JAVA JARSIGNER

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 Java Jarsigner using PKCS #11 libraries. For additional questions related to your HSM, see the relevant administrator's guide.

[1.2] APPLICATION DESCRIPTION

From Oracle's documentation website: "Java's `jarsigner` tool is used for two purposes:

1. To sign Java ARchive (JAR) files.
2. To verify the signatures and integrity of signed JAR files.

The JAR feature enables the packaging of class files, images, sounds, and other digital data in a single file for faster and easier distribution. A tool named `jar` enables developers to produce JAR files. (Technically, any zip file can also be considered a JAR file, although when created by the `jar` command or processed by the `jarsigner` command, JAR files also contain a `META-INF/MANIFEST.MF` file.)

A digital signature is a string of bits that is computed from some data (the data being signed) and the private key of an entity (a person, company, and so on). Similar to a handwritten signature, a digital signature has many useful characteristics:

- Its authenticity can be verified by a computation that uses the public key corresponding to the private key used to generate the signature.
- It cannot be forged, assuming the private key is kept secret.
- It is a function of the data signed and thus cannot be claimed to be the signature for other data as well.
- The signed data cannot be changed. If the data is changed, then the signature cannot be verified as authentic.

To generate an entity's signature for a file, the entity must first have a public/private key pair associated with it and one or more certificates that authenticate its public key. A certificate is a digitally signed statement from one entity that says that the public key of another entity has a particular value.

The `jarsigner` command uses key and certificate information from a keystore to generate digital signatures for JAR files. A keystore is a database of private keys and their associated X.509 certificate chains that authenticate the corresponding public keys. The `keytool` command is used to create and administer keystores.

The `jarsigner` command uses an entity's private key to generate a signature. The signed JAR file contains, among other things, a copy of the certificate from the keystore for the public key corresponding to the private key used to sign the file. The `jarsigner` command can verify the digital signature of the signed JAR file using the certificate inside it (in its signature block file).

The `jarsigner` command can generate signatures that include a time stamp that lets a systems or deployer (including Java Plug-in) to check whether the JAR file was signed while the signing certificate was still valid. In addition, APIs allow applications to obtain the timestamp information.

At this time, the `jarsigner` command can only sign JAR files created by the `jar` command or zip files. JAR files are the same as zip files, except they also have a `META-INF/MANIFEST.MF` file. A `META-INF/MANIFEST.MF` file is created when the `jarsigner` command signs a zip file.

The default `jarsigner` command behavior is to sign a JAR or zip file. Use the `-verify` option to verify a signed JAR file.

The `jarsigner` command also attempts to validate the signer's certificate after signing or verifying. If there is a validation error or any other problem, the command generates warning messages. If you specify the `-strict` option, then the command treats severe warnings as errors. See Errors and Warnings."

[1.3] 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

Other:

- OpenSSL
- Java Development Kit (JDK) 8

[3] INSTALL FUTUREX PKCS #11 (FXPKCS11)

In a Windows environment, the easiest way to install the **Futurex PKCS #11 (FXPKCS11)** module is with **Futurex Tools (FXTools)**. You can download FXTools from the Futurex Portal. In a Linux environment, you must download a tarball of the FXPKCS11 binaries from the Futurex Portal and then extract the tar file locally where you want the application to be installed on your system. The following sections provide step-by-step installation instructions for both of these scenarios.

Note: Install FXPKCS11 on the same computer as the application integrating with the Vectera Plus HSM.

[3.1] INSTALLING THE FXPKCS11 MODULE USING FXTOOLS IN WINDOWS

Run the Futurex Tools installer as an administrator and follow the prompts in the setup wizard to complete the installation.

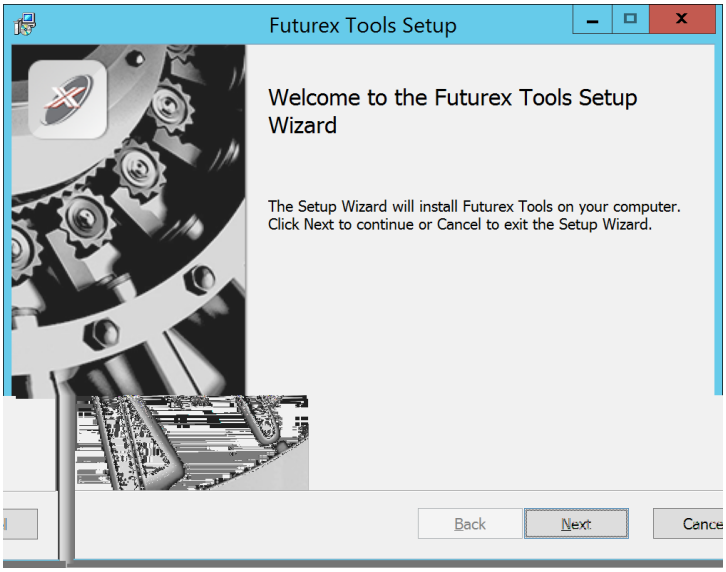


FIGURE: FUTUREX TOOLS SETUP WIZARD

The Setup Wizard installs all tools on the system by default. You can override the defaults and choose not to install certain modules. The installation provides the following services:

- **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**- A client used to connect a Futurex Excrypt Touch to a local laptop through USB, which can then connect to a remote Futurex device.

If the Futurex Secure Access Client was selected, the process will also install the Futurex Excrypt Touch driver, which might start minimized or in the background.

After the installation completes, 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, which are located in their corresponding directory with a .cfg extension. In addition, the installation registers the CNG and CSP Modules in the Windows Registry (HKEY_LOCAL_ MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider), and installs them in the C:\Windows\System32\ directory.

[3.2] INSTALLING THE FXPKCS11 MODULE IN LINUX

Extract the tarball file for your Linux distrubution in the desired working directory.

Note: To make the Futurex PKCS #11 module accessible system-wide, move it to the /usr/local/bin directory as an administrative user. If only the current user needs to use the module, then install it in $HOME/bin.

The extracted content of the tar file is a single fxpkcs11 directory. Inside the fxpkcs11 directory is the following files and directories:

- **fxpkcs11.cfg**: FXPKCS11 configuration file
- **x86/**: This folder contains the module files for 32-bit architecture
- **x64/**: This folder contains the module files for 64-bit architecture

The x86 and x64 directories each contain two subdirectories, OpenSSL-1.0.x and OpenSSL-1.1.x. These OpenSSL directories contain the following FXPKCS11 module files built with the respective OpenSSL versions:

- **configTest**: Program to test configuration and connection to the HSM
- **libfxpkcs11.so**: FXPKCS11 Library File
- **libfxpkcs11-Debug.so**: FXPKCS11 Debug Library File
- **PKCS11Manager**: Program to test connection and manage the HSM through the FXPKCS11 library

By default, the FXPKCS11 module looks for the FXPKCS11 configuration file (i.e., fxpkcs11.cfg) in the /etc directory. Alternatively, a system environment variable can be defined for the location of the FXPKCS11 configuration file. To do so permanently, open the /etc/profile file in a text editor as an administrative user, add the following line at the bottom, and save the file.

```
export FXPKCS11_CFG=/usr/local/bin/fxpkcs11/fxpkcs11.cfg
```

Note: The file location specified above must be specific to where the FXPKCS11 configuration file is saved on your system.

