JAVA JARSIGNER

Integration Guide

Applicable Devices:

KMES Series 3

TABLE OF CONTENTS

[1] DOCUMENT INFORMATION

[1.1] DOCUMENT OVERVIEW

The purpose of this document is to provide information regarding the configuration of the Futurex KMES Series 3 with Java Jarsigner using PKCS #11 libraries. For additional questions related to your KMES Series 3 device, see the relevant user guide.

[1.2] ABOUT JAVA JARSIGNER

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."

[2] PREREQUISITES

Supported Hardware:

- KMES Series 3, version 6.1.4.x and above, with the *PKCS11* license enabled

Supported Operating Systems:

- Windows 7 and above
- Linux

Other:

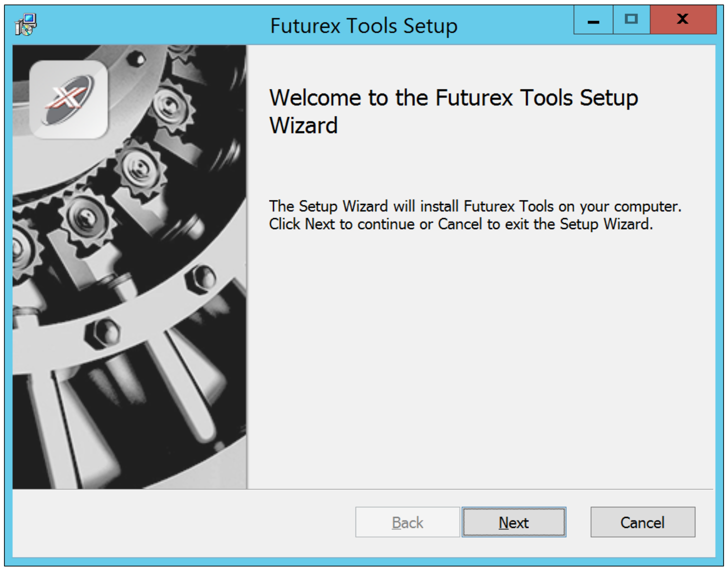
- Java Development Kit (JDK) 8

[3] INSTALL FUTUREX PKCS #11 (FXPKCS11)

In a Windows environment, the easiest way to install the Futurex PKCS #11 module is using **FXTools**. FXTools can be downloaded from the Futurex Portal. In a Linux environment you simply need to download a tarball of the FXPKCS11 binaries from the Futurex Portal, and then extract the tar file locally wherever you want the application to be installed on your system. Step by step installation instructions for both of these scenarios are provided in the following subsections:

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

- Run the FXTools Installer as an administrator



Welcome to the Futurex Tools Setup Wizard

The Setup Wizard will install Futurex Tools on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

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 that are 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 FXPKCS11 MODULE IN LINUX

Extract the tarball file (`fxpkcs11-linux-x.xx-xxxx.tar`) in the desired working directory.

NOTE: To make the Futurex PKCS #11 module accessible system-wide, it needs to be placed into the `/usr/local/bin` directory 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` -> 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

Within both the `x86` and `x64` directories are two directories. One called `OpenSSL-1.0.x` and the other called `OpenSSL-1.1.x`. Both of these OpenSSL directories contain the FXPKCS11 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` -> FXPKCS11 Library File
- `PKCS11Manager` -> Program to test connection and manage the HSM through the FXPKCS11 library

System environment variables need to be defined for the FXPKCS11 library and the FXPKCS11 configuration file. To do so, open the `/etc/profile` file in a text editor and add the following two lines at the bottom:

```
export FXPKCS11_MODULE=/usr/local/bin/fxpkcs11/libfxpkcs11.so
export FXPKCS11_CFG=/usr/local/bin/fxpkcs11/fxpkcs11.cfg
```

NOTE: The locations defined above for the FXPKCS11 library and FXPKCS11 configuration files need to be specific to where they are installed on your system.

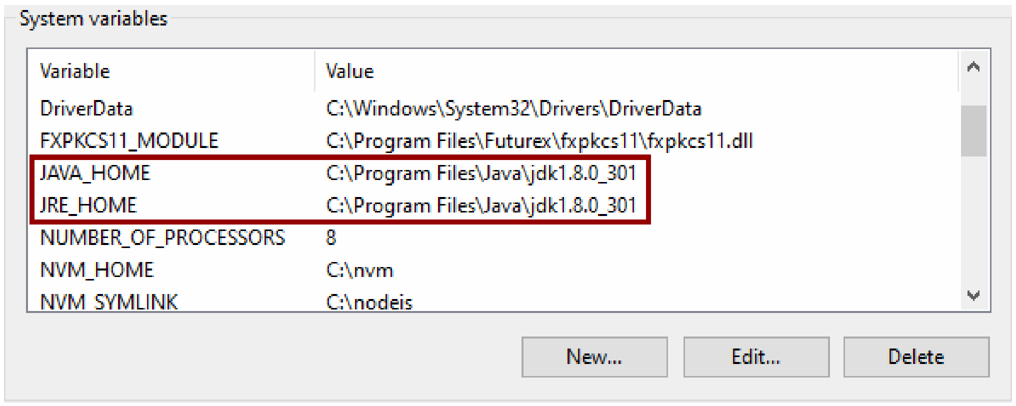
[4] SETTING SYSTEM ENVIRONMENT VARIABLES FOR THE JAVA LIBRARY

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;



