 Library File
- *PKCS11Manager* -> Program to test connection and manage the HSM through the PKCS #11 library

The *configTest* and *PKCS11Manager* programs look for the *fxpkcs11.cfg* file at the following path:

```
/etc/fxpkcs11.cfg
```

Because of this, it is necessary either to move the *fxpkcs11.cfg* file from the */usr/local/bin/fxpkcs11* directory to the */etc* directory, or to set the FXPKCS11_CFG environment variable to point to the *fxpkcs11.cfg* file.

[4] INSTALL EXCRYPT MANAGER (IF USING WINDOWS)

The following two sections will cover how to install the **Excrypt Manager** and **FXCLI** applications. These tools are used to configure the HSM in subsequent sections. Note that installing Excrypt Manager is optional, but installing FXCLI is required, as FXCLI is the method that is used for configuring TLS mutual authentication between the Vectera Plus and the application that is being integrated.

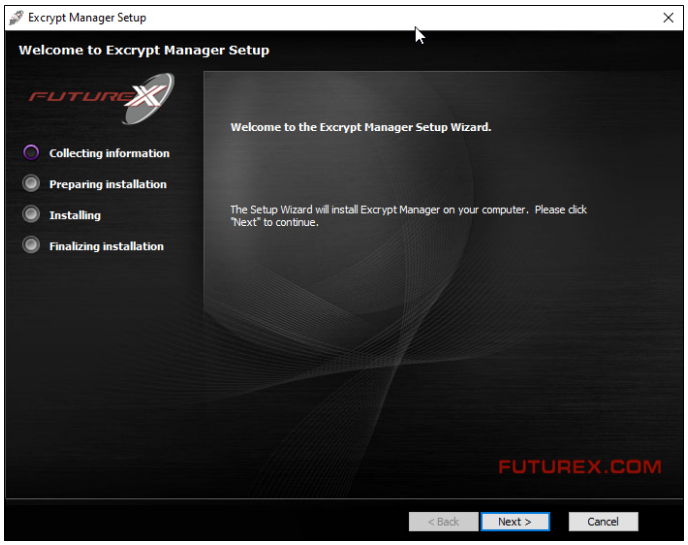
NOTE: Excrypt Manager needs to be installed on the workstation that is being used to configure the HSM.

Excrypt Manager is a Windows application that can be used to configure the HSM in subsequent sections. HSM configuration can also be completed using FXCLI, the Excrypt Touch, or the Guardian Series 3. For more information about using these tools/devices to configure the HSM, please see the relevant Administrator's Guide.

NOTE: If you plan to use a Virtual HSM for the integration, all configurations will need to be performed using either FXCLI, the Excrypt Touch, or the Guardian Series 3.

NOTE: The Excrypt Manager version must be from the 4.4.x branch or later to be compatible with the HSM firmware, which must be 6.7.x.x or later.

- Run the Excrypt Manager installer as an administrator.



The installation wizard will ask you to specify where you want Excrypt Manager to be installed. The default location is "*C:\Program Files\Futurex\Excrypt Manager*". Once that is done click "Install".

[5] INSTALL FUTUREX COMMAND LINE INTERFACE (FXCLI)

NOTE: FXCLI needs to be installed on the workstation that is being used to configure the HSM.

[5.1] INSTRUCTIONS FOR INSTALLING FXCLI IN WINDOWS

As mentioned in section 4, **Futurex Client Tools (FXCLI)** is included in the **FXTools** installation package. Just as with the **Futurex PKCS #11 (FXPKCS11)** module, the easiest way to install FXCLI on Windows is through installing FXTools. FXTools can be downloaded from the Futurex Portal.

- Run the FXTools installer as an administrator

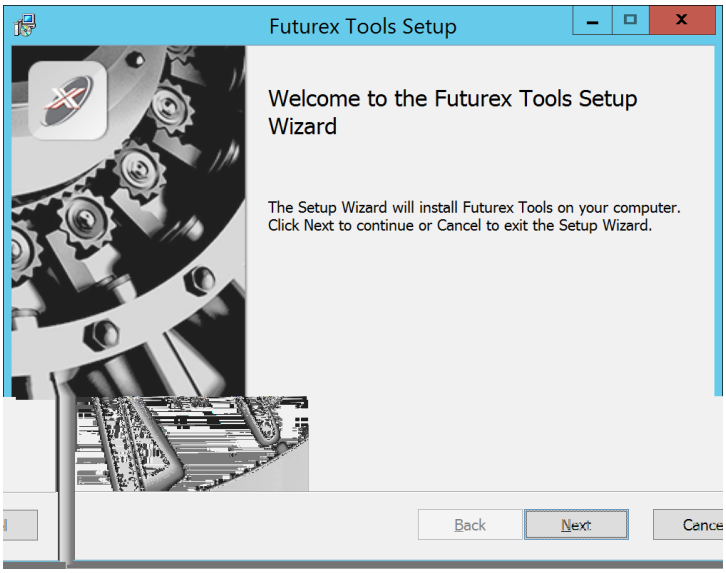


FIGURE: FUTUREX TOOLS SETUP WIZARD

By default, all tools are installed on the system. A user can overwrite and choose not to install certain modules.

NOTE: Since FXTools is only being used to install FXCLI in this case, it is not necessary to include any of the other services in the installation.

- **Futurex Client Tools –**Command Line Interface (CLI) and associated SDK for both Java and C.
- **Futurex CNG Module –**The Microsoft Next Generation Cryptographic Library.
- **Futurex Cryptographic Service Provider (CSP) –**The legacy Microsoft cryptographic library.
- **Futurex EKM Module –**The Microsoft Enterprise Key Management library.
- **Futurex PKCS #11 Module –**The Futurex PKCS #11 library and associated tools.
- **Futurex Secure Access Client –**The client used to connect a Futurex Excrypt Touch to a local laptop, via USB, and a remote Futurex device.

[5.2] INSTRUCTIONS FOR INSTALLING FXCLI IN LINUX

Download the FXCLI module

Users can download the appropriate FXCLI package files for their system from the Futurex Portal.

If the system is **64-bit**, users should select from the files marked **amd64**. If the system is **32-bit**, users should select from the files marked **i386**.

If running an OpenSSL version in the **1.0.x** branch, users should select from the files marked **ssl1.0**. If running an OpenSSL version in the **1.1.x** branch, users should select from the files marked **ssl1.1**.

Futurex offers the following features for FXCLI:

- Java Software Development Kit (**java**)
- HSM command line interface (**cli-hsm**)
- KMES command line interface (**cli-kmes**)
- Software Development Kit headers (**devel**)
- YAML parser used to parse bash output (**cli-fxparse**)

Install FXCLI

If installing an `.rpm` package, run the following command in a terminal:

```
$ sudo rpm -ivh [fxcl-xxxx.rpm]
```

If installing a `.deb` package, run the following command in a terminal:

```
$ sudo dpkg -i [fxcl-xxxx.deb]
```

After the installation is completed, system environment variables must be defined for the location of the FXCLI binaries. To do so permanently you must add the following two lines to your *.bashrc* file:

```
PATH=$PATH:/usr/bin/fxcli-hsm
PATH=$PATH:/usr/bin/fxcli-kmes
```

[6] CONFIGURE THE FUTUREX HSM

In order to establish a connection between the PKCS #11 library and the Futurex HSM, a few configuration items need to first be performed, which are the following:

NOTE: All of the steps in this section can be completed through either Excrypt Manager or FXCLI (if using a physical HSM rather than a virtual HSM). Optionally, steps 4 through 6 can be completed through the Guardian Series 3, which will be covered in Appendix A.

1. Connect to the HSM via the front USB port (NOTE: If you are using a virtual HSM for the integration you will have to connect to it over the network either via FXCLI, the Excrypt Touch, or the Guardian Series 3)
    a. Connecting via Excrypt Manager
    b. Connecting via FXCLI
2. Validate the correct features are enabled on the HSM
3. Setup the network configuration
4. Load the Futurex FTK
5. Configure a Transaction Processing connection and create a new Application Partition
6. Create a new Identity that has access to the Application Partition created in the previous step
7. Configure TLS Authentication. There are two options for this:
    a. Enabling server-side authentication
    b. Creating client certificates for mutual authentication

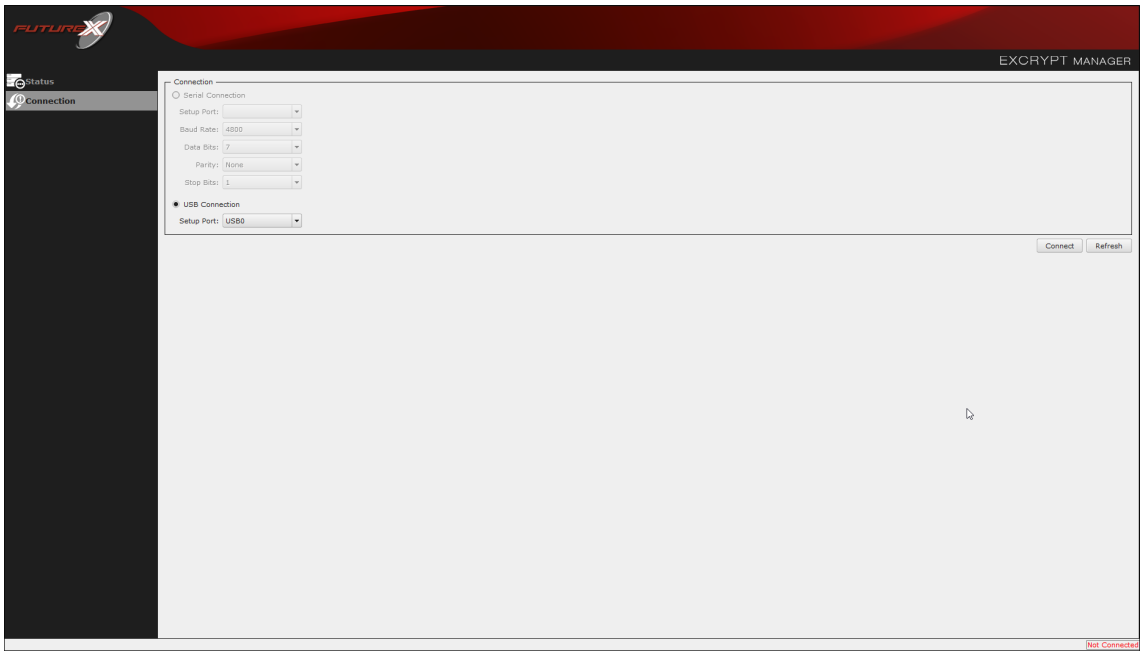
Each of these action items is detailed in the following subsections.