[4] INSTALL EXCRYPT MANAGER (IF USING WINDOWS)

Sections 4 and 5 of this integration guide cover the installation of Excrypt Manager and FXCLI. Excrypt Manager is a Windows application that provides a GUI-based method for configuring the HSM, while FXCLI provides a command-line-based method for configuring the HSM and can be installed on all platforms.

Note: If you will be configuring the Vectera Plus from a Linux computer, you can skip this section. If you will be configuring the Vectera Plus from a Windows computer, installing FXCLI in the next section is still required because FXCLI is the only method that can be used to configure TLS certificates in section 6.7.

Note: Install Excrypt Manager on the workstation you will use to configure the HSM.

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.

To install Excrypt Manager, run the Excrypt Manager installer as an administrator and follow the prompts in the setup wizard to complete the installation.

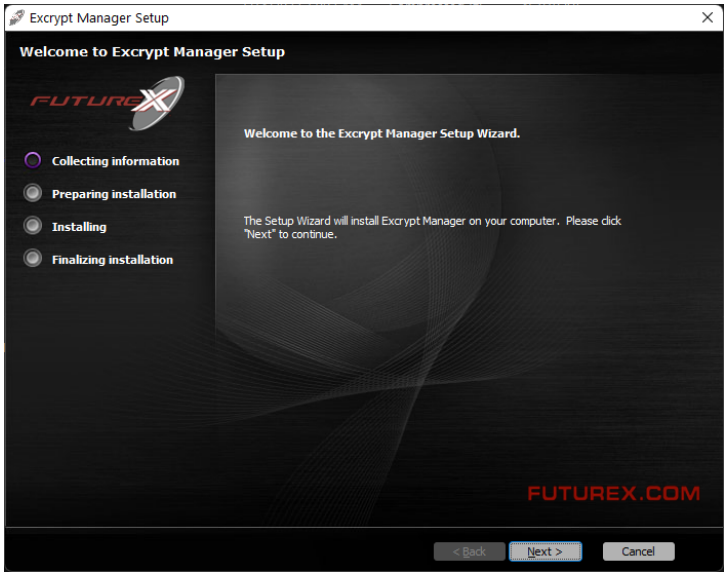


FIGURE: EXCRYPT MANAGER SETUP WIZARD

The installation wizard prompts you to specify where you want to install Excrypt Manager. The default location is C:\Program Files\Futurex\Excrypt Manager\. After choosing a location, select **[Install]**.

[5] INSTALL FUTUREX COMMAND LINE INTERFACE (FXCLI)

Note: Install FXCLI on the workstation you will use to configure the HSM.

[5.1] INSTALLING FXCLI IN WINDOWS

As mentioned in section 3, the FXTools installation package includes Futurex Client Tools (FXCLI). Similar to the Futurex PKCS #11 (FXPKCS11) module, the easiest way to install FXCLI on Windows is by installing FXTools. You can download FXTools from the Futurex Portal.

To install FXCLI, run the Futurex Tools installer as an administrator and follow the prompts in the setup wizard to complete the installation.

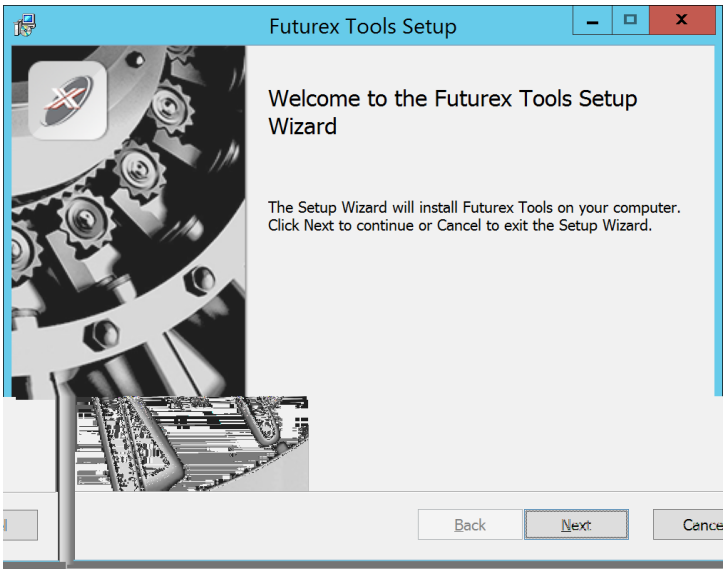


FIGURE: FUTUREX TOOLS SETUP WIZARD

The setup wizard installs all tools on the system by default. You can override the defaults and choose not to install certain modules. The installation provides the following services:

- **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**:A client used to connect a Futurex Excrypt Touch to a local laptop through USB, which can then connect to a remote Futurex device.

[5.2] INSTALLING FXCLI IN LINUX

Download FXCLI

You can download the appropriate FXCLI package files for your system from the Futurex Portal.

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

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

To install an rpm package, run the following command in a terminal:

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

To install a deb package, run the following command in a terminal:

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

Running FXCLI

To enter the HSM FXCLI prompt, run the following command in a terminal:

```
$ fxcli-hsm
```

After entering the FXCLI prompt, you can run **help** to list all of the available FXCLI commands.

[6] SETTING SYSTEM ENVIRONMENT VARIABLES FOR THE JAVA LIBRARY

System environment variables must be defined for the location of the Java library. The variable settings are:

- JAVA_HOME = path to JDK installation directory
- JRE_HOME = path to JDK installation directory
- PATH = ; (add all the paths described above)

Windows example:

- JAVA_HOME = C:\Program Files\Java\jdk1.8.0_301
- JRE_HOME = C:\Program Files\Java\jdk1.8.0_301
- PATH = …; C:\Program Files\Java\jdk1.8.0_301; C:\Program Files\Java\jdk1.8.0_301\bin;