Variable	Value
DriverData	C:\Windows\System32\Drivers\DriverData
FXPKCS11_MODULE	C:\Program Files\Futurex\fxpkcs11\fxpkcs11.dll
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_301
JRE_HOME	C:\Program Files\Java\jdk1.8.0_301
NUMBER_OF_PROCESSORS	8
NVM_HOME	C:\nvm
NVM_SYMLINK	C:\nodejs

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
```

[5] 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:

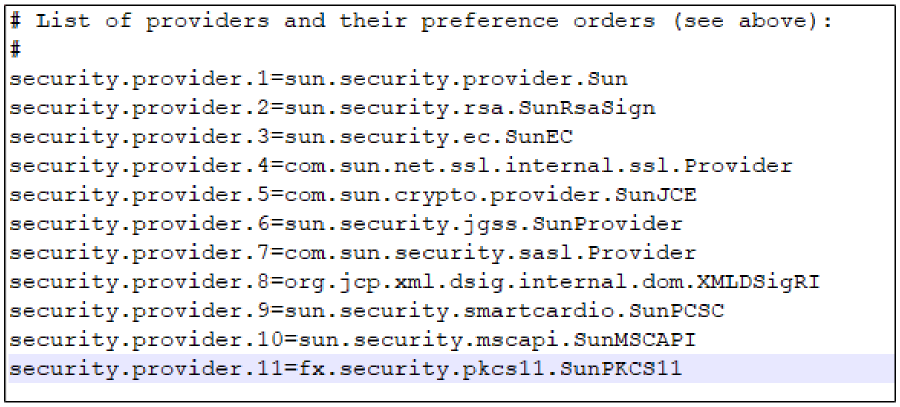
fxjp11.dll (library) -> *C:\Program Files\Java\jdk1.8.0_301\jre\lib\ext*

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

[6] 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

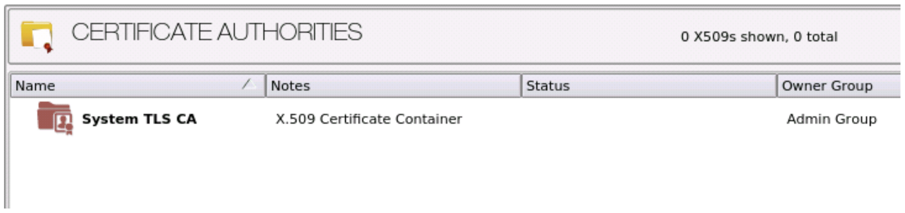
[7] KMES SERIES 3 CONFIGURATION

The first half of this section will cover the steps needed to configure TLS communication between the KMES Series 3 and the computer where Java Jarsigner and the Futurex PKCS #11 library are installed. The second half of this section will cover general KMES configurations that need to be made for the KMES to provide code signing and verification functionality for Java ARchive (JAR) files.

[7.1] CONFIGURE TLS COMMUNICATION BETWEEN THE KMES SERIES 3 AND THE COMPUTER WHERE JARSIGNER AND FXPKCS11 ARE INSTALLED

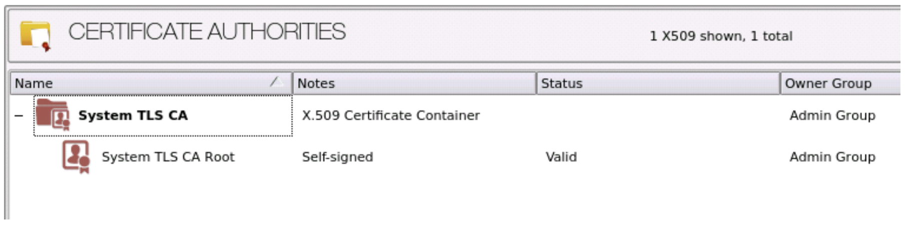
[7.1.1] Create a Certificate Authority (CA)

1. Log in to the KMES Series 3 application interface with the default Admin identities.
2. Select *Certificate Authorities* in the left menu, then click the **Add CA...** button at the bottom of the page.
3. In the *Certificate Authority* dialog, enter a name for the Certificate Container, leave all other fields as the default values, then click **OK**.
4. The Certificate Container that was just created will be listed now in the Certificate Authorities menu.



Name	Notes	Status	Owner Group
System TLS CA	X.509 Certificate Container		Admin Group

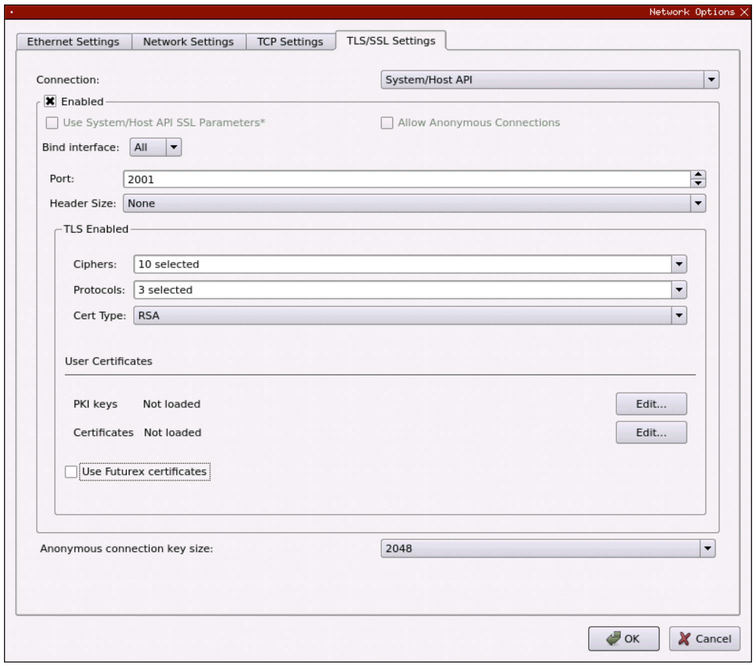
5. Right-click on the Certificate Container and select **Edit...**. In the *Certificate Authority* dialog, check the box that says, "Can be used for PKI authentication", then click **OK** to save.
6. Right-click on the Certificate Container again and select **Add Certificate -> New Certificate...**
7. In the *Subject DN* tab, set a Common Name for the certificate, such as "System TLS CA Root".
8. In the *Basic Info* tab, change the Major key to the **PMK**. All other settings can be left as the default values.
9. In the *V3 Extensions* tab, select the "Example Certificate Authority" profile, then click **OK**.
10. The root CA certificate will be listed now under the previously created Certificate Container.



Name	Notes	Status	Owner Group
System TLS CA	X.509 Certificate Container		Admin Group
System TLS CA Root	Self-signed	Valid	Admin Group

[7.1.2] Generate a CSR for the System/Host API connection pair

1. Go to *Configuration* -> *Network Options*.
2. In the *Network Options* dialog, select the *TLS/SSL Settings* tab.
3. Under the **System/Host API** connection pair, uncheck "Use Futurex certificates", then click **Edit...** next to PKI keys in the User Certificates section.