[6.1] CONNECT TO THE HSM VIA THE FRONT USB PORT

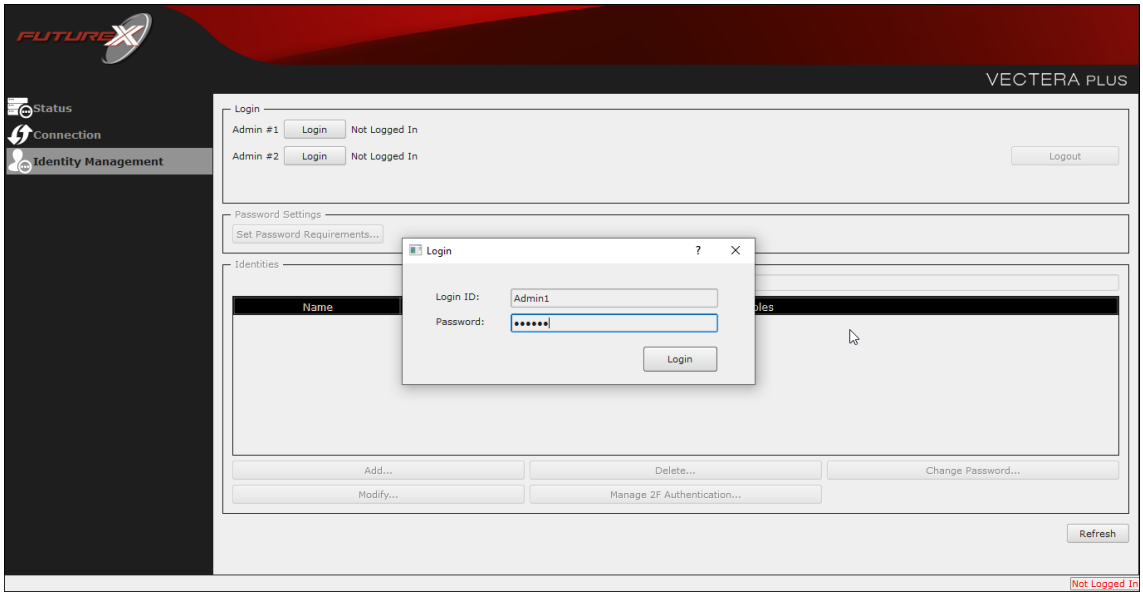
For both Excrypt Manager and FXCLI you need to connect your laptop to the front USB port on the HSM.

Connecting via Excrypt Manager

Open Excrypt Manager, click “Refresh” in the lower right-hand side of the Connection menu. Then select “USB Connection” and click “Connect”.

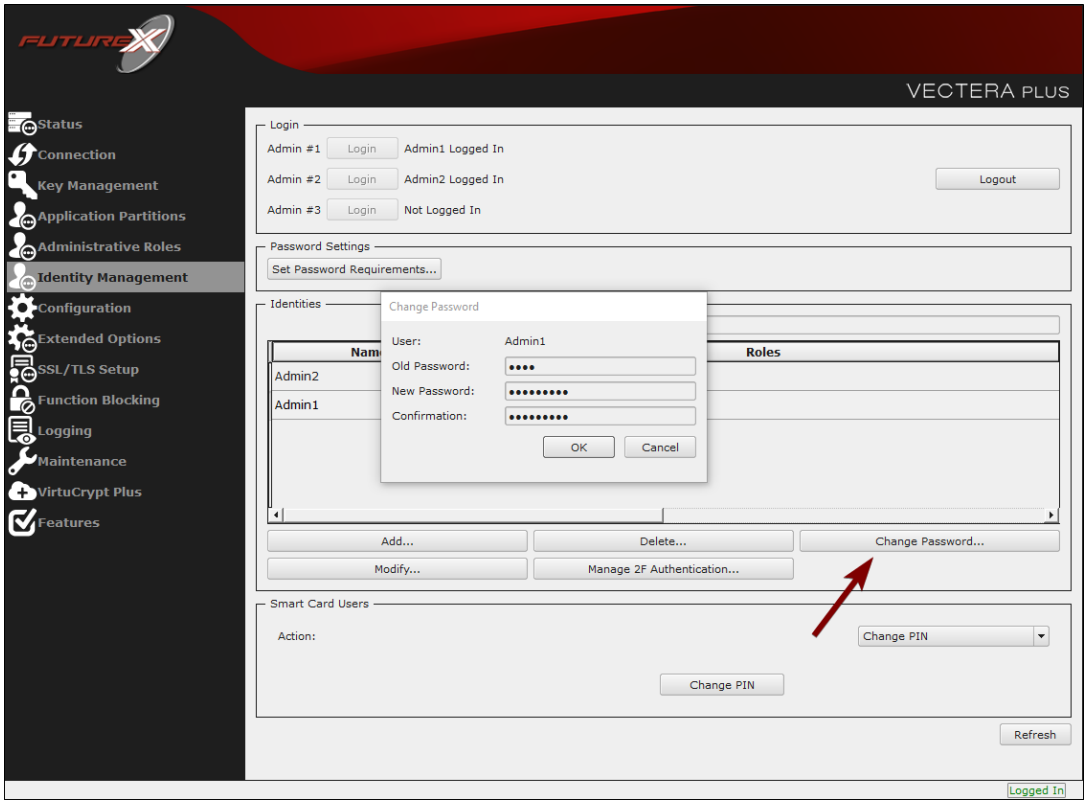


Login with both default Admin identities.



The default Admin passwords (i.e. “safe”) must be changed for both of your default Admin Identities (e.g. “Admin1” and “Admin2”) in order to load the major keys onto the HSM.

To do so via Excrypt Manager navigate to the Identity Management menu, select the first default Admin identity (e.g. “Admin1”), then click the “Change Password…” button. Enter the old password, then enter the new password twice, and click “OK”. Perform the same steps as above for the second default Admin identity (e.g. “Admin2”).



Connecting via FXCLI

Open the FXCLI application and run the following commands:

```
$ connect usb
$ login user
```

NOTE: The "login" command will prompt for the username and password. You will need to run it twice because you must login with both default Admin identities.

The default Admin passwords (i.e. “safe”) must be changed for both of your default Admin Identities (e.g. “Admin1” and “Admin2”) in order to load the major keys onto the HSM.

The following FXCLI commands can be used to change the passwords for each default Admin Identity.

```
$ user change-password -u Admin1
$ user change-password -u Admin2
```

NOTE: The user change-password commands above will prompt you to enter the old and new passwords. It is necessary to run the command twice (as shown above) because the default password must be changed for both default Admin identities.

[6.2] FEATURES REQUIRED IN HSM

In order to establish a connection between the PKCS #11 Library and the Futurex HSM, the HSM must be configured with the following features:

- **PKCS #11** -> Enabled
- **Command Primary Mode** -> General Purpose (GP)

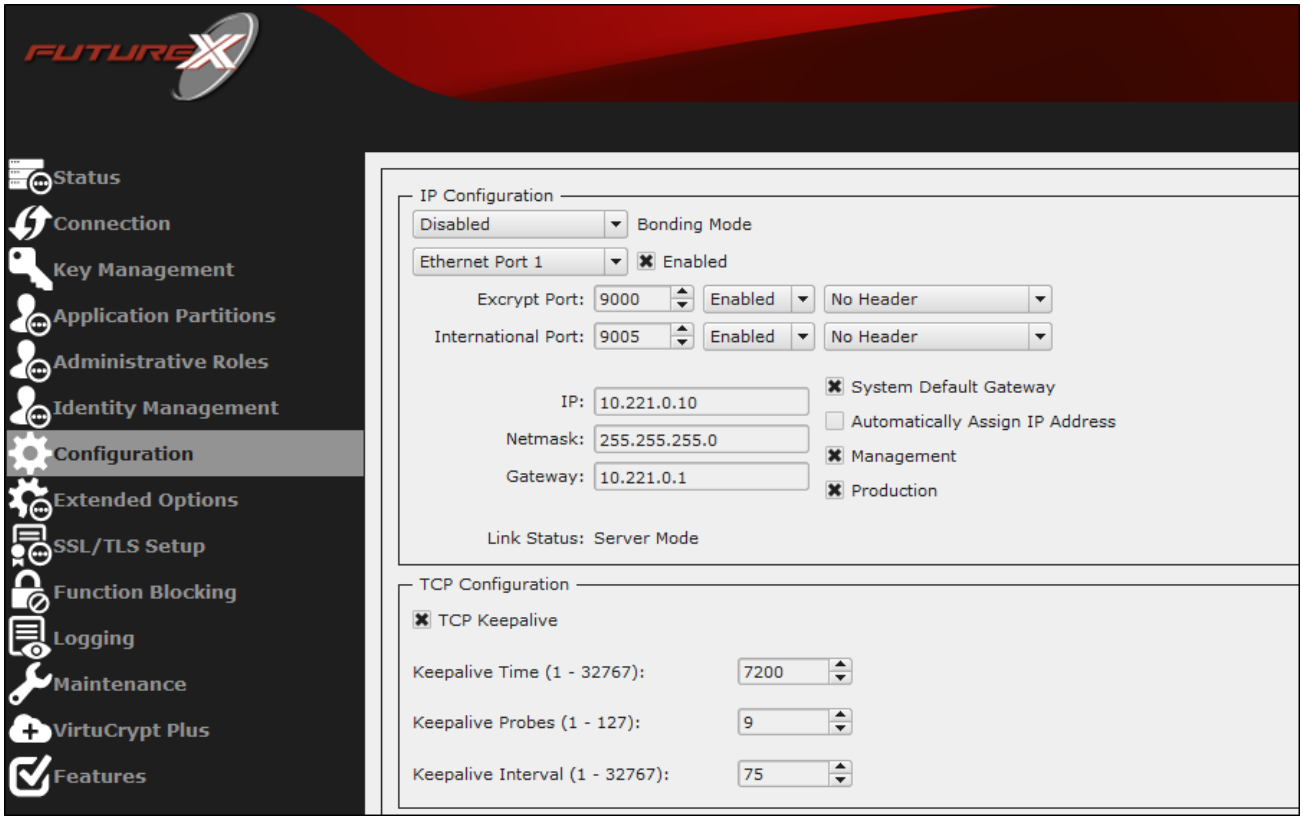
NOTE: For additional information about how to update features on your HSM, please refer to your HSM Administrator's Guide, section **"Download Feature Request File"**.

NOTE: Command Primary Mode = General Purpose, will enable the option to create the FTK major key in the HSM. This key will be required to be able to use the PKCS #11 library to communicate with the HSM. For detailed information about how to load major keys in HSMs please refer to your HSM Administrator's Guide.

[6.3] NETWORK CONFIGURATION (HOW TO SET THE IP OF THE HSM)

For this step you will need to be logged in with an identity that has a role with permissions ***Communication:Network Settings****. The default Administrator role and Admin identities can be used.*

Navigate to the *Configuration* page. There you will see the option to modify the IP configuration, as shown below:



Alternatively, the following **FXCLI** command can be used to set the IP for the HSM:

```
$ network interface modify --interface Ethernet1 --ip 10.221.0.10 --netmask 255.255.255.0 --gateway
10.221.0.1
```

NOTE: The following should be considered at this point:

- All of the remaining HSM configurations in this section can be completed using the Guardian Series 3 (please refer to Appendix A for instructions on how to do so), with the exception of the final subsection that covers how to create connection certificates for mutual authentication.
- If you are performing the configuration on the HSM directly now, but plan to add the HSM to a Guardian later, it may be necessary to synchronize the HSM after it is added to a Device Group on the Guardian.
- If configuration through a CLI is required for your use-case, then you should manage the HSMs directly.