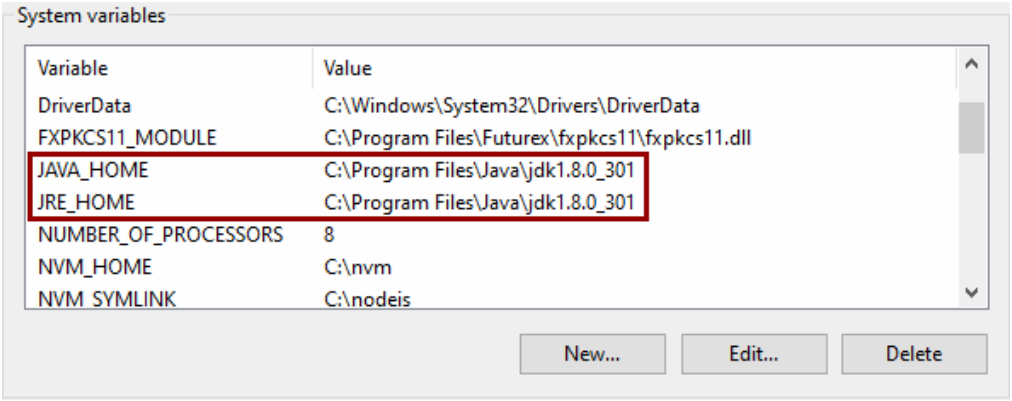


FIGURE: EXAMPLE SYSTEM VARIABLE SETTINGS

Linux example:

To define system environment variables for path to the JDK installation directory in Linux, open the `/etc/profile` file in a text editor and add the following lines at the bottom:

```
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
JRE_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre

PATH=$PATH:$JAVA_HOME/bin

export JAVA_HOME
export JRE_HOME
export PATH
```

[7] INSTALL FXJCE FILES

The Java provider relies on a JNI (Java Native Interface) library, which must be in the server's *$JAVA_HOME/jre/lib* directory. It also requires a provider, which should be saved in the *$JAVA_HOME/jre/lib/ext* directory.

Extract the files from the zip file (*fxjce-OperatingSystem_x.xx.zip*) corresponding to the operating system in the working folder. Examples for each operating system are below:

Linux:

libfxjp11.so (library) -> */usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext*

sunpkcs11-fx.jar (extension) -> */usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext*

Windows:

fxjp11.dll (library) -> *C:\Program Files\Java\jdk1.8.0_301\jre\lib\ext*

sunpkcs11-fx.jar (extension) -> *C:\Program Files\Java\jdk1.8.0_301\jre\lib\ext*

[8] REGISTERING THE JAVA PROVIDER

The Java provider must be registered before it can be used. To register, modify the *java.security* file on the system (typically located in *$JAVA_HOME/jre/lib/security*). Append a line similar to the following to the provider list in the *java.security* file:

```
security.provider.11=fx.security.pkcs11.SunPKCS11
```

```
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=sun.security.ec.SunEC
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
security.provider.7=com.sun.security.sasl.Provider
security.provider.8=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.9=sun.security.smartcardio.SunPCSC
security.provider.10=sun.security.mscapi.SunMSCAPI
security.provider.11=fx.security.pkcs11.SunPKCS11
```

FIGURE: SAMPLE JAVA.SECURITY FILE

[9] 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. Please refer to the applicable guide for configuring HSMs with the Guardian Series 3.

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

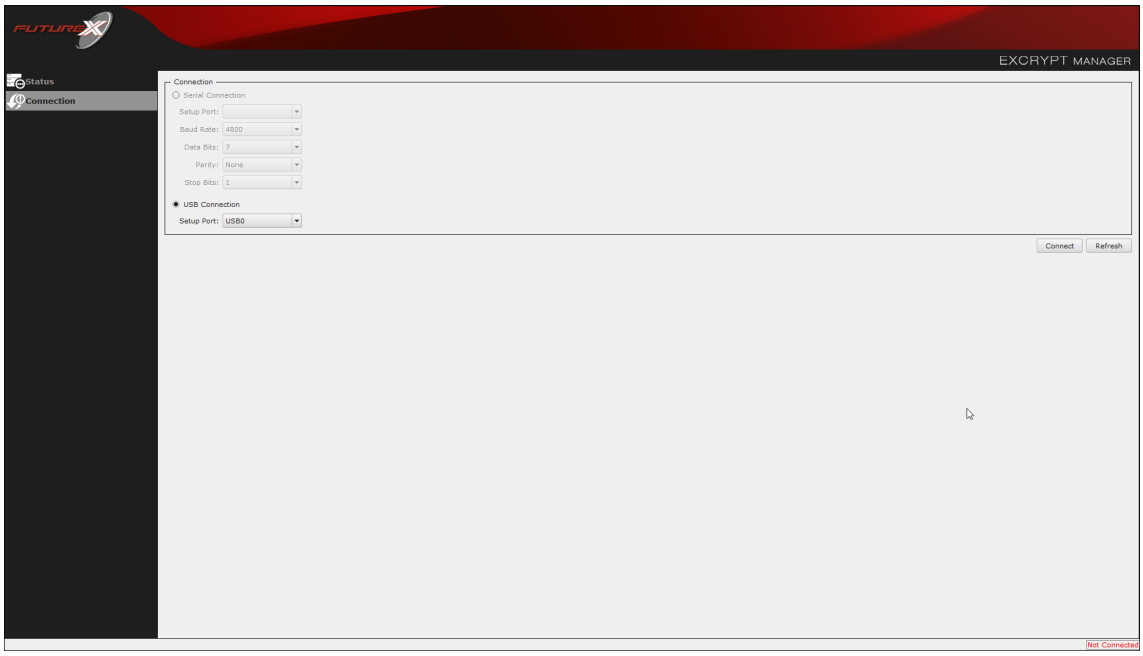
Each of these action items is detailed in the following subsections.

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

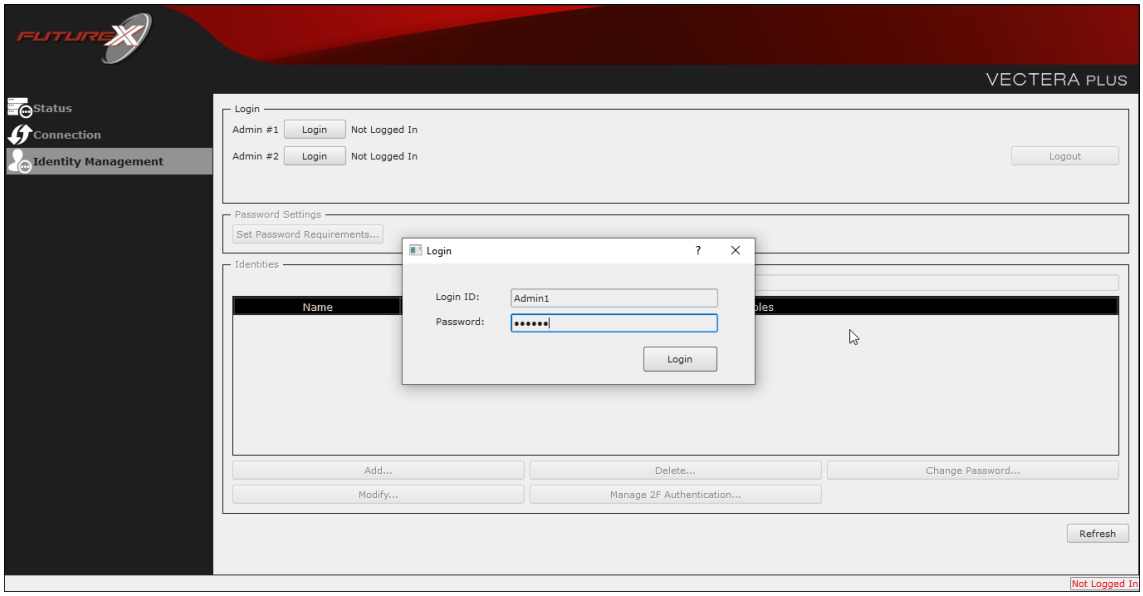
For both Excrypt Manager and FXCLI you need to connect your laptop to the front USB port on the HSM.