Network Options

Ethernet Settings | Network Settings | TCP Settings | TLS/SSL Settings

Connection: System/Host API

☒ Enabled

☐ Use System/Host API SSL Parameters*

☐ Allow Anonymous Connections

Bind interface: All

Port: 2001

Header Size: None

TLS Enabled

Ciphers: 10 selected

Protocols: 3 selected

Cert Type: RSA

User Certificates

PKI keys Not loaded Edit...

Certificates Not loaded Edit...

☐ Use Futurex certificates

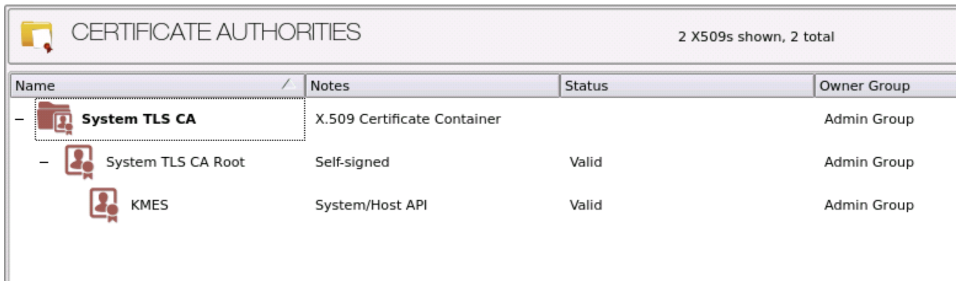
Anonymous connection key size: 2048

OK Cancel

4. In the *Application Public Keys* dialog, click **Generate...**
5. There will be a warning stating that SSL will not be functional until new certificates are imported. Select **Yes** if you wish to continue.
6. In the *PKI Parameters* dialog, change the Encrypting key to the **PMK**, then change the Key Size to **2048** and click OK.
7. It should show that a PKI Key Pair is loaded now in the *Application Public Keys* dialog. If this is the case, click **Request...**
8. In the *Subject DN* tab, set a Common Name for the certificate, such as "KMES".
9. In the *V3 Extensions* tab, select the "Example TLS Server Certificate" profile.
10. In the *PKCS #10 Info* tab, select a save location for the CSR, then click **OK**.
11. There should be a message stating that the certificate signing request was successfully written to the file location that was selected. Click **OK**.
12. Click **OK** again to save the *Application Public Keys* settings.
13. In the main *Network Options* dialog, it should now say "Loaded" next to **PKI keys** for the System/Host API connection pair.

[7.1.3] Sign the System/Host API CSR

1. Go to the *Certificate Authorities* menu.
2. Right-click on the root CA certificate created in section 2.1.1, then select **Add Certificate** -> **From Request...**
3. In the file browser, find and select the CSR that was generated for the System/Host API connection pair.
4. Once loaded, none of the settings need to be modified for the certificate. Click **OK**.
5. The signed System/Host API certificate should now show under the root CA certificate on the *Certificate Authorities* page.



CERTIFICATE AUTHORITIES — 2 X509s shown, 2 total

Name	Notes	Status	Owner Group
System TLS CA	X.509 Certificate Container		Admin Group
System TLS CA Root	Self-signed	Valid	Admin Group
KMES	System/Host API	Valid	Admin Group

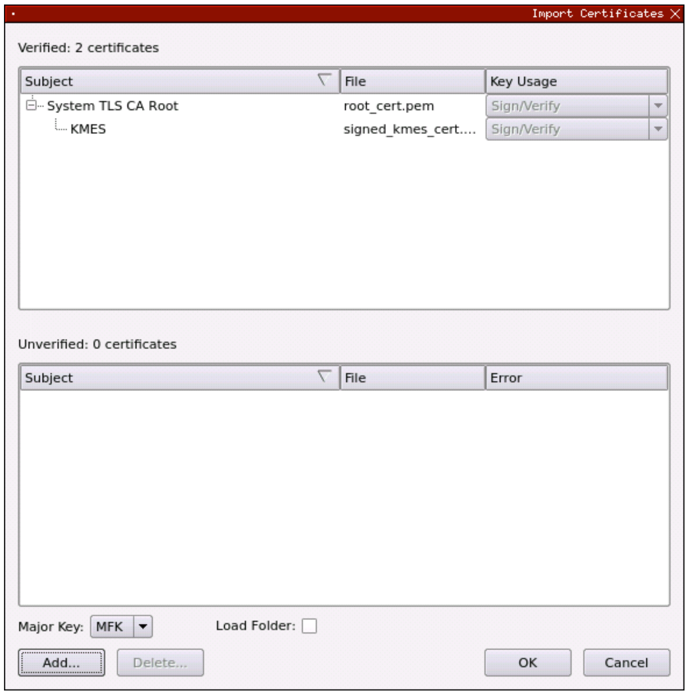
[7.1.4] Export the root CA and signed System/Host API TLS certificates

1. Right-click on the root CA certificate, then select **Export** -> **Certificate(s)...**
2. Change the encoding to **PEM**. Then click **Browse...** and select a save location, as well as a name for the export file.
3. There should be a message stating that the file was successfully written to the location that was selected. Click **OK**.
4. Right-click on the signed System/Host API certificate, then select **Export** -> **Certificate(s)...**
5. Change the encoding to **PEM**. Then click **Browse...** and select a save location, as well as a name for the export file.