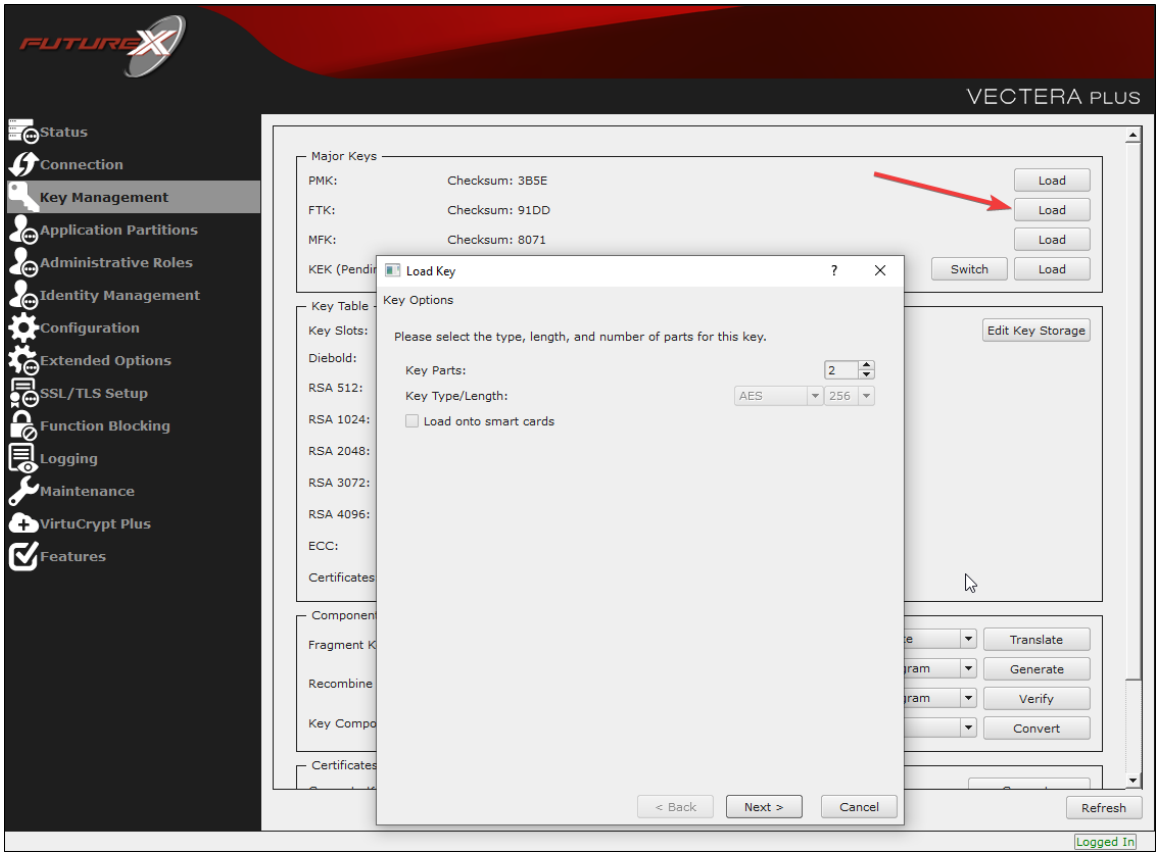
[6.4] LOAD FUTUREX KEY (FTK)

For this step you will need to be logged in with an identity that has a role with permissions ***Major Keys:Load****. The default Administrator role and Admin identities can be used.*

The FTK is used to wrap all keys stored on the HSM used with PKCS #11. If using multiple HSMs in a cluster, the same FTK can be used for syncing HSMs. Before an HSM can be used with PKCS #11, it must have an FTK.

NOTE: This process can also be completed using FXCLI, the Excrypt Touch, or the Guardian Series 3. For more information about how to load the FTK into an HSM using these tools/devices, please see the relevant Administrative Guide.

After logging in, select *Key Management*, then "Load" under FTK. Keys can be loaded as components that are XOR'd together, M-of-N fragments, or generated. If this is the first HSM in a cluster, it is recommended to generate the key and save to smart cards as M-of-N fragments.



Alternatively, the following **FXCLI** commands can be used to load an FTK onto an HSM.

If this is the first HSM you are setting up you will need to generate a random FTK. Optionally, you can also load it onto smart cards simultaneously with the -m and -n flags.

```
$ majorkey random --ftk -m [number_from_2_to_9] -n [number_from_2_to_9]
```

If it's a second HSM that you're setting up in a cluster then you will load the FTK from smart cards with the following command:

```
$ majorkey recombine --key ftk
```

[6.5] CONFIGURE A TRANSACTION PROCESSING CONNECTION AND CREATE AN APPLICATION PARTITION

For this step you will need to be logged in with an identity that has a role with permissions ***Role:Add, Role:Assign All Permissions, Role:Modify, Keys:All Slots****, and* ***Command Settings:Excrypt****. The default Administrator role and Admin identities can be used.*

NOTE: For the purposes of this integration guide you can consider the terms "Application Partition" and "Role" to be synonymous. For more information regarding Application Partitions, Roles, and Identities, please refer to the relevant Administrator's guide.

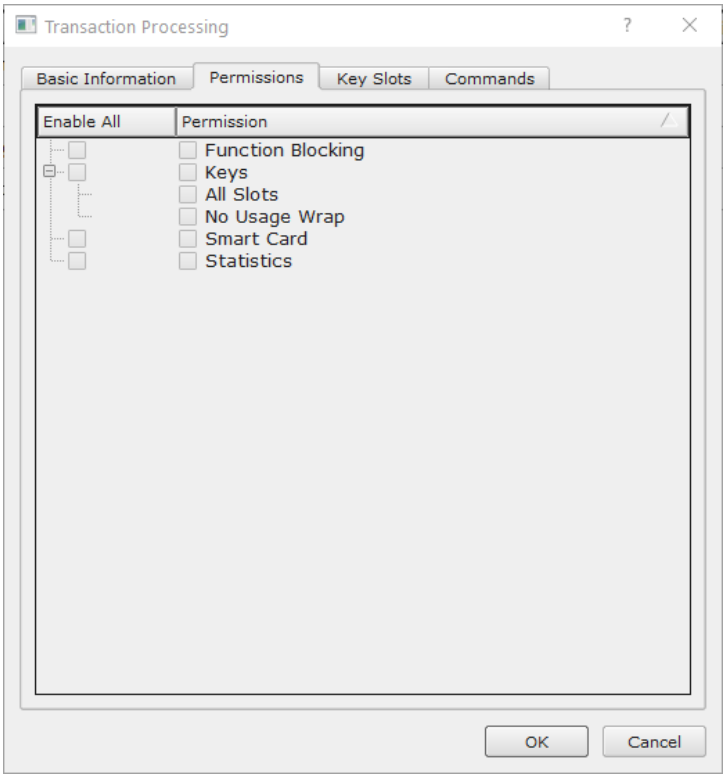
Configure a Transaction Processing Connection

Before an application logs in to the HSM with an authenticated user, it first connects via a “Transaction Processing” connection to the **Transaction Processing** Application Partition. For this reason, it is necessary to take steps to harden this Application Partition. The following three things need to be configured for the Transaction Processing partition:

1. It should not have access to the “All Slots” permissions
2. It should not have access to any key slots
3. Only the PKCS #11 communication commands should be enabled

Go to *Application Partitions*, select the Transaction Processing Application Partition, and click Modify.

Navigate to the "Permissions" tab and ensure that the “All Slots” key permission is unchecked. None of the other key permissions should be enabled either.



Under the "Key Slots" tab you need to ensure that there are no key ranges specified. By default, the Transaction Processing Application Partition has access to the entire range of key slots on the HSM.

Lastly, under the “Commands” tab make sure that only the following **PKCS #11 Communication commands** are enabled:

- **ECHO**: Communication Test/Retrieve Version
- **PRMD**: Retrieve HSM restrictions
- **RAND**: Generate random data
- **HASH**: Retrieve device serial
- **GPKM**: Retrieve key table information
- **GPKS**: General purpose key settings get/change
- **GPKR**: General purpose key settings get (read-only)

Alternatively, the following **FXCLI** commands can be used to remove all permissions and key ranges that are currently assigned to the **Transaction Processing** role and enable only the PKCS #11 Communication commands:

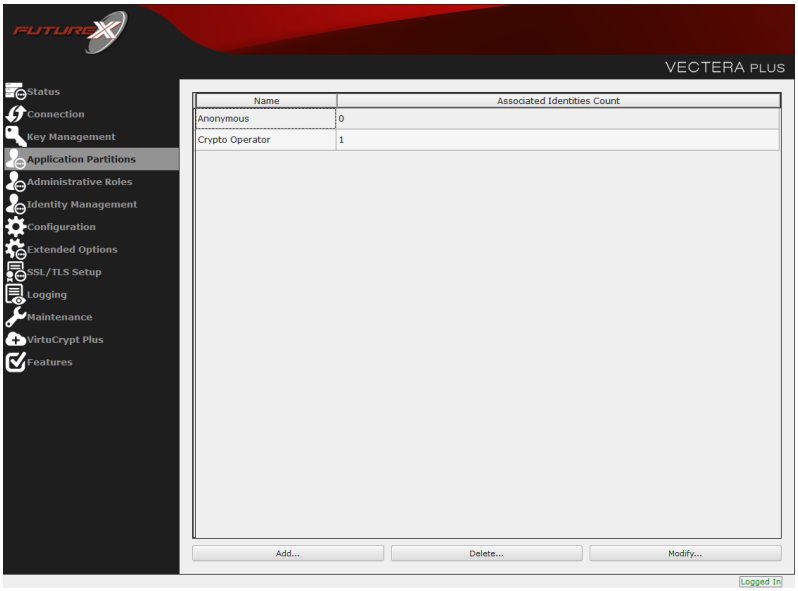
```
$ role modify --name Anonymous --clear-perms --clear-key-ranges
```

```
$ role modify --name Anonymous --add-perm Excrypt:ECHO --add-perm Excrypt:PRMD --add-perm Excrypt:RAND
--add-perm Excrypt:HASH --add-perm Excrypt:GPKM --add-perm Excrypt:GPKS --add-perm Excrypt:GPKR
```