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”.

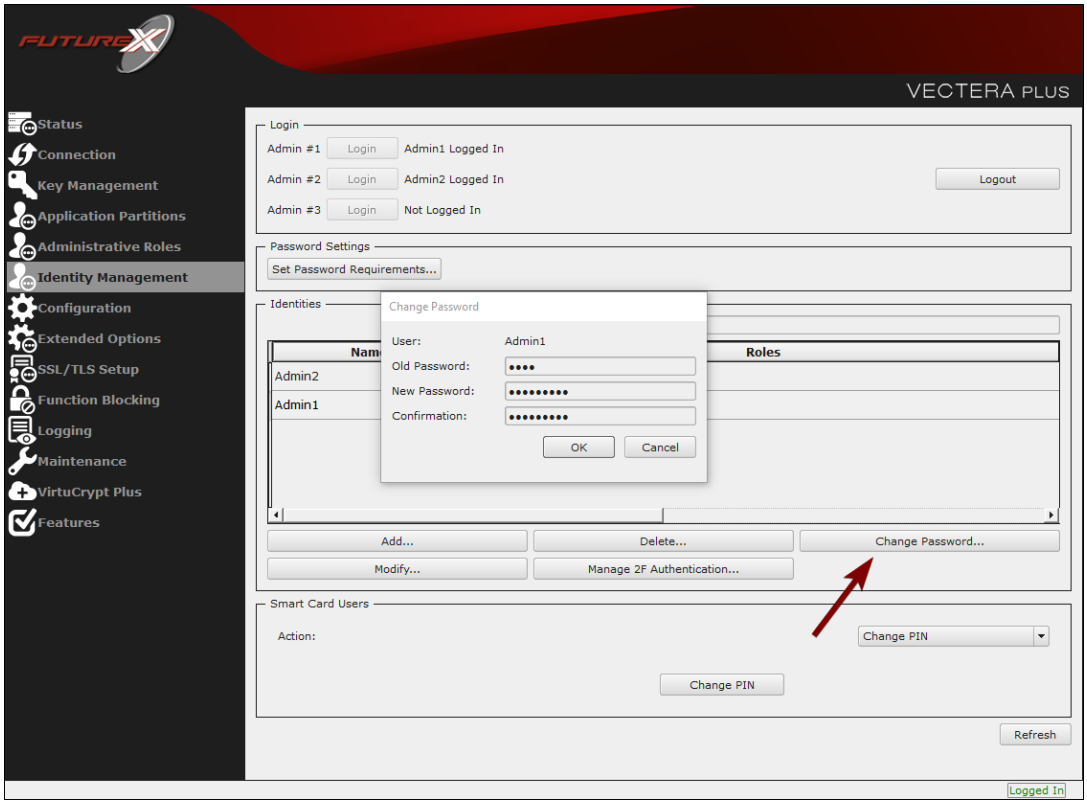


Login with both default Admin identities.



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.

[9.2] 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)

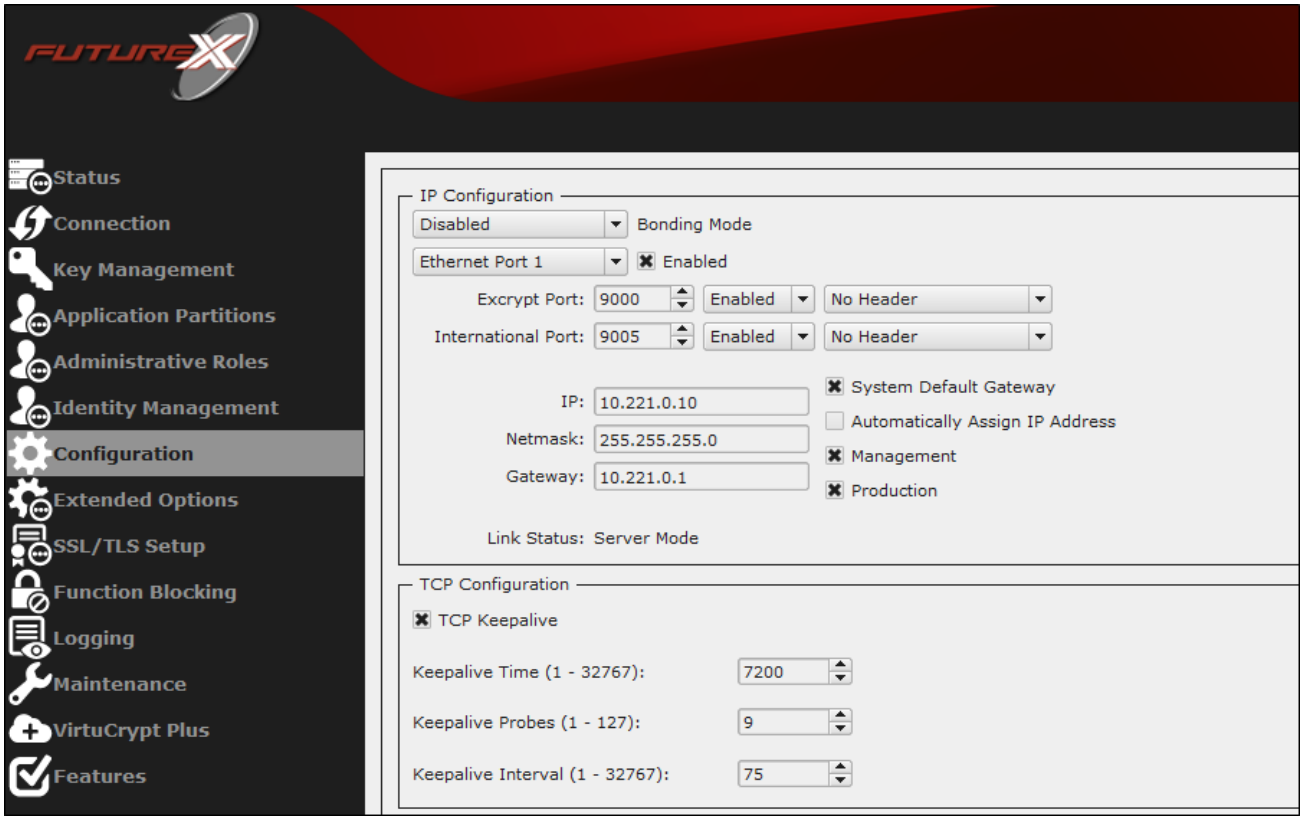
NOTE: For additional information about how to update features on your HSM, please refer to your HSM Administrator's Guide, section **"Download Feature Request File"**.