6. There should be a message stating that the file was successfully written to the location that was selected. Click **OK**.

[7.1.5] Load the exported TLS certificates into the System/Host API connection pair

1. Go to *Configuration -> Network Options*.
2. In the *Network Options* dialog, select the *TLS/SSL Settings* tab.
3. Click **Edit...** next to Certificates in the User Certificates section.
4. Right-click on the **System/Host API SSL CA** X.509 Certificate Container, then select **Import...**
5. Click **Add...** at the bottom of the *Import Certificates* dialog.

6. In the file browser, find and select both the root CA certificate and the signed System/Host API certificate, then click **Open**. The certificate chain should appear as shown below:



Verified: 2 certificates

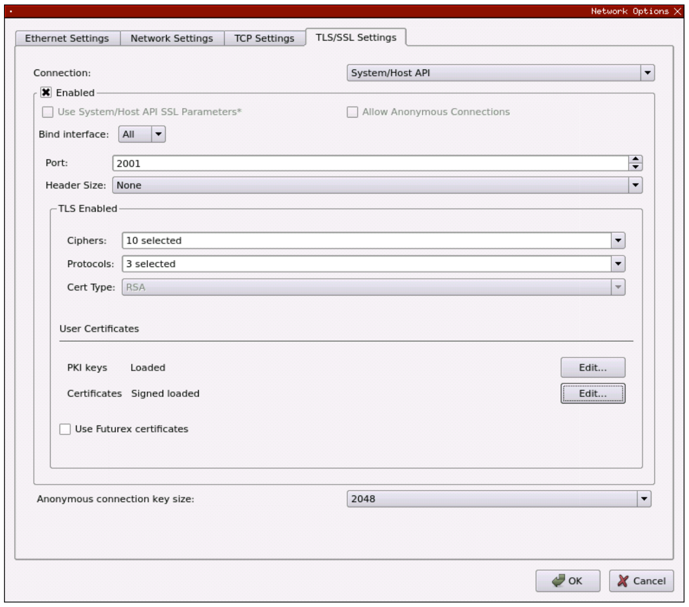
Subject	File	Key Usage
System TLS CA Root	root_cert.pem	Sign/Verify
KMES	signed_kmes_cert....	Sign/Verify

Unverified: 0 certificates

Subject	File	Error

Major Key: MFK
Load Folder: ☐

7. Click **OK** to save the changes. In the *Network Options* dialog, the System/Host API connection pair should show "Signed loaded" next to Certificates in the User Certificates section, as shown below:



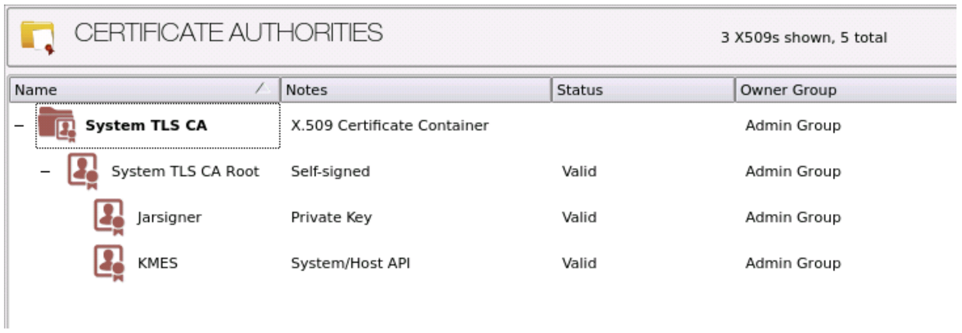
Connection: System/Host API
☒ Enabled
☐ Use System/Host API SSL Parameters*
☐ Allow Anonymous Connections
Bind interface: All
Port: 2001
Header Size: None
TLS Enabled
Ciphers: 10 selected
Protocols: 3 selected
Cert Type: RSA
User Certificates
PKI keys Loaded
Certificates Signed loaded
☐ Use Futurex certificates
Anonymous connection key size: 2048

8. Click **OK** to save and exit the Network Options dialog.

[7.1.6] Generate a signed client TLS certificate for Jarsigner/FXPKCS11

1. Go to the *Certificate Authorities* menu.
2. Right-click on the root CA certificate and select **Add Certificate** -> **New Certificate...**

3. In the *Subject DN* tab, set a Common Name for the certificate, such as "Jarsigner".
4. All settings in the *Basic Info* tab can be left as the default values.
5. In the *V3 Extensions* tab, select the "Example TLS Client Certificate" profile, then click **OK**.
6. The signed Jarsigner certificate will be listed now under the root CA certificate.



Name	Notes	Status	Owner Group
System TLS CA	X.509 Certificate Container		Admin Group
System TLS CA Root	Self-signed	Valid	Admin Group
Jarsigner	Private Key	Valid	Admin Group
KMES	System/Host API	Valid	Admin Group

[7.1.7] Allow export of certificates using passwords

1. Navigate to *Configuration* -> *Options*.
2. Check the box next to the first menu option, which says, "Allow export of certificates using passwords".
3. Click **Save**.

[7.1.8] Export the signed Jarsigner client TLS certificate as a PKCS #12 file

1. Go to the *Certificate Authorities* menu.
2. Right-click on the signed Jarsigner certificate, then select **Export** -> **PKCS12...**
3. In the *Export PKCS12* dialog, select **Export Selected** under Export Options, then click **Next >**.