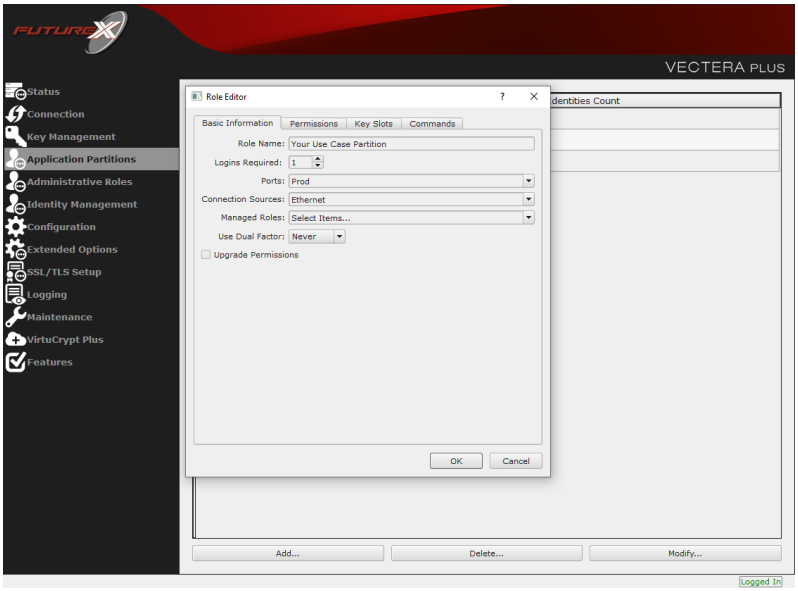
Create an Application Partition

In order for application segregation to occur on the HSM, an Application Partition must be created specifically for your use case. Application partitions are used to segment the permissions and keys on an HSM between applications. The process for configuring a new application partition is outlined in the following steps:

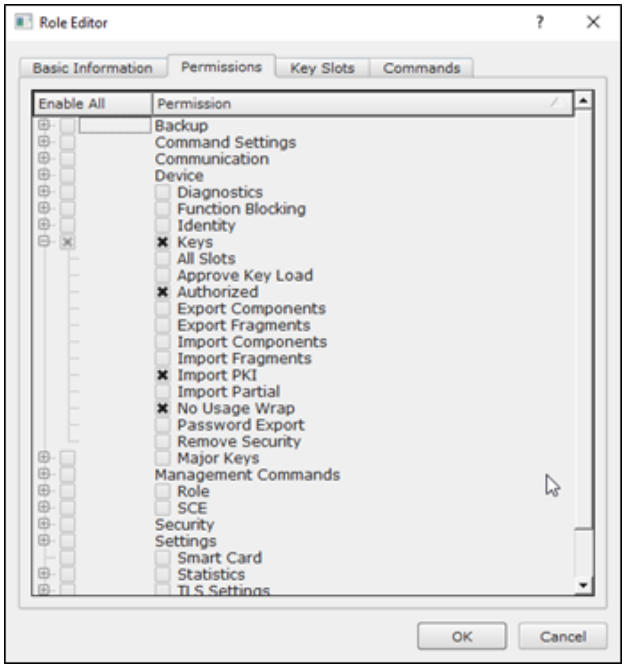
Navigate to the *Application Partitions* page and click the "Add" button at the bottom.



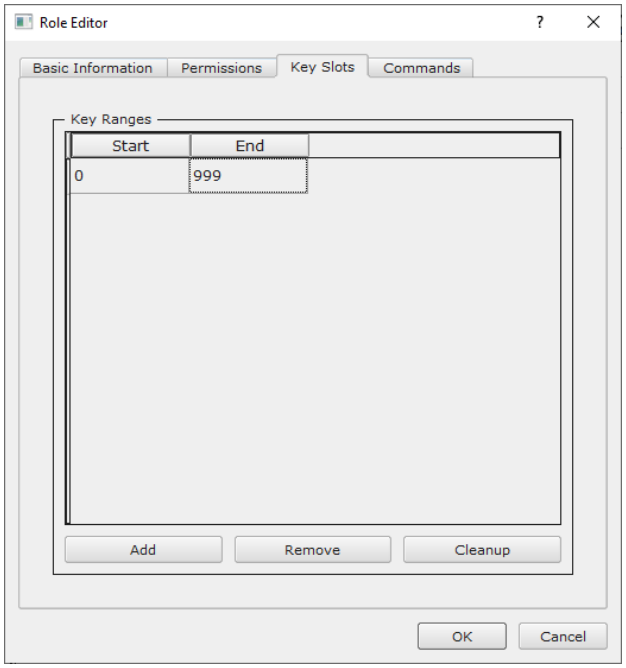
Fill in all of the fields in the *Basic Information* tab exactly how you see below (except for the *Role Name* field). In the *Role Name* field, specify any name that you would like for this new Application Partition. *Logins Required* should be set to “1”. *Ports* should be set to “Prod”. *Connection Sources* should be configured to “Ethernet”. The *Managed Roles* field should be left blank because we’ll be specifying the exact Permissions, Key Slots, and Commands that we want this Application Partition/Role to have access to. Lastly, the *Use Dual Factor* field should be set to “Never”.



Under the “Permissions” tab, select the key permissions shown in the screenshot below. The **Authorized** permission allows for keys that require login. The **Import PKI** permission allows trusting an external PKI, which is used by some applications to allow for PKI symmetric key wrapping (It is not recommended to enable unless using this use case). The **No Usage Wrap** permission allows for interoperable key wrapping without defining key usage as part of the wrapped key (This is only recommended if exchanging keys with external entities or using the HSM to wrap externally used keys).



Under key slots, it is recommended that you create a range of 1000 total keys (here we’ve specified the key range 0-999), which do not overlap with another Application Partition. Within this range, there must be ranges for both symmetric and asymmetric keys. If more keys are required by the application, configure accordingly.



Based on application requirements there are particular functions that need to be enabled on the Application Partition in order to utilize the HSMs functionality. The most often used commands are included below. These can be enabled under the "Commands" tab.

PKCS #11 Communication Commands

- **ECHO**: Communication Test/Retrieve Version
- **PRMD**: Retrieve HSM restrictions
- **RAND**: Generate random data
- **HASH**: Retrieve device serial
- **GPKM**: Retrieve key table information
- **GPKS**: General purpose key settings get/change
- **GPKR**: General purpose key settings get (read-only)

Key Operations Commands

- **APFP**: Generate PKI Public Key from Private Key
- **ASYL**: Load asymmetric key into key table
- **GECC**: Generate an ECC Key Pair
- **GPCA**: General purpose add certificate to key table
- **GPGS**: General purpose generate symmetric key
- **GPKA**: General purpose key add
- **GPKD**: General purpose key slot delete/clear
- **GRSA**: Generate RSA Private and Public Key
- **LRSA**: Load key into RSA Key Table
- **RPFP**: Get public components from RSA private key

Interoperable Key Wrapping

- **GPKU**: General purpose key unwrap (unrestricted)
- **GPUK**: General purpose key unwrap (preserves key usage)
- **GPKW**: General purpose key wrap (unrestricted)
- **GPWK**: General purpose key wrap (preserves key usage)

Data Encryption Commands

- **ADPK**: PKI Decrypt Trusted Public Key
- **GHSH**: Generate a Hash (Message Digest)
  *Starting in firmware version 7.x, this function is enabled by default and does not need to be specified.
- **GPED**: General purpose data encrypt and decrypt
- **GPGC**: General purpose generate cryptogram from key slot
- **GPMC**: General purpose MAC (Message Authentication Code)
- **GPSR**: General purpose RSA encrypt/decrypt or sign/verify with recovery
- **HMAC**: Generate a hash-based message authentication code
- **RDPK**: Get Clear Public Key from Cryptogram

Signing Commands

- **ASYS**: Generate a Signature Using a Private Key
- **ASYV**: Verify a Signature Using a Public Key
- **GPSV**: General purpose data sign and verify
- **RSAS**: Generate a Signature Using a Private Key

Alternatively, the following **FXCLI** commands can be used to create the new Application Partition and enable all of the functions that are needed:

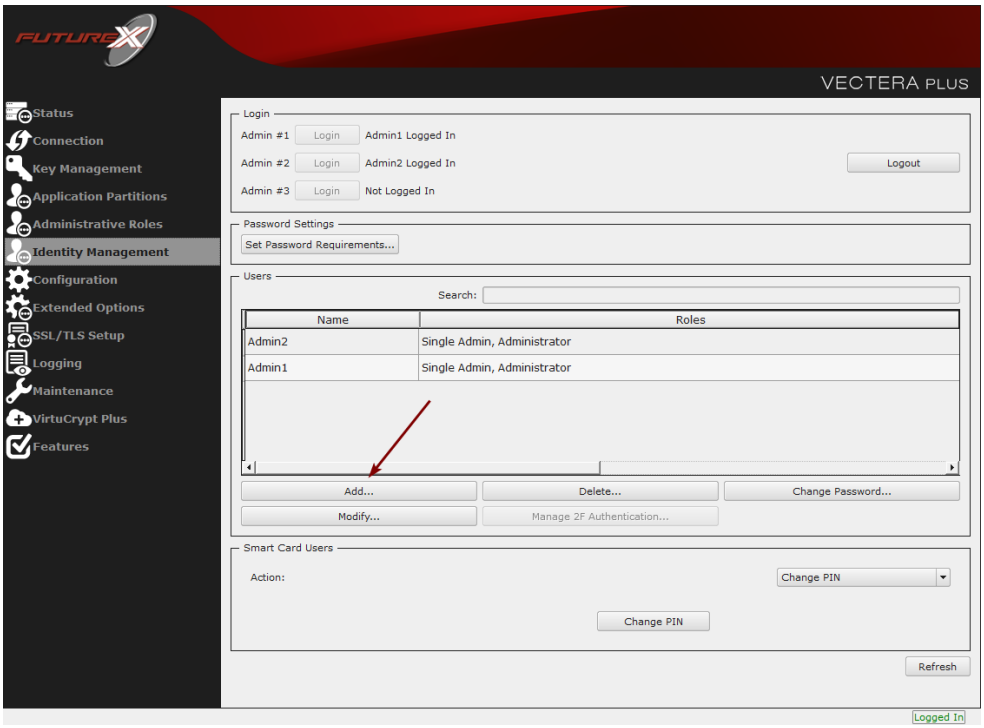
```
$ role add --name Role_Name --application --key-range (0,999) --perm "Keys:Authorized" --perm "Key-
s:Import PKI" --perm "Keys:No Usage Wrap"
```

```
$ role modify --name [role_name] --clear-perms --add-perm Excrypt:ECHO --add-perm Excrypt:PRMD --add-
perm Excrypt:RAND --add-perm Excrypt:HASH --add-perm Excrypt:GPKM --add-perm Excrypt:GPKS --add-perm
Excrypt:GPKR --add-perm Excrypt:APFP --add-perm Excrypt:ASYL --add-perm Excrypt:GECC --add-perm
Excrypt:GPCA --add-perm Excrypt:GPGS --add-perm Excrypt:GPKA --add-perm Excrypt:GPKD --add-perm
Excrypt:GRSA --add-perm Excrypt:LRSA --add-perm Excrypt:RPFP --add-perm Excrypt:GPKU --add-perm
Excrypt:GPUK --add-perm Excrypt:GPKW --add-perm Excrypt:GPWK --add-perm Excrypt:ADPK --add-perm
Excrypt:GHSH --add-perm Excrypt:GPED --add-perm Excrypt:GPGC --add-perm Excrypt:GPMC --add-perm
Excrypt:GPSR --add-perm Excrypt:HMAC --add-perm Excrypt:RDPK --add-perm Excrypt:ASYS --add-perm
Excrypt:ASYV --add-perm Excrypt:GPSV --add-perm Excrypt:RSAS
```

