PKCS#11 Wrapper – Jarsigner Signing Integration Document

Jarsigner is a command-line tool within the Java Development Kit (JDK) used for digitally signing Java Archive (JAR) files. Because it ships with the JDK, no additional signing tool needs to be installed.

Jarsigner reads its signing key from a keystore. By giving it a keystore type of PKCS11 together with Encryption Consulting's PKCS#11 wrapper configuration, that keystore becomes your HSM — the keystore itself is NONE, because there is no file on disk. The private key never leaves the HSM and every signing operation is recorded centrally.

Section 1 covers the steps common to every platform. After that, follow only the track for your operating system — Sections 2 and 3 for Linux (Ubuntu), Sections 4 and 5 for macOS, or Sections 6 and 7 for Windows. Each track ends with signing and verification.

Prerequisites: Access to the CodeSign Secure portal, administrative rights on the client machine, and a JAR file to sign.

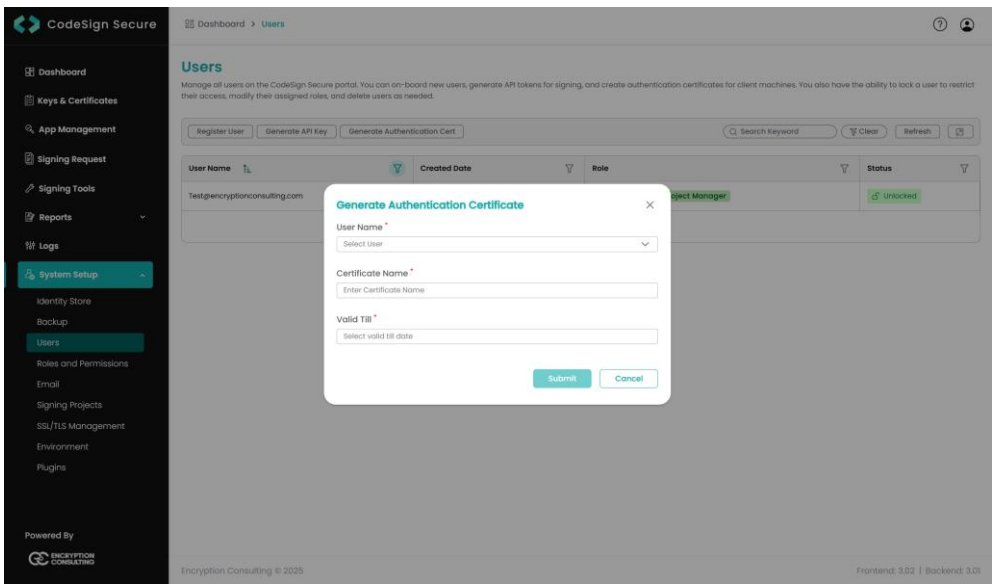
1. Common Setup

Every platform needs the same two things before Jarsigner will work: an authentication certificate that identifies your machine to CodeSign Secure, and the two configuration files that ship inside the PKCS#11 Wrapper download.

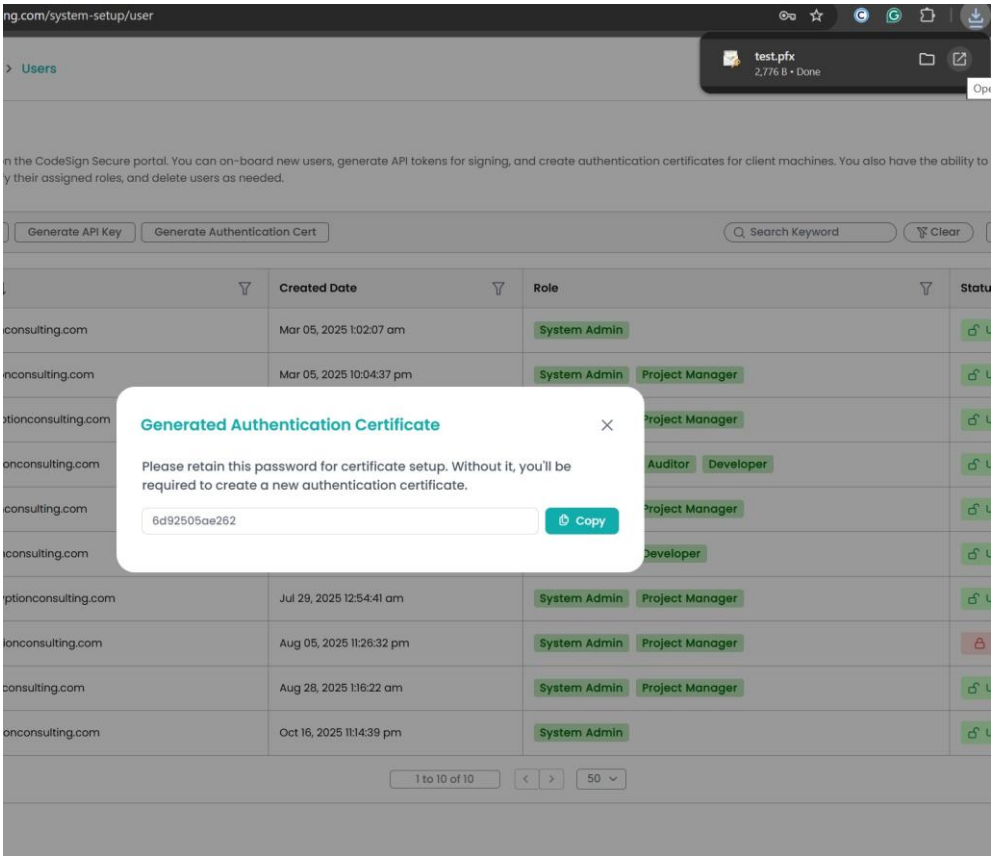
Steps:

Step 1: Generate a P12 Authentication Certificate

- Generate a P12 Authentication certificate from the System Setup > User > Generate Authentication Certificate dropdown in the CodeSign Secure portal.



- Enter the certificate name, user name, and expiry date. A certificate is generated and downloaded, and its password is displayed.



NOTE: The password is displayed only once, so copy and store it safely. Both the certificate path and its password are entered into ec_pkcs11client.ini in your platform track.

Step 2: Understand the Two Configuration Files

- The PKCS#11 Wrapper download contains two files that must be edited before use. The same fields apply on every platform.
  - **pkcs11properties.cfg –** Describes the PKCS#11 token. Update it with the correct library path for your platform. This is also the file passed to Jarsigner as -providerArg.
  - **ec_pkcs11client.ini –** Points the wrapper at your CodeSign Secure server and at the authentication certificate from Step 1. The fields to update are listed below.

Field	Description
`name`	The path location of the Log xml file you downloaded with the PKCS11 Wrapper tool (EC_PKCS11_CLIENT-LogConfig.xml).
`url`	Verify the server url of your CodeSign Secure portal.
`username`	The username of your CodeSign Secure account.

Field	Description
`passcode`	The secret passcode which was used to set up the CodeSign Secure portal.
`path`	The path location of your SSL Authentication certificate.
`passwd`	The password of your SSL Authentication certificate.

NOTE: When using Luna HSM, the slot number must also be added to the pkcs11properties.cfg file. Each platform track below shows this variation.

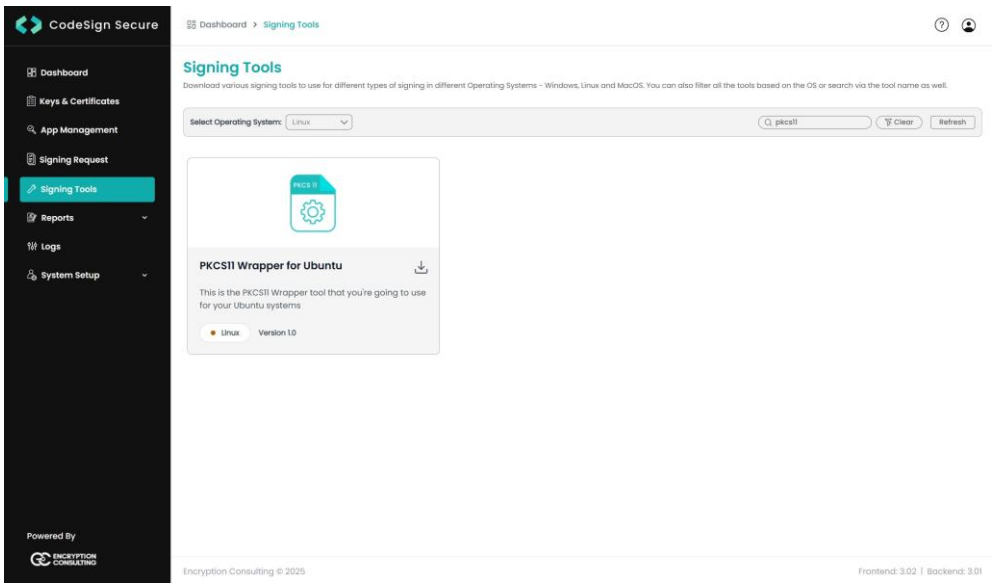
2. Linux (Ubuntu): Install and Configure

To perform signing using the Jarsigner Tool and our PKCS11 Wrapper in a Linux (Ubuntu) machine, you would need to follow the below steps.

Steps:

Step 1: Download the PKCS#11 Wrapper for Ubuntu

- Go to EC CodeSign Secure v3.02's Signing Tools section and download the PKCS11 Wrapper for Ubuntu.



Step 2: Edit the Configuration Files

- Go to your Ubuntu client system and edit the configuration files (ec_pkcs11client.ini and pkcs11properties.cfg) downloaded in the PKCS11 Wrapper.

```
aryan@test:~/PKCS_Wrapper_Ubuntu$ cat ec_pkcs11client.ini
[logconfig_file]
name=/home/aryan/PKCS_Wrapper_Ubuntu/EC_PKCS11_CLIENT-LogConfig.xml

[server_url]
url=https://devcodesignsecure.encryptionconsulting.com

[pfxfile_path]
path=/home/aryan/PKCS_Wrapper_Ubuntu/Pkcs11Test.pfx

[pfxfile_passwd]
passwd=de48b6924cb8

[login_token]
username=Aryan@encryptionconsulting.com
passcode=8@!$U3QXS5Xtu2vf
aryan@test:~/PKCS_Wrapper_Ubuntu$ cat pkcs11properties.cfg
name=signingmanager
library=/home/aryan/PKCS_Wrapper_Ubuntu/ec_pkcs11client.so
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
aryan@test:~/PKCS_Wrapper_Ubuntu$
```

NOTE: When using Luna HSM, add the slot number in the pkcs11properties.cfg file as shown below.

```
aryan@test:~/PKCS_Wrapper_Ubuntu$ cat pkcs11properties.cfg
name=signingmanager
library=/home/aryan/PKCS_Wrapper_Ubuntu/ec_pkcs11client.so
slot=3
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
```

Step 3: Set the EC_INI_FILE_PATH Environment Variable

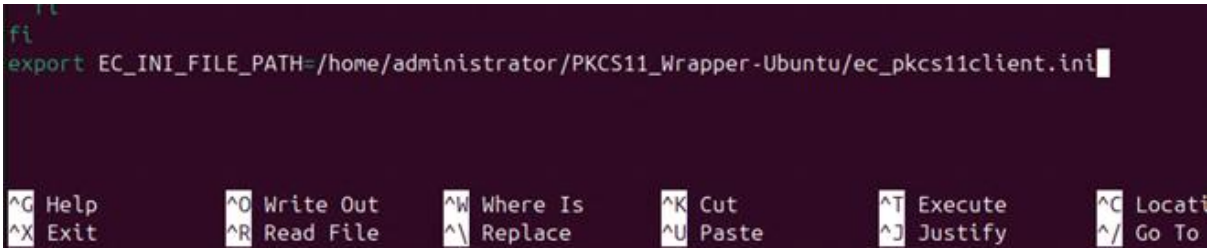
- Now we need to add the environment variable for the pkcs11 client library.
- Run the below command to open the bashrc file:

```
nano ~/.bashrc