4. Specify a password for the PKCS #12 file, then click **Next >**.
5. Click **Finish**.

NOTE: The `export_pkcs12.p12` file will be saved in the root directory of either the USB or SFTP mount point, depending on which is configured. This PKCS #12 file needs to be moved to the computer where jarsigner is installed. In a later section, it will be configured in the Futurex PKCS #11 configuration file and used for TLS communication with the KMES Series 3.

[7.2] GENERAL KMES CONFIGURATIONS FOR COMMUNICATION BETWEEN JARSIGNER/FXPKCS11 AND THE KMES SERIES 3

[7.2.1] Enable the required Host API commands

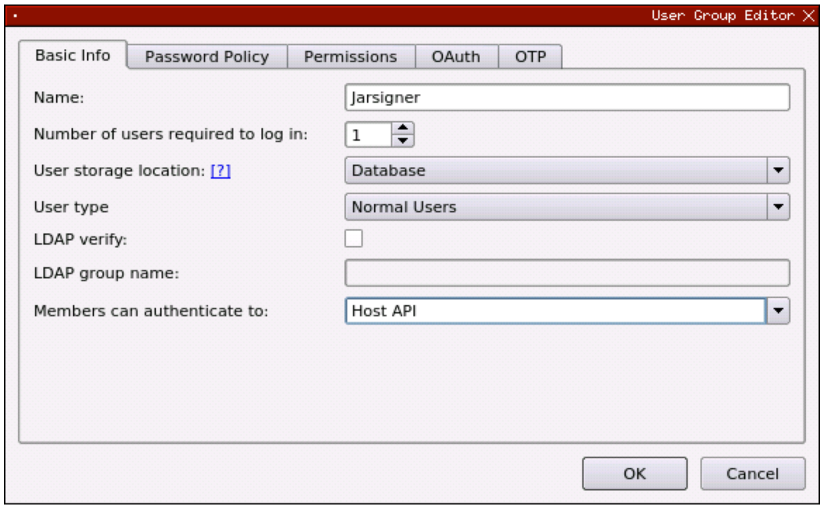
1. Go to *Configuration* -> *Host API Options*.
2. Enable the following commands:

- ECHO - Communication Test/Retrieve Version
- RAFA - Enumerate issuance policies
- RAGA - Retrieve issuance policy details
- RAGO - Retrieve Request (Hash Signing)
- RAUO - Upload Request (Hash Signing)
- RKCP - Get Command Permissions
- RKLN - Lookup Objects
- RKLO - Login User
- RKRK - Retrieve Generated Keys
- TIME - Set Time

3. Click **Save**.

[7.2.2] Create a Jarsigner User Group with the required permissions

1. Select *Users* in the left menu, then click the **Add Group...** button at the bottom of the page.
2. Specify a name for the group, such as "Jarsigner", then ensure that the settings below are selected.

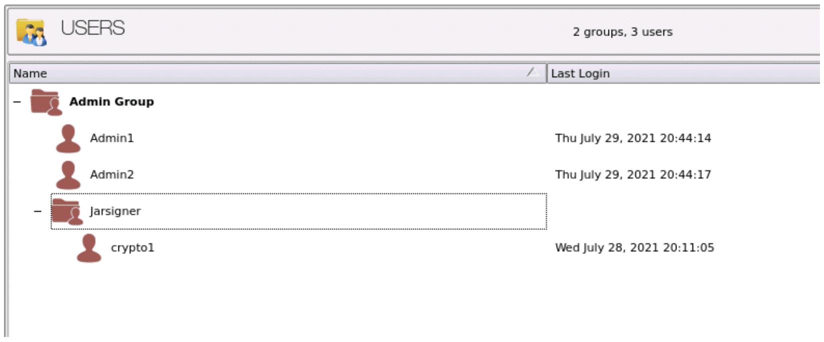


Name: Jarsigner
Number of users required to log in: 1
User storage location: [?] Database
User type Normal Users
LDAP verify: ☐
LDAP group name:
Members can authenticate to: Host API

3. In the *Permissions* tab, ensure that only the following permissions are selected:
   - Manage certificates -> Export
   - Manage certificates -> Upload
   - Manage keys (top-level permission only)
4. Click **OK** to save.
5. Disregard the warning that there are only 0 users currently in the group. Click **OK**.

[7.2.3] Create a single login user within the Jarsigner User Group

1. On the *Users* page, right-click on the **Jarsigner** User Group and select **Add** -> **User...**
2. In the *Basic Info* tab, set a user name and password for the user.
3. Click **OK**, and you should see the new user listed under the **Jarsigner** User Group, as shown below:



USERS 2 groups, 3 users

Name	Last Login
Admin Group	
Admin1	Thu July 29, 2021 20:44:14
Admin2	Thu July 29, 2021 20:44:17
Jarsigner	
crypto1	Wed July 28, 2021 20:11:05

[7.2.4] Create a Signing Approval Group and give the Jarsigner User Group permissions to use it

1. Select *Signing Approval* in the left menu, then click the **Add Approval Group...** button at the bottom of the page.
2. Set a name for the Approval Group, such as "Jarsigner", then click **OK** to save.
3. Right-click on the **Jarsigner** Approval Group, then select **Permission...**
4. Give the **Jarsigner** User Group the **Use** permission, then click **OK**.

[7.2.5] Create a Code Signing certificate

This subsection will describe two different methods that can be used to issue a code signing certificate.

Issued by a CA on the KMES