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.

[9.3] 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 the applicable guide for configuring HSMs with the Guardian Series 3), 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.

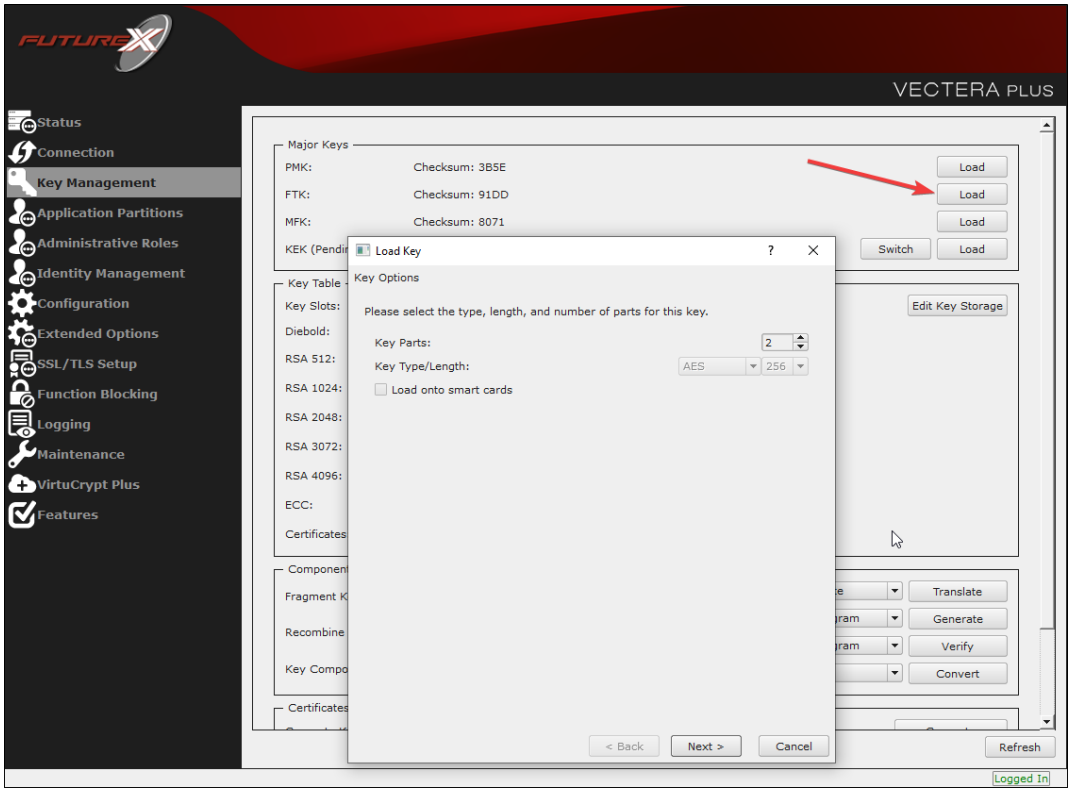
[9.4] LOAD FUTUREX KEY (FTK)

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
```

[9.5] 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.

Under the "Permissions" tab, leave the top-level **Keys** permission checked, but uncheck the **All Slots** sub permission.

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:

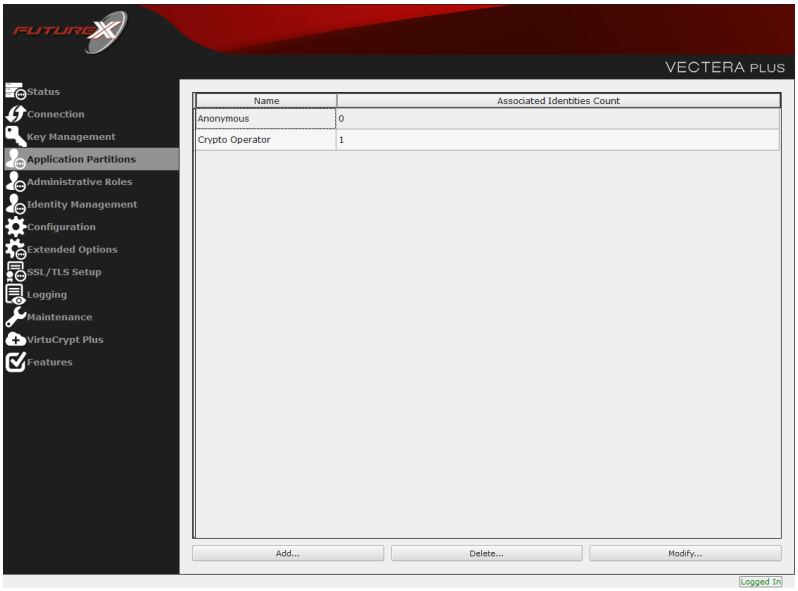
```
$ role modify --name Anonymous --clear-perms --clear-key-ranges
```