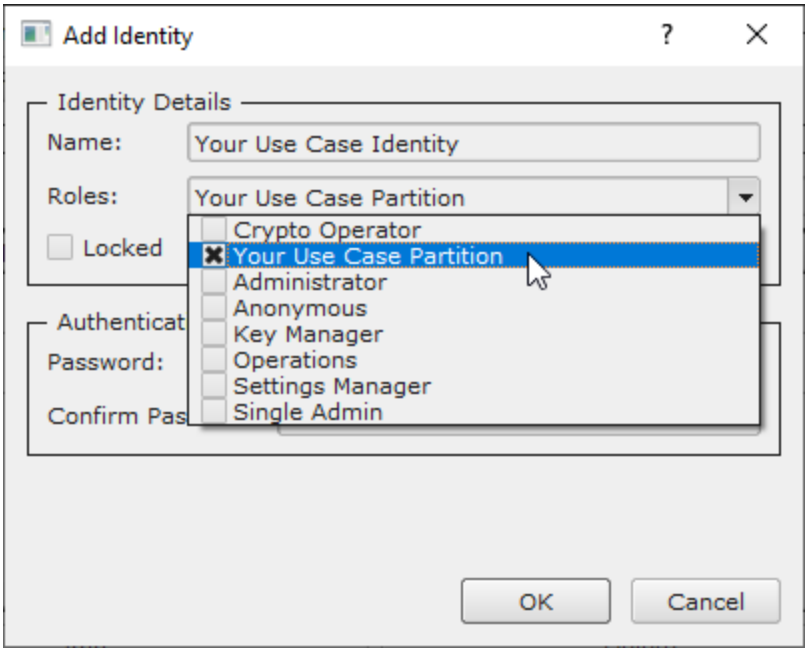
[6.6] CREATE NEW IDENTITY AND ASSOCIATE IT WITH THE NEWLY CREATED APPLICATION PARTITION

*For this step you will need to be logged in with an identity that has a role with permissions **Identity:Add**. The default Administrator role and Admin identities can be used.*

A new identity must be created, which will need to be associated with the Application Partition created in the previous step. To create this new identity, go to *Identity Management*, and click "Add".



Specify a name for the new identity, and in the Roles dropdown select the name of the Application Partition created in the previous step. This will associate the new Identity with the Application Partition that you created.



Alternatively, the following **FXCLI** command can be used to create a new Identity and associate it with the role that was created:

```
$ identity add --name Identity_Name --role Role_Name --password safest
```

This new identity must be set in fxpkcs11.cfg file, in the following section:

```
#HSM crypto operator identity name
<CRYPTO-OPR>     [insert name of identity that you created]     </CRYPTO-OPR>

# Production connection
<PROD-ENABLED>    YES        </PROD-ENABLED>
<PROD-PORT>        9100         </PROD-PORT>
```

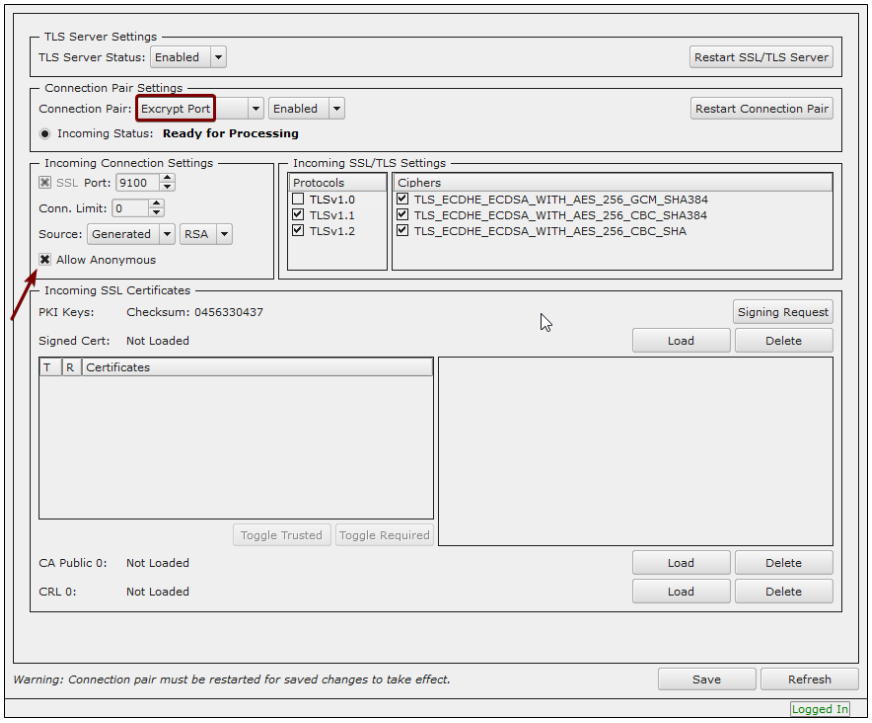
NOTE: Crypto Operator in the fxpkcs11.cfg file must match exactly the name of the identity created in the HSM.

[6.7] CONFIGURE TLS AUTHENTICATION

For this step you will need to be logged in with an identity that has a role with permissions ***Keys:All Slots, Management Commands:Certificates, Management Commands:Keys, Security:TLS Sign****, and* ***TLS Settings:Upload Key****. The default Administrator role and Admin identities can be used.*

Enable Server-Side Authentication (Option 1)

Mutually authenticating to the HSM using client certificates is recommended, but server-side authentication is also supported. To enable server-side authentication go to *SSL/TLS Setup*, then select the Excrypt Port and enable the "Allow Anonymous" setting.



Alternatively, the following **FXCLI** command can be used to enable server-side authentication with the "Allow Anonymous" SSL/TLS setting:

```
$ tls-ports set -p "Excrypt Port" --anon
```

Create Connection Certificates for Mutual Authentication (Option 2)

Mutually authenticating to the HSM using client certificates is recommended, and enforced by default. In the example below, FXCLI is utilized to generate a CA that then signs the HSM server certificate and a client certificate. The client keys and CSR are generated in Windows PowerShell with OpenSSL. For other options for managing certificates required for mutual authentication with the HSM, please review the relevant Administrator's guide.

Find the **FXCLI** program that was installed with FXTools, and run it as an administrator.

Things to note:

- For this example, the computer running FXCLI is connected to the front port of the HSM. Remote management is possible however, using the HSMs Web Portal, or the Excrypt Touch.
- For commands that create an output file, if you do not specify a file path (as is the case here) it will save the file to the directory from which the FXCLI program is executed.
- Using user-generated certificates requires a PMK to be loaded on the HSM.
- If you run **help** by itself it will show a full list of available commands. You can see all of the available options for any given command by running the command name followed by **help**.

```
# Connect your laptop to the HSM via the USB port on the front, then run this command.
$ connect usb
```

```
# Log in with both default Admin identities. This command will prompt for the username and password.
You will need to run this command twice.
$ login user
```

```
# Generate TLS CA and store it in an available key slot on the HSM
$ generate --algo RSA --bits 2048 --usage mak --name TlsCaKeyPair --slot next
```

```
# Create root certificate
$ x509 sign \
    --private-slot TlsCaKeyPair \
    --key-usage DigitalSignature --key-usage KeyCertSign \
    --ca true --pathlen 0 \
    --dn 'O=Futurex\CN=Root' \
    --out TlsCa.pem
```

```
# Generate the server keys for the HSM
$ tls-ports request --pair "Excrypt Port" --file production.csr --pki-algo RSA
```

```
# Sign the server CSR with the newly created TLS CA
$ x509 sign \
    --private-slot TlsCaKeyPair \
    --issuer TlsCa.pem \
    --csr production.csr \
    --eku Server --key-usage DigitalSignature --key-usage KeyAgreement \
    --ca false \
    --dn 'O=Futurex\CN=Production' \
    --out TlsProduction.pem
```

```
# Push the signed server PKI to the production port on the HSM
$ tls-ports set --pair "Excrypt Port" \
    --enable \
    --pki-source Generated \
    --clear-pki \
    --ca TlsCa.pem \
    --cert TlsProduction.pem \
    --no-anon
```

***NOTE**: The following OpenSSL commands will need to be run from Windows PowerShell, rather than from the FXCLI program.*

```
# Generate the client keys
$ openssl genrsa -out privatekey.pem 2048
```

```
# Generate client CSR
$ openssl req -new -key privatekey.pem -out ClientPki.csr -days 365
```

Using FXCLI, sign the CSR that was just generated using OpenSSL.

```
# Sign the client CSR under the root certificate that was created
$ x509 sign  \
 --private-slot TlsCaKeyPair \
 --issuer TlsCa.pem \
 --csr ClientPki.csr \
 --eku Client --key-usage DigitalSignature --key-usage KeyAgreement \
 --dn 'O=Futurex\CN=Client' \
 --out SignedPki.pem
```

Switch back to Windows PowerShell for the remaining commands.

```
## Make PKCS12 file
# Concatenate the signed client cert and private key into one pem file
$ cat SignedPki.pem >> Tree.pem
```

```
$ cat privatekey.pem >> Tree.pem
```

```
# Use OpenSSL to create a PKCS#12 file that can be used to authenticate, as a client, using our PKCS
#11 library
$ openssl pkcs12 -export -in Tree.pem -out PKI.p12 -name "ClientPki" -password pass:safest
```

[7] EDIT THE CONFIGURATION FILE

[7.1] DEFINE CONNECTION INFORMATION

The *fxpkcs11.cfg* file allows the user to set the PKCS #11 library to connect to the HSM. To edit, run a text editor as an Administrator and edit the configuration file accordingly. Most notably, the fields shown below must be set inside the **<HSM>** section (note that the full *fxpkcs11.cfg* file is not included).

NOTE: Our PKCS #11 library expects the PKCS #11 config file to be in a certain location (*C:\Program Files\Futurex\fxpkcs11\fxpkcs11.cfg* for Windows and */etc/fxpkcs11.cfg* for Linux), but that location can be overwritten using an environment variable (FXPKCS11_CFG).

```
# Connection information
<ADDRESS>             10.0.5.58           </ADDRESS>

# Load balancing
<FX-LOAD-BALANCE>           YES               </FX-LOAD-BALANCE>

# Log configuration
<LOG-FILE> C:\Program Files\Futurex\fxpkcs11\fxpkcs11.log </LOG-FILE>

# HSM crypto operator identity name
<CRYPTO-OPR>          [identity_name]     </CRYPTO-OPR>

# Production connection
<PROD-ENABLED>        YES               </PROD-ENABLED>
<PROD-PORT>           9100              </PROD-PORT>

# Production SSL information
<PROD-TLS-ANONYMOUS>    NO              </PROD-TLS-ANONYMOUS>
<PROD-TLS-CA>     C:\Program Files\Futurex\fxpkcs11\TlsCa.pem    </PROD-TLS-CA>
<PROD-TLS-CA>     C:\Program Files\Futurex\fxpkcs11\TlsProduction.pem    </PROD-TLS-CA>
<PROD-TLS-KEY>     C:\Program Files\Futurex\fxpkcs11\PKI.p12     </PROD-TLS-KEY>
<PROD-TLS-KEY-PASS>      safest          </PROD-TLS-KEY-PASS>
```

In the **<ADDRESS>** field, the IP of the HSM that the PKCS #11 library will connect to is specified.