1. Go to the *Certificate Authorities* menu and click the **Add CA...** button at the bottom of the page.
2. In the *Certificate Authority* dialog, enter a name for the Certificate Container, such as "Jarsigner". Set the owner field to the group that contains the user created in section 7.2.3, then click **OK**. The new Certificate Container will be listed now in the Certificate Authorities menu.
3. Right-click on the **Jarsigner** Certificate Container and select **Add Certificate -> New Certificate...**
4. In the *Subject DN* tab, set a Common Name for the certificate, such as "Root".
5. In the *Basic Info* tab, change the Major key to the **PMK**. All other settings can be left as the default values.
6. In the *V3 Extensions* tab, select the "Example Certificate Authority" profile, then click **OK**. The **Root** CA certificate will be listed now under the "Jarsigner" Certificate Container.
7. Right-click on the **Root** CA certificate that was just created and select **Add Certificate** -> **New Certificate...**.

8. In the *Subject DN* tab, set a Common Name for the certificate, such as "Code Signing".
9. Move straight to the *V3 Extensions* tab, select the "Example Code Signing Certificate" profile, and then click **OK**. The **Code Signing** certificate will be listed now under the **Root** CA certificate inside of the **Jarsigner** Certificate Container.

Issued by an external CA

For this method, the external CA certificate(s) need to be imported into an empty Certificate Container on the KMES. A Certificate Signing Request (CSR) will then be generated, which the external CA will use to issue a code signing certificate. The code signing certificate will then be imported into the Certificate Container on the KMES that contains the external CA certificate.

1. Go to the *Certificate Authorities* menu and click the **Add CA...** button at the bottom of the page.
2. In the *Certificate Authority* dialog, enter a name for the Certificate Container, such as "Jarsigner". Set the owner field to the group that contains the user created in section 7.2.3, then click **OK**. The new Certificate Container will be listed now in the Certificate Authorities menu.
3. Right-click on the **Jarsigner** Certificate Container and select **Import** -> **Certificate(s)...**. This will open the *Import Certificates* dialog.
4. Click the **Add...** button in the bottom left-hand portion of the dialog, then find and select the external CA certificate(s) that will be issuing the code signing certificate in the file browser. The CA certificate(s) will populate in the Verified section of the *Import Certificates* dialog.
5. Click **OK** to save. The external CA certificate(s) should be listed now in tree form under the **Jarsigner** Certificate Container.
6. Next, we'll create a placeholder code signing certificate, from which a CSR can be generated. Right-click on the lowest level CA certificate in the tree and select **Add Certificate** -> **Pending...**. This will open the *Create X.509 Certificate* dialog.
7. In the *Subject DN* tab, set a Common Name for the certificate, such as "Code Signing".