```
$ role modify --name Anonymous --add-perm "Keys" --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
```

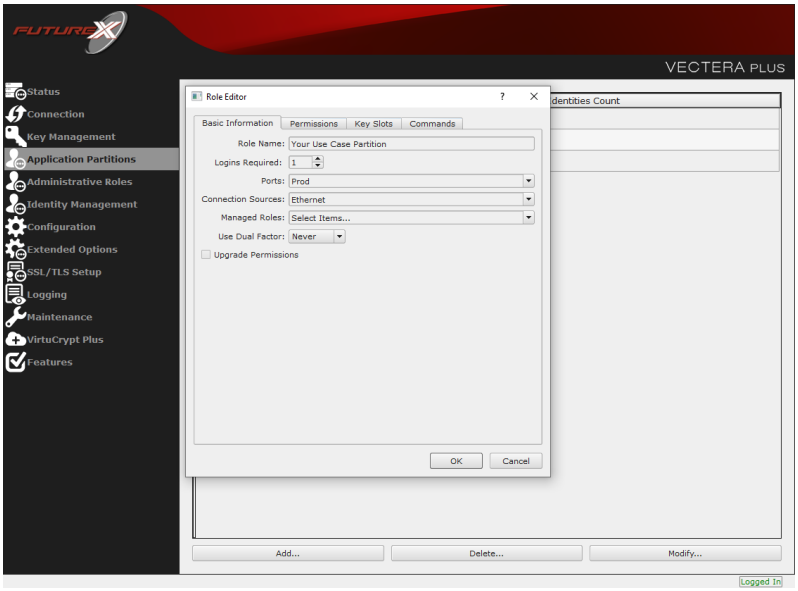
Create an Application Partition

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:

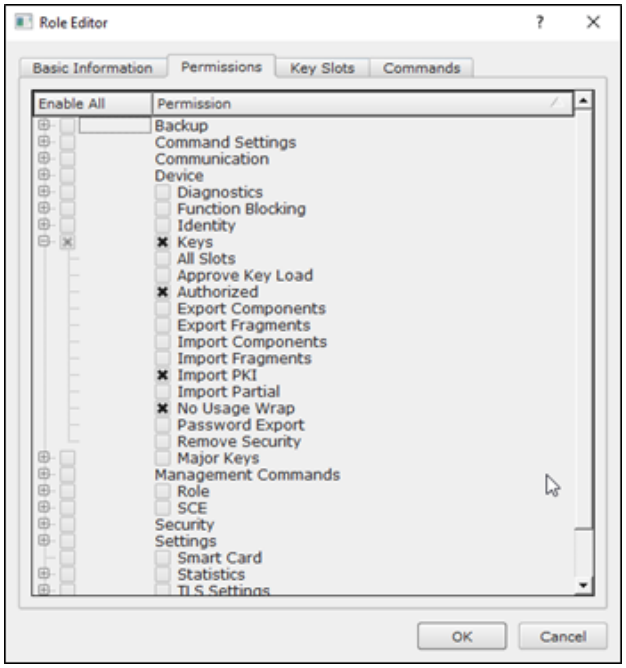
Navigate to the *Application Partitions* page and click the "Add" button at the bottom.



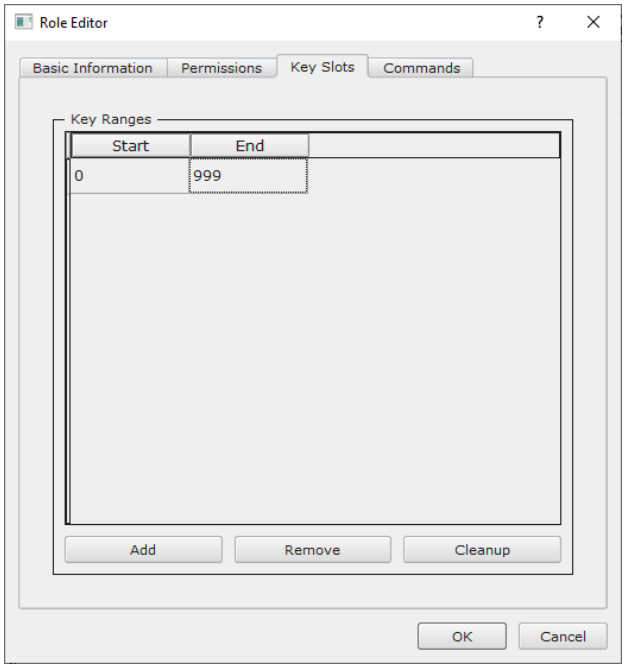
Fill in all of the fields in the *Basic Information* tab exactly how you see below (except for the *Role Name* field). In the *Role Name* field, specify any name that you would like for this new Application Partition. *Logins Required* should be set to "1". *Ports* should be set to "Prod". *Connection 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 commands that need to be enabled to perform the example Keytool and Jarsigner commands in the last two sections of the integration guide are listed below. These can be enabled under the "Commands" tab.

PKCS #11 Communication Commands

- **ECHO**: Communication Test/Retrieve Version
- **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

- **GPCA**: General Purpose Add Certificate to Key Table
- **GPKD**: General Purpose Key Slot Delete/Clear
- **GRSA**: Generate RSA Private and Public Key
- **LRSA**: Load key into RSA Key Table

Data Encryption Commands

- **GPSV**: General purpose data sign and verify

Miscellaneous Commands

- **TIME**: Set the HSM internal clock

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

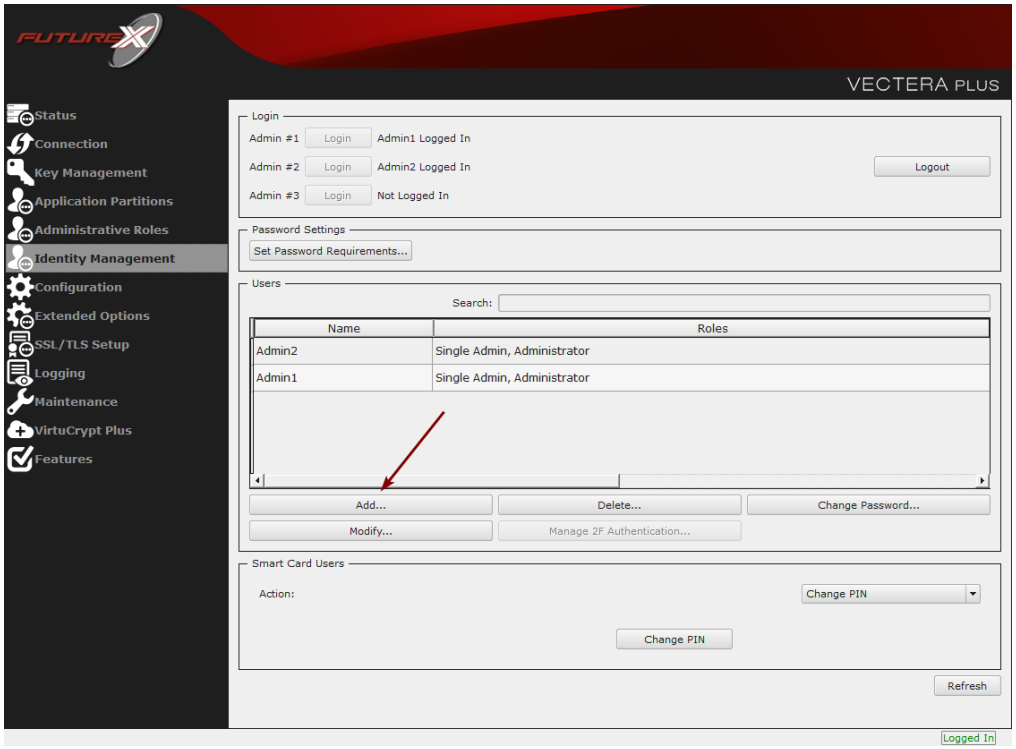
```
$ role add --name Role_Name --application --key-range (0,999) --perm "Keys:Authorized" --perm "Key-
s:Import PKI" --perm "Keys:No Usage Wrap"
```