If a Guardian is being used to manage HSMs in a cluster, the **<FX-LOAD-BALANCE>** field must be defined as "YES". If a Guardian is not being used it should be set to "NO".

In the **<LOG-FILE>** field, set the path to the PKCS #11 log file.

In the **<CRYPTO-OPR>** field, the name of identity created in step 7.6 needs to be specified.

The **<PROD-ENABLED>** and **<PROD-PORT>** fields declare that the PKCS #11 library will connect to Production port 9100.

The **<PROD-TLS-ANONYMOUS>** field defines whether the PKCS #11 library will be authenticating to the server or not.

The **<PROD-TLS-KEY>** field defines the location of the client private key. Supported formats for the TLS private key are PKCS #1 clear private keys, PKCS #8 encrypted private keys, or a PKCS #12 file that contains the private key and certificates encrypted under the password specified in the **<PROD-TLS-KEY-PASS>** field.

Because a PKCS #12 file is defined in the **<PROD-TLS-KEY>** field in this example, it is not necessary to define the signed client cert with the **<PROD-TLS-CERT>** tag, or the CA cert/s with one or more instances of the **<PROD-TLS-CA>** tag.

For additional details reference the Futurex PKCS #11 technical reference found on the Futurex Portal.

Once the fxpkcs11.cfg is edited, run the "**PKCS11Manager**" file to test the connection against the HSM, and check the fxpkcs11.log for errors and information. For more information, see our Administrator's Guide.

[7.2] SPECIAL COMPATIBILITY MODE CONFIGURATION REQUIRED FOR THIS INTEGRATION

This integration requires two special defines in the **<CONFIG>** section of the *fxpkcs11.cfg* file.

```
<FORCED-LABEL-USAGE> hsm_demo = ENCRYPT | DECRYPT </FORCED-LABEL-USAGE>
<FORCED-LABEL-USAGE> hsm_hmac_demo = SIGN | VERIFY </FORCED-LABEL-USAGE>
```

These defines force specific usages for the two keys that Vault creates on the HSM, based on the key labels that are specified.

NOTE: The "hsm_demo" and "hsm_hmac_demo" key labels correspond with what is defined for the "key_label" and "hmac_key_label" values in the *vault.hcl* file (covered in section 9.4).

[8] STEPS TO CONFIGURE THE FUTUREX PKCS #11 LIBRARY WITH HASHICORP VAULT

NOTE: Vault's Hardware Security Module (HSM) auto-unseal and Seal Wrap features require Vault Enterprise with the Governance & Policy Module.

[8.1] DOWNLOAD VAULT

Precompiled Vault binaries are available for download at https://releases.hashicorp.com/vault/ and Vault Enterprise binaries are available for download by following the instructions made available to HashiCorp Vault customers.

This integration requires the Enterprise HSM binary. It is available at this link to use for testing: https://releases.hashicorp.com/vault/1.5.0+ent.hsm/

[8.2] INSTALL VAULT

Unzip the downloaded package and move the *vault* binary to */usr/local/bin/*.

```
$ unzip vault_${VAULT_VERSION}+ent.hsm_linux_amd64.zip
```

Set the owner of the Vault binary.

```
$ sudo chown root:root vault
```

Check that vault is available on the system path.

```
$ sudo mv vault /usr/local/bin/
```

Verify the Vault version.

```
$ vault --version
```

The **vault** command features opt-in autocompletion for flags, subcommands, and arguments (where supported).

```
$ vault -autocomplete-install
```

Enable autocompletion.

```
$ complete -C /usr/local/bin/vault vault
```

Give Vault the ability to use the mlock syscall without running the process as root. The mlock syscall prevents memory from being swapped to disk.

```
$ sudo setcap cap_ipc_lock=+ep /usr/local/bin/vault
```

Create a unique, non-privileged system user to run Vault.

```
$ sudo useradd --system --home /etc/vault.d --shell /bin/false vault
```

[8.3] CONFIGURE SYSTEMD

Systemd uses documented sane defaults so only non-default values must be set in the configuration file.

Create a Vault service file at /etc/systemd/system/vault.service.

```
$ sudo touch /etc/systemd/system/vault.service
```

Add the below configuration to the Vault service file:

```
[Unit]
Description="HashiCorp Vault - A tool for managing secrets"
Documentation=https://www.vaultproject.io/docs/
Requires=network-online.target
After=network-online.target
ConditionFileNotEmpty=/etc/vault.d/vault.hcl
StartLimitIntervalSec=60
StartLimitBurst=3

[Service]
User=vault
Group=vault
ProtectSystem=full
ProtectHome=read-only
PrivateTmp=yes
PrivateDevices=yes
SecureBits=keep-caps
AmbientCapabilities=CAP_IPC_LOCK
Capabilities=CAP_IPC_LOCK+ep
CapabilityBoundingSet=CAP_SYSLOG CAP_IPC_LOCK
NoNewPrivileges=yes
ExecStart=/usr/local/bin/vault server -config=/etc/vault.d/vault.hcl
ExecReload=/bin/kill --signal HUP $MAINPID
KillMode=process
KillSignal=SIGINT
Restart=on-failure
RestartSec=5
TimeoutStopSec=30
StartLimitInterval=60
StartLimitIntervalSec=60
StartLimitBurst=3
LimitNOFILE=65536
LimitMEMLOCK=infinity

[Install]
WantedBy=multi-user.target
```

[8.4] CONFIGURE VAULT

Vault uses documented sane defaults so only non-default values must be set in the configuration file.

Create */etc/vault.d* directory.

```
$ sudo mkdir --parents /etc/vault.d
```

Create a Vault configuration file, *vault.hcl*.

```
$ sudo touch /etc/vault.d/vault.hcl
```

Set the ownership of the */etc/vault.d* directory.

```
$ sudo chown --recursive vault:vault /etc/vault.d
```

Set the file permissions.

```
$ sudo chmod 640 /etc/vault.d/vault.hcl
```

Configure HSM Auto-unseal and Entropy Augmentation

When a Vault server is started, it normally starts in a sealed state where a quorum of existing unseal keys is required to unseal it. By integrating Vault with an HSM, the Vault server can be automatically unsealed by the trusted HSM key provider.

To integrate the Vault Enterprise server with an HSM cluster, the configuration file must define the PKCS11 seal stanza providing necessary connection information.

Example: *vault.hcl*

```
# Provide your Futurex HSM connection information
seal "pkcs11" {
  lib = "/usr/local/bin/fxpkcs11/x64/OpenSSL-1.1.x/libfxpkcs11.so"
  slot = "0"
  key_label = "hsm_demo"
  hmac_key_label = "hsm_hmac_demo"
  generate_key = "true"
}

# Add the entropy stanza
entropy "seal" {
  mode = "augmentation"
}

# Configure the storage backend for Vault
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"
}

ui = true
disable_mlock = true
```

NOTE: For the purpose of this guide, the storage backend is set to the local file system (*/tmp/vault*) to make the verification step easy.

The example configuration defines the following in its seal stanza:

Parameter	Description
lib	Path to the PKCS #11 library on the machine where Vault Enterprise is installed.
slot	The slot number to use (this should be set to "0" because "0" is the slot that is set by default in the FXPKCS11 config file).
key_label	Defines the label of the key to use.
hmac_key_label	Defines the label of the key to use for HMACing.
generate_key	If no existing key with the label specified by key_label can be found at Vault initialization time, Vault generates a key.

NOTE: For this integration, the **generate_key** parameter needs to be set to "true" so that Vault will automatically create the encryption keys that it uses for the Seal Wrap functionality on the HSM. The values set for the **key_label** and **hmac_key_label** parameters correspond with the special key label defines that must be set in the **<CONFIG>** section of the *fxpkcs11.cfg* file (covered in section 8.2).

For the full list of configuration parameters, please refer to the Vault documentation here.

[8.5] START THE VAULT SERVER

First, log in with the **vault** user.

Next, set the PKCS #11 PIN for login with the following command (this is the password of the Identity created on the HSM and defined in the FXPKCS11 config file).

```
$ export VAULT_HSM_PIN='safest'
```

NOTE: The PKCS #11 PIN can also be set in the Vault configuration file (i.e., *vault.hcl*) with the **pin** parameter, but this is not recommended in a production setting. Best practice is to specify the pin with the VAULT_HSM_PIN environment variable, as shown here. This prevents the password from being exposed if the config file is compromised or stored in an unsecure location. If set via the environment variable, Vault will obfuscate the environment variable after reading it. The one caveat is that the VAULT_HSM_PIN environment variable will need to be re-set if Vault is restarted.

Now, start the Vault server with the following command.

```
$ vault server -config=/etc/vault.d/vault.hcl
```

If the above command is successful, something similar to the following output is expected:

```
==> Vault server configuration:
     HSM PKCS#11 Version: 2.20
             HSM Library: FxPKCS11
     HSM Library Version: 4.23
     HSM Manufacturer ID: Futurex
                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_dur-
ation: "1m30s", max_request_size: "33554432", tls: "disabled")
               Log Level: info
                   Mlock: supported: true, enabled: false
           Recovery Mode: false
                 Storage: file
                 Version: Vault v1.5.0+ent.hsm
==> Vault server started! Log data will stream in below:
```

Open a new terminal window and leave the terminal running where the Vault server was started.

[8.6] INITIALIZE VAULT

In the new terminal, first, set the VAULT_ADDR environment variable.

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

Check the Vault status.

```
$ vault status
```

The output should be similar to this:

```
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
```

Initialize Vault.