8. In the *V3 Extensions* tab, select the "Example Code Signing Certificate" profile.
9. Click **OK**. The **Code Signing** placeholder certificate will be listed now under the external CA certificate(s).
10. Right-click on placeholder **Code Signing** certificate and select **Export** -> **Signing Request...**. This will open the *Create PKCS #10 Request* dialog.
11. Leave all of the settings in the *Subject DN* tab as the default values.
12. In the *V3 Extensions* tab, select the "Example Code Signing Certificate" profile.
13. In the *PKCS #10 Info* tab, specify a save location for the CSR, then click **OK**. There should be a message stating that the certificate signing request was successfully written to the location you specified.
14. The CSR file then needs to be taken to an external certificate authority. Using the CSR, the external CA will issue a code signing certificate.

NOTE: After the external CA issues the code signing certificate, it needs to be copied to the storage medium that is configured on the KMES.

15. In the *Certificate Authorities* menu on the KMES, right-click on the placeholder **Code Signing** certificate and select **Replace** -> **With Signed Certificate...**. This will open the *Import Certificates* dialog.
16. Click the **Add...** button in the bottom left-hand portion of the dialog, then find and select the externally signed code signing certificate in the file browser. The code signing certificate will populate under the CA certificate(s) in the Verified section of the *Import Certificates* dialog.
17. Click **OK** to save.

[7.2.6] Apply an Issuance Policy to the Jarsigner code signing certificate

1. Go to the *Certificate Authorities* menu.
2. Right-click on the **Code Signing** certificate and select **Issuance Policy** -> **Add...**
3. Under the *Basic Info* tab:
   - Set Approvals to **0** to allow anonymous signing.
   - Select any hashes that you wish to allow.

   NOTE: Specifying an Alias is not required.
4. In the *X.509* tab, set the Default approval group to **Jarsigner**.
5. In the *Object Signing* tab, select the **Allow object signing** checkbox.
6. Click **OK** to apply the Issuance Policy to the Jarsigner code signing certificate.

[7.2.7] Create a Jarsigner Application Key Group

1. Select *Application Keys* in the left menu, then click the **Add Application Key Group...** button at the bottom of the page. This will open the *Application Key Group* dialog.
2. Select **Futurex Data Protection** in the Service dropdown.
3. Specify a name for the Application Key Group, such as "Jarsigner Key Group".
4. In the Key Length dropdown, select **2048 bits**.
5. Click **OK** to finish creating the Jarsigner Application Key Group.

[8] EDIT THE FUTUREX PKCS #11 CONFIGURATION FILE

[8.1] DEFINE CONNECTION INFORMATION

The *fxpkcs11.cfg* file allows the user to set the FXPKCS11 library to connect to the KMES Series 3. 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 **<KMS>** 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).