```
$ role modify --name [role_name] --clear-perms --add-perm Excrypt:ECHO --add-perm Excrypt:HASH --
add-perm Excrypt:GPKM --add-perm Excrypt:GPKS --add-perm Excrypt:GPKR --add-perm Excrypt:GPCA --
add-perm Excrypt:GPKD --add-perm Excrypt:GRSA --add-perm Excrypt:LRSA --add-perm Excrypt:GPSV --
add-perm Excrypt:TIME
```

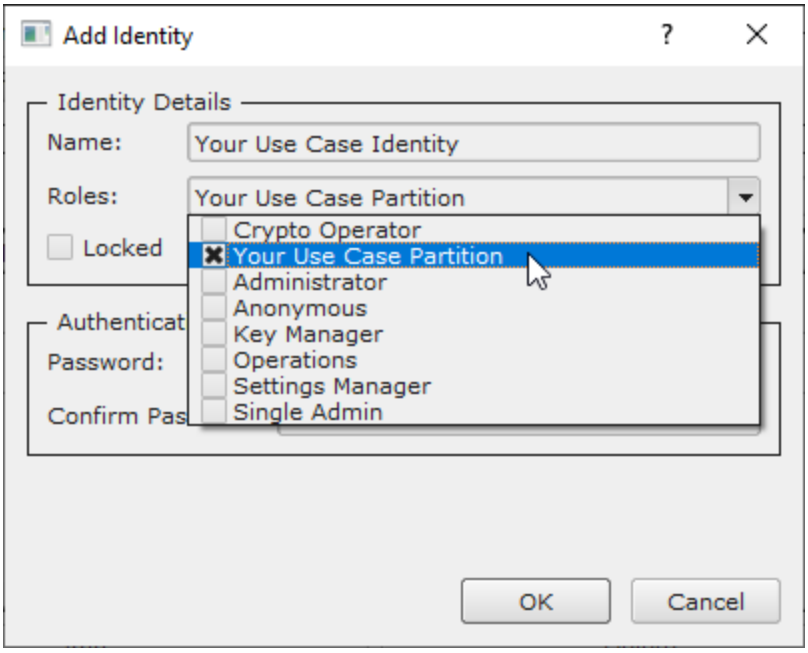
[9.6] CREATE NEW IDENTITY AND ASSOCIATE IT WITH THE NEWLY CREATED APPLICATION PARTITION

*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>
```

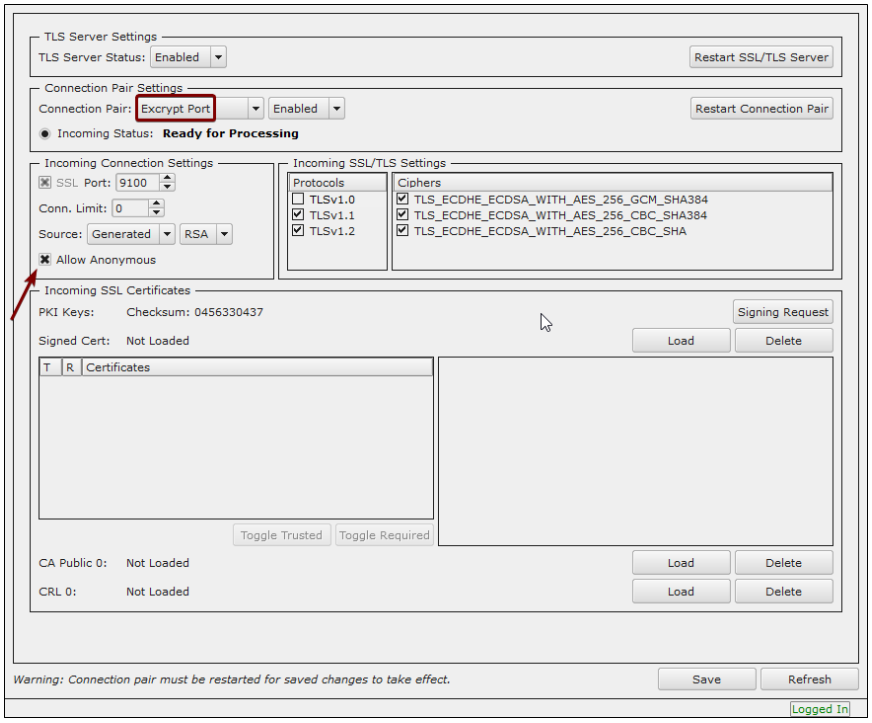
NOTE: Crypto Operator in the fxpkcs11.cfg file must match exactly the name of the identity created in the HSM.

[9.7] 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 pass-
word. 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.

```
# 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 -inkey privatekey.pem -in SignedPki.pem -certfile TlsCa.pem -out PKI.p12
```

[10] EDIT THE FUTUREX PKCS #11 CONFIGURATION FILE

The Futurex PKCS #11 configuration file (i.e., fxpkcs11.cfg) is used by the Futurex PKCS #11 library to connect to the HSM. It enables the user to modify certain configurations and set connection details. This section covers the **<HSM>** portion of the FXPKCS11 config file, where the connection details are set.

Note: By default, the FXPKCS11 library looks for the configuration file at C:\Program Files\Futurex\fxpkcs11\fxpkcs11.cfg for Windows and /etc/fxpkcs11.cfg for Linux. Alternatively, the FXPKCS11_CFG environment variable can be set to the location of the fxpkcs11.cfg file.

Open the fxpkcs11.cfg file in a text editor as an administrator and edit it accordingly.

```
<HSM>
    # Which PKCS11 slot
    <SLOT>                  0                       </SLOT>
    <LABEL>                 Futurex                 </LABEL>

    # HSM crypto operator user name
    <CRYPTO-OPR>            [identity_name]             </CRYPTO-OPR>
    # Automatically login on session open
    #<CRYPTO-OPR-PASS>       [identity_password]          </CRYPTO-OPR-PASS>

    # Connection information
    <ADDRESS>               10.0.8.30     </ADDRESS>
    <PROD-PORT>             9100                    </PROD-PORT>
    <PROD-TLS-ENABLED>      YES                     </PROD-TLS-ENABLED>
    <PROD-TLS-ANONYMOUS>    NO                      </PROD-TLS-ANONYMOUS>
#     <PROD-TLS-CA>          /home/user/tls/root.pem        </PROD-TLS-CA>
#     <PROD-TLS-CA>          /home/user/tls/sub1.pem     </PROD-TLS-CA>
#     <PROD-TLS-CA>          /home/user/tls/sub2.pem     </PROD-TLS-CA>
    <PROD-TLS-KEY>          /home/user/tls/PKI.p12      </PROD-TLS-KEY>
    <PROD-TLS-KEY-PASS>     safest                  </PROD-TLS-KEY-PASS>

    # YES = This is communicating through a Guardian
    <FX-LOAD-BALANCE>       NO                      </FX-LOAD-BALANCE>
</HSM>
```

The **<SLOT>** and **<LABEL>** fields specify PKCS11 slot 0 and the label *Futurex*.