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

The output should be similar to this:

```
Recovery Key 1: E22HrXUFAyQy0PVUy+renVPXoLZ0bSRjWAsTZ64rE24=

Initial Root Token: s.mvX8cihrFqMWEiM9YFnlw9E1

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.
```

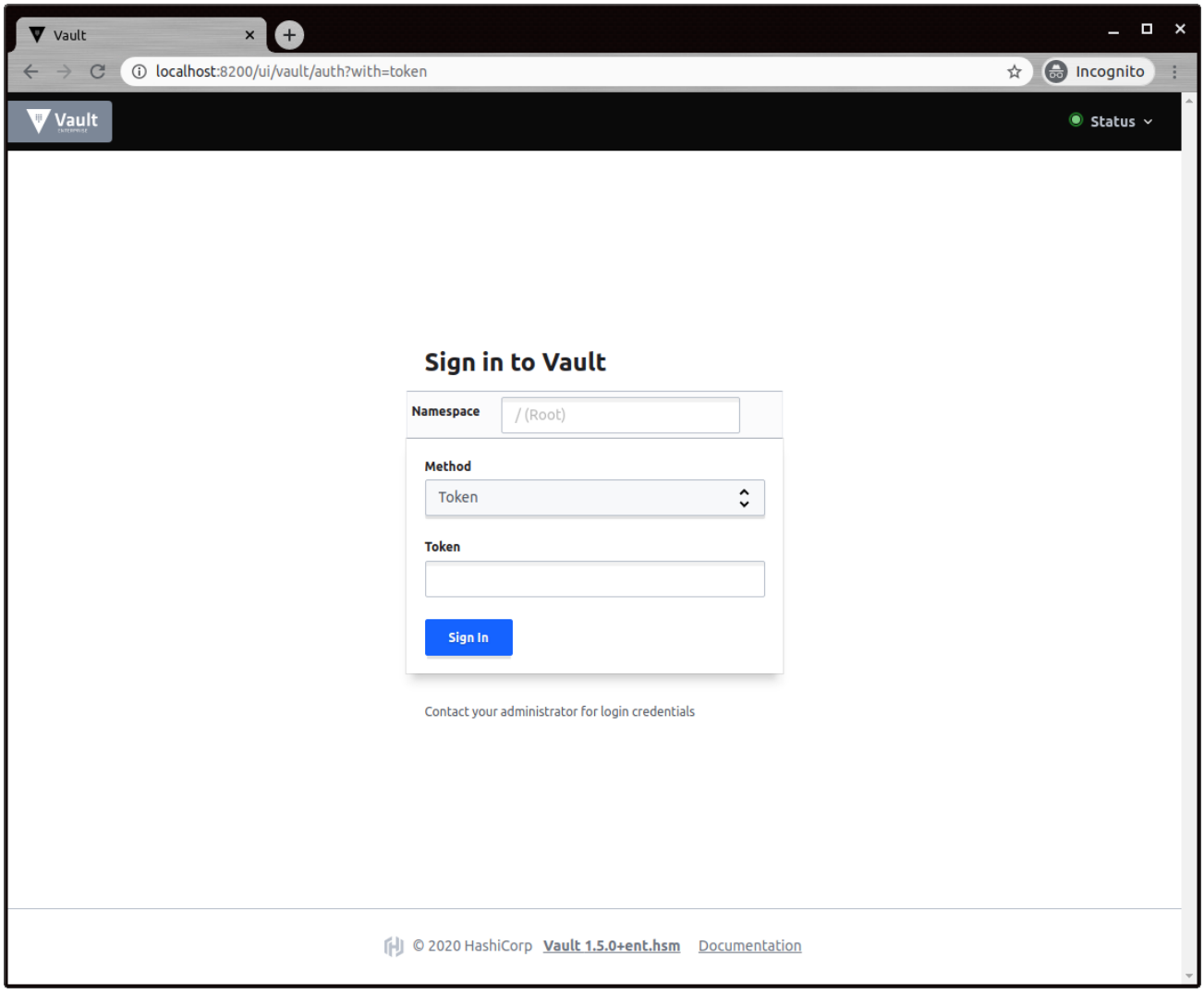
Set the **VAULT_TOKEN** environment variable value to the generated **Root Token** value displayed in the terminal output.

```
$ export VAULT_TOKEN="s.mvX8cihrFqMWEiM9YFnlw9E1"
```

To interact with Vault, you must provide a valid token. Setting this environment variable allows interaction with Vault via the CLI.

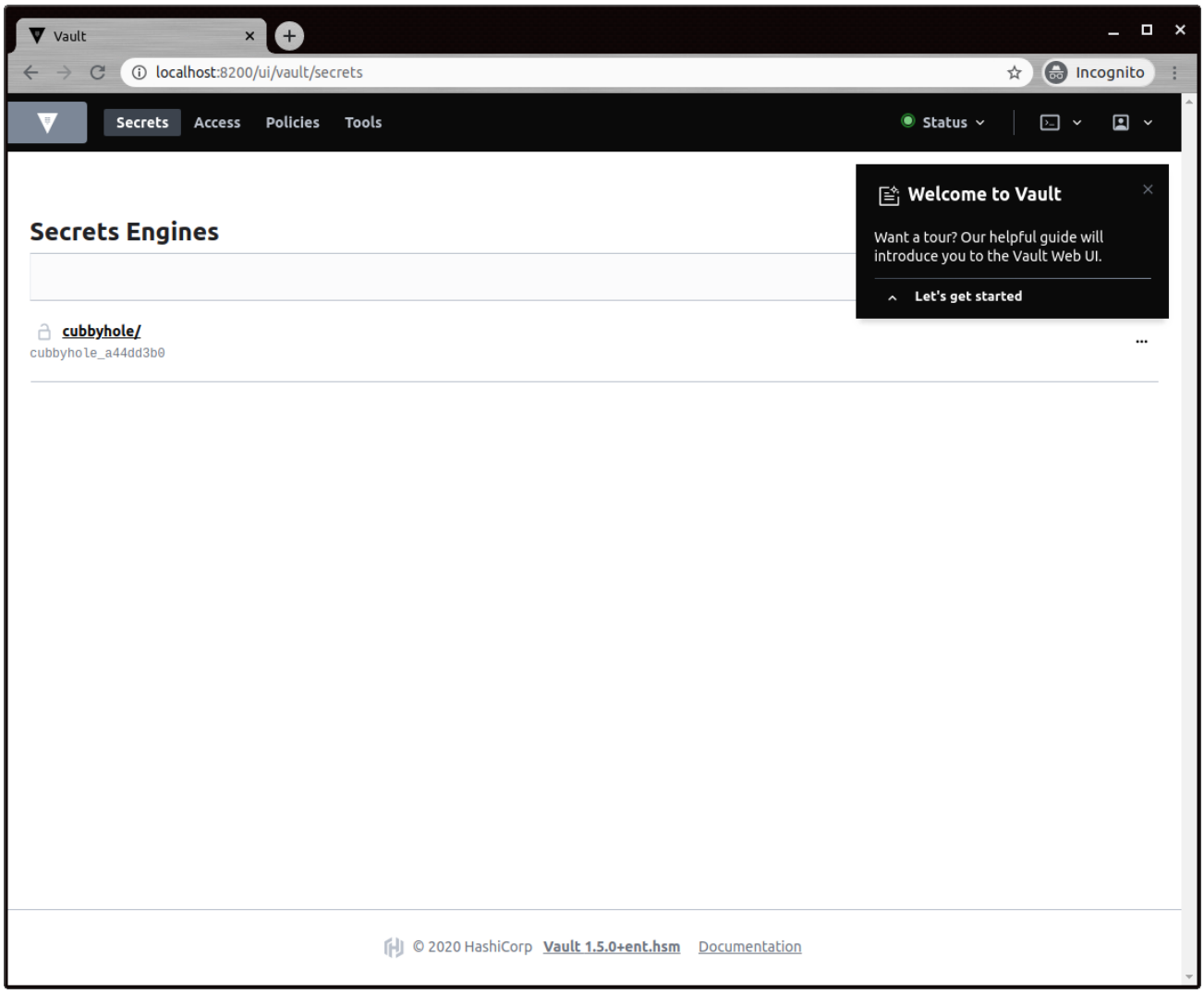
[8.7] ACCESSING THE VAULT UI

Go to http://localhost:8200 in a web browser.



Copy and paste the Initial Root Token that was output from the Vault initialization command into the "Token" field, then click "Sign In".

If the login is successful you will see the Vault UI homepage shown below.



[8.8] ENABLE AND TEST THE SEAL WRAP FEATURE

Enable Seal Wrap

Method 1: CLI command

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

Enable "k/v v1" without seal wrap at *kv-unwrapped*.

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

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
```

To enable seal wrap, pass the "-seal-wrap" flag when you enable a secrets engine.

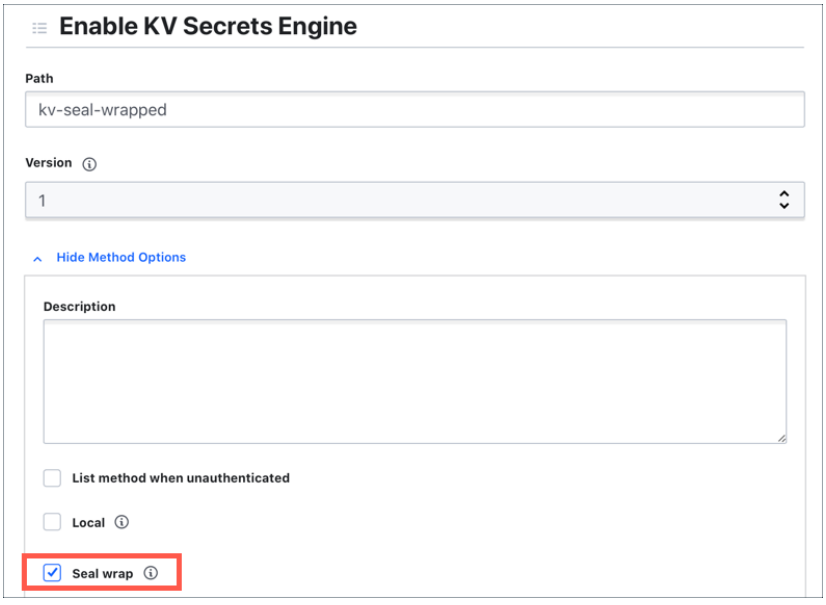
2. List the enabled secrets engines with details.

```
$ vault secrets list -detailed

Path                Plugin       Accessor              ...    Seal Wrap    ...
----                ------       --------                     -----------  ...
cubbyhole/          cubbyhole    cubbyhole_b36dd7e1    ...    false        ...
identity/           identity     identity_b5650a96     ...    false        ...
kv-seal-wrapped/    kv           kv_fe02767b           ...    true         ...
kv-unwrapped/       kv           kv_36d321c6           ...    false        ...
...
```

Notice that the **Seal Wrap** parameter value is "true" for *kv-seal-wrapped/*.

Method 2: Web UI

1. Open a web browser and launch the Vault UI (e.g. http://127.0.0.1:8200/ui) and then login.
2. Select **Enable new engine**.
3. Select **KV** from the list, and then click **Next**.
4. Enter "kv-unwrapped" in the path field and select **Version 1** for KV version.
5. Return to the **Secrets Engines** page and click **Enable Engine**.
6. Select **KV** from the list, and then click **Next**.
7. Enter "kv-seal-wrapped" in the path field. Select **Version 1** for KV version.
8. Click **Method Options** to expand, and select the check box for **Seal Wrap**.



9. Click **Enable Engine**.

Test the Seal Wrap feature

Method 1: CLI command

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.

View the encrypted secrets

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

```
# Configure the storage backend for Vault
storage "file" {
  path = "/tmp/vault"
}
```

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

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

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.

View the secret at rest. One of the directories maps to *kv-unwrapped/unwrapped*.

Example:

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

View its content. The password value is encrypted.

```
$ cat _unwrapped

{"Value":"AAAAAQICk547prhuhMiBXLq2lx8ZkMpSB3p+GKHAwuMhKrZGSeqsFevMS6YoqTVlbvpU9B4zWPZ2HA
SeNZ3YMw=="}
```

Another directory maps to *kv-seal-wrapped/wrapped*.

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

View its content. The password value is encrypted.

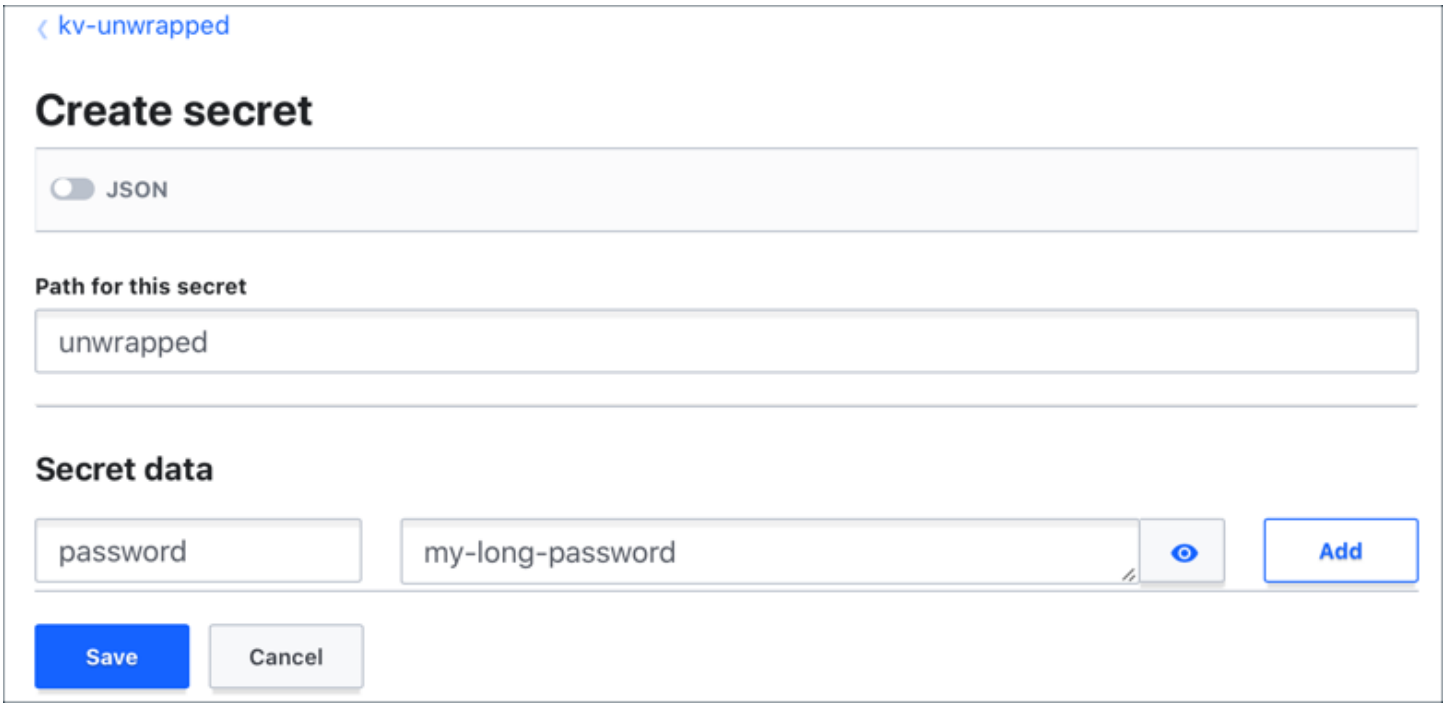
```
$ cat _wrapped

{"Value":"ClBAg9oN7zBBaDBZcsilDAyGkL7soPe7vBA5+ADADuyzo8GuHZHb9UFN2nF1h0OpKEgCIkG3JNHcXt
tZqCi6szcuNBgF3pwhWGwB4FREM3b5CRIQYK7239Q92gRGrcBBeZD6ghogEtSBDmZJBahk7n4lIYF3X4iBqmwZgH
Vo4lzWur7rzncgASofCIIhENEEGghoc21fZGVtbyINaHNtX2htYWNfZGVtb3M="}
```

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

Method 2: Web UI

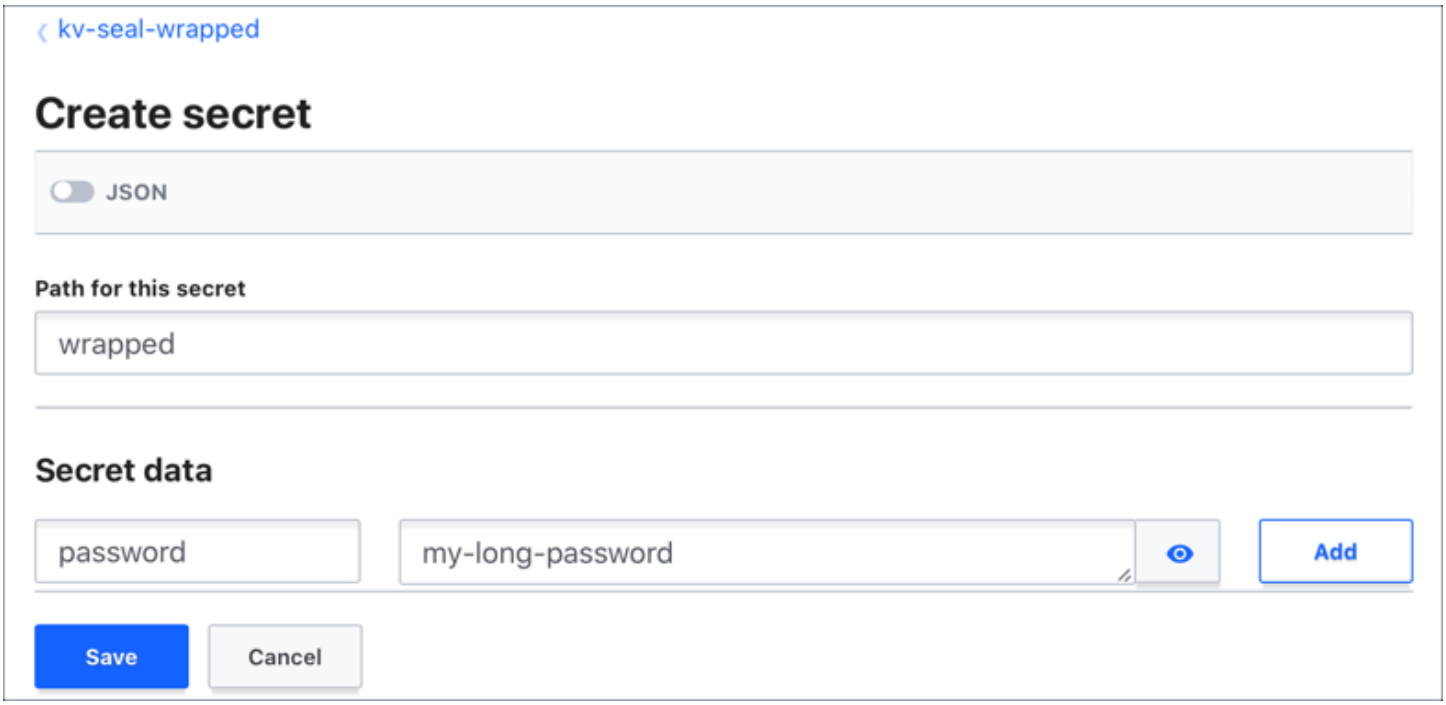
1. Select **kv-unwrapped** and click **Create secret**.

2. Enter "unwrapped" in the **Path for this secret** field, "password" in the **secret key** field, and "my-long-password" in the **value** field.



3. Click **Save**.

4. Repeat the same step for **kv-seal-wrapped** to write the same secret at the *kv-seal-wrapped/wrapped* path.



5. Click **Save**.

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

View the encrypted secrets

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

```
# Configure the storage backend for Vault
storage "file" {
  path = "/tmp/vault"
}
```

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

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

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.

View the secret at rest. One of the directories maps to *kv-unwrapped/unwrapped*.

Example:

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

View its content. The password value is encrypted.

```
$ cat _unwrapped

{"Value":"AAAAAQICk547prhuhMiBXLq2lx8ZkMpSB3p+GKHAwuMhKrZGSeqsFevMS6YoqTVlbvpU9B4zWPZ2HA
SeNZ3YMw=="}
```

Another directory maps to *kv-seal-wrapped/wrapped*.

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

View its content. The password value is encrypted.

```
$ cat _wrapped

{"Value":"ClBAg9oN7zBBaDBZcsilDAyGkL7soPe7vBA5+ADADuyzo8GuHZHb9UFN2nF1h0OpKEgCIkG3JNHcXt
tZqCi6szcuNBgF3pwhWGwB4FREM3b5CRIQYK7239Q92gRGrcBBeZD6ghogEtSBDmZJBahk7n4lIYF3X4iBqmwZgH
Vo4lzWur7rzncgASofCIIhENEEGghoc21fZGVtbyINaHNtX2htYWNfZGVtb3M="}
```

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

[8.9] ENABLE AND TEST THE ENTROPY AUGMENTATION FEATURE

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 external entropy source on a **transit** secrets engine.

NOTE: The Entropy Augmentation feature must be enabled via the CLI. At this time, enabling Entropy Augmentation via the Web UI is not supported.

1. 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
```

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

```
$ vault secrets list -detailed

Path          Plugin       Accessor              ...  External Entropy Access  ...
----          ------       --------              ...  -----------------------  ...
cubbyhole/    cubbyhole    cubbyhole_a4084622    ...  false                    ...
identity/     identity     identity_b5738cb7     ...  false                    ...
sys/          system       system_a8b3552e       ...  false                    ...
transit/      transit      transit_88cd3066      ...  true                     ...
```

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

3. 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.

Example:

Create a new encryption key named, "orders".

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

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:AY3ZF2bwGfwZ9dJLSztCLdpPUHkfl/kwaQeRITvKgn74bGYyMI+n34w1CMO8aeg=
```

Now, test to verify that you can decrypt.

```
$ vault write transit/decrypt/orders \
        ciphertext="vault:v1:AY3ZF2bwGfwZ9dJLSztCLdpPUHkfl/kwaQeRITvKgn74bGYyMI+n34w1CMO8aeg="
```

Decode to get the original data.

```
$ base64 --decode <<< Y3JlZGl0LWNhcmQtbnVtYmVyCg==

credit-card-number
```

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 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
```

APPENDIX A: XCEPTIONAL SUPPORT

In today's high-paced environment, we know you are looking for timely and effective resolutions for your mission-critical needs. That is why our Xceptional Support Team will help do whatever it takes to ensure you have the best experience and support possible. Every time. Guaranteed.

- 24x7x365 mission critical support
- Level 1 to level 3 support
- Extremely knowledgeable subject matter experts

At Futurex, we strive to supply you with the latest data encryption innovations as well as our best-in-class support services. Our Xceptional Support Team goes above and beyond to meet your needs and provide you with exclusive services that cannot be found anywhere else in the industry.

- Technical Services
- Onsite Training
- Virtual Training
- Customized Consulting
- Customized Software Solutions
- Secure Key Generation, Printing, and Mailing
- Remote Key Injection
- Certificate Authority Services

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

ENGINEERING CAMPUS

864 Old Boerne Road

Bulverde, Texas, USA 78163

Phone: +1 830-980-9782

+1 830-438-8782

E-mail: info@futurex.com

XCEPTIONAL SUPPORT

24x7x365

Toll-Free: 1-800-251-5112

E-mail: support@futurex.com

SOLUTIONS ARCHITECT

E-mail: solutions@futurex.com