```
<KMS>
    # Which PKCS11 slot
    <SLOT>                  0                       </SLOT>

    # Login username
    <CRYPTO-OPR>            crypto1                 </CRYPTO-OPR>

    # Connection information
    <ADDRESS>               10.0.8.30 </ADDRESS>
    <PROD-PORT>             2001                    </PROD-PORT>
    <PROD-TLS-ENABLED>      YES                     </PROD-TLS-ENABLED>
    <PROD-TLS-ANONYMOUS>    NO                      </PROD-TLS-ANONYMOUS>
    <PROD-TLS-CA>           /home/bbarrows/tls/root.pem        </PROD-TLS-CA>
    <PROD-TLS-KEY>          /home/bbarrows/tls/signed_jarsigner_cert.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>
</KMS>
```

The **<SLOT>** field can be left as the default value of 0.

In the **<CRYPTO-OPR>** field, specify the name of user that was created on the KMES in section 7.2.3.

In the **<ADDRESS>** field, specify the IP of the KMES that the PKCS #11 library should connect to.

In the **<PROD-PORT>** field, set the PKCS #11 library to connect to the default Host API port on the KMES, port 2001.

The **<PROD-TLS-ENABLED>** field needs to be set to "YES" because the only way to connect to the Host API port on the KMES is over TLS.

The **<PROD-TLS-ANONYMOUS>** field defines whether the PKCS #11 library will be authenticating to the KMES or not. Since we're connecting to the Host API port, this value must be set to "NO".

The location of the CA certificate/s needs to be defined with one or more instances of the **<PROD-TLS-CA>** tag. In this example, there is only one CA certificate.

The **<PROD-TLS-KEY>** tag 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. For this integration, in the **<PROD-TLS-KEY>** tag, we're specifying the location of the signed Jarsigner certificate that was exported from the KMES as PKCS #12 file in section 7.1.8. In the **<PROD-TLS-KEY-PASS>** tag, we're specifying the password of that PKCS #12 file.

If a Guardian is being used to manage KMES Series 3 devices 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”.

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

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

[9] 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.

[9.1] 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 KMES 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 3 entries

Jarsigner:Code Signing:C, PrivateKeyEntry,

Certificate fingerprint (SHA-256):
A6:9B:22:A7:0B:D0:59:57:80:EE:AD:61:9D:FC:41:9F:CE:5D:32:9B:32:E0:3C:D3:86:56:DC:24:02:CE:8E:B1

Jarsigner:Code Signing:PK, SecretKeyEntry,

Jarsigner:Root:C, trustedCertEntry,

Certificate fingerprint (SHA-256):
5F:D9:42:15:80:31:CF:07:17:7C:28:CC:F8:A8:CD:6A:11:ED:B5:93:9F:7B:D4:1B:B7:DE:AB:D0:28:53:E9:A9
```

Now that we've confirmed the keys needed for code signing are accessible, run the following command to sign a JAR file using the KMES-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 "Jarsigner:Code
Signing:C"
```

NOTE: The last field in the jarsigner command above, "Jarsigner:Code Signing:C" (i.e., "Certificate Container Name:Code Signing Certificate Name:C"), is the format required for the FXPKCS11 library to be able to find the keys needed for code signing on the KMES. If the certificate container and code signing certificate were named differently than outlined in section 7.2.5, modify the above jarsigner command accordingly. The "C" value at the end of the last field needs to be included, as shown, due to required formatting.

The command will prompt for the passphrase of the keystore. Type in the password of the user that is configured in the FXPKCS11 configuration file, then click Enter.

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`.

[9.2] 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