In the **<CRYPTO-OPR>** field, specify the name of the identity you created for the Application Partition.

The **<CRYPTO-OPR-PASS>** field allows you to specify the password of the identity configured in the **<CRYPTO-OPR>** field. This can be used to log the application into the HSM automatically, if required.

In the **<ADDRESS>** field, specify the IP address of the HSM that the FXPKCS11 library should connect to.

In the **<PROD-PORT>** field, specify the port number of the HSM that the FXPKCS11 library should connect to.

The **<PROD-TLS-ENABLED>** field should be set to *YES*.

The **<PROD-TLS-ANONYMOUS>** field defines whether the FXPKCS11 library authenticates to the server.

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, the signed client cert does not need to be defined with the **<PROD-TLS-CERT>** tag, nor do the CA cert/s need to be defined with one or more instances of the **<PROD-TLS-CA>** tag.

If you use Guardian to manage HSMs in a cluster, define the **<FX-LOAD-BALANCE>**field as *YES*. Otherwise, set it to *NO*.

After you finish editing the fxpkcs11.cfg file, run the PKCS11Manager file to test the connection against the HSM and check the fxpkcs11.log for errors and information. For more information, refer to the Futurex PKCS #11 technical reference found on the Futurex Portal.

[11] JAVA KEYSTORE CREATION

In this section, Java **keytool** commands will be used to generate a new key pair on the Vectera Plus, create a Certificate Signing Request (CSR), issue a certificate by means of an internal or external CA, then import the signed certificate and it's accompanying CA certificate into a Java keystore.

The purpose of these steps is so that the signed certificate can be used to sign a JAR file in the next section using **jarsigner**.

NOTE: The Keytool application is included in the JDK 8 installation, so no additional configuration is required to run the following Keytool commands.

[11.1] GENERATE A SERVER KEY PAIR AND SELF-SIGNED CERTIFICATE

```
$ keytool -genkeypair -keyalg RSA -keysize 2048 -alias JarsignerDemo -keystore NONE -storetype
PKCS11 -providername "Futurex" -providerclass "fx.security.pkcs11.SunPKCS11" -ext
extendedKeyUsage=codeSigning -ext KeyUsage=digitalSignature
```

NOTE: *-alias* is a field used to set a name to identify the key pair and certificate to be generated. It can be any name (example: *JarsignerDemo*).

Upon the execution of the previous instruction, the Keytool application will ask for information for the server certificate to be generated.

Enter the KeyStore password: (The password that you set here will be used in all keytool and jarsigner commands moving forward.)

1. What is your first and last name?

[Unknown]: www.example.com

2. What is the name of your organizational unit?

[Unknown]: Engineering

3. What is the name of your organization?

[Unknown]: Futurex

4. What is the name of your City or Locality?

[Unknown]: Bulverde

5. What is the name of your State or Province?

[Unknown]: TX

6. What is the two-letter country code for this unit?

[Unknown]: US

7. Is CN=www.example.com, OU=Engineering, O=Futurex, L=Bulverde, ST=TX, C=US correct?

[no]: yes

[11.2] GENERATE AND EXPORT A CSR

```
$ keytool -certreq -alias JarsignerDemo -file example.csr -keystore NONE -storetype PKCS11 -pro-
vidername "Futurex" -providerclass "fx.security.pkcs11.SunPKCS11"
```

Enter KeyStore password:

The CSR must be signed by a CA, either third-party or internal. Once signed, the certificate returned by the CA will be imported along with the CA certificate.

[11.3] IMPORT THE CA ROOT CERTIFICATE

```
$ keytool -import -trustcacerts -alias JarsignerDemoCA -keystore NONE -file ssl-ca-cert.pem -store-
type PKCS11 -providername "Futurex" -providerclass "fx.security.pkcs11.SunPKCS11"
```

Enter KeyStore password:

1. Trust this certificate?

[no]: yes

2. Certificate was added to KeyStore

[11.4] IMPORT THE SIGNED CERTIFICATE (CERTIFICATE SIGNED BY CA)

```
$ keytool -importcert -alias JarsignerDemo -keystore NONE -file signed-example-cert.pem -storetype
PKCS11 -providername "Futurex" -providerclass "fx.security.pkcs11.SunPKCS11"
```

Enter the KeyStore password:

Certificate reply was installed in KeyStore.

[12] JARSIGNER COMMAND EXAMPLES

As mentioned in the Document Information section at the beginning of the guide, Java's `jarsigner` tool is used for two purposes:

1. To sign Java ARchive (JAR) files.
2. To verify the signatures and integrity of signed JAR files.

Examples of both are provided in the subsections that follow.

[12.1] SIGNING A JAVA ARCHIVE (JAR) FILE

Before attempting to sign a Java ARchive (JAR) file, it is a good practice to run the following `keytool` command to ensure that the keys stored on the HSM needed for signing are accessible:

```
$ keytool -keystore NONE -storetype PKCS11 -list
```

The response should be similar to the following:

```
Keystore type: PKCS11
Keystore provider: Futurex

Your keystore contains 2 entries

JarsignerDemo, PrivateKeyEntry,
Certificate fingerprint (SHA-256):
1F:1F:44:11:C2:6C:35:93:B8:DF:D9:32:8A:39:2D:96:99:42:DA:DF:39:D5:F3:D0:93:EA:77:91:5A:ED:80:CE

JarsignerDemoCA, trustedCertEntry,
Certificate fingerprint (SHA-256):
9F:B7:23:3C:20:5A:4B:59:C1:25:F9:11:76:21:EA:6E:4A:79:EF:1A:6C:17:45:A6:D8:37:1C:59:E2:6B:C3:02
```

Now that we've confirmed the keys needed for code signing are accessible, run the following command to sign a JAR file using the HSM-stored keys (**NOTE:** The command needs to be run from the same directory where the `example.jar` file is stored.):

```
$ jarsigner -keystore NONE -storetype PKCS11 -signedjar demo_signed.jar example.jar JarsignerDemo
```

NOTE: The last field in the jarsigner command above, "JarsignerDemo", needs to match the alias that was specified in the `keytool -importcert` command in the previous section.

The command will prompt for the passphrase of the keystore. Type in the password that was specified for the "JarsignerDemo" keystore in the previous section.

If the signing is successful, the response will include a confirmation message that says, "jar signed.".

NOTE: Please refer to Oracle's documentation regarding other flags that can be used in the jarsigner command above, such as `-tsa` and `-tsacert`.

[12.2] VERIFYING THE SIGNATURE OF A SIGNED JAR FILE

The signed JAR file that was output from the `jarsigner` command in the previous subsection was called `demo_signed.jar`. Now, run the following command to verify the signature of that file.

```
$ jarsigner -verify demo_signed.jar -verbose -certs
```

If the verification is successful, the response will include a confirmation message that says, "jar verified.".

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 does 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 you cannot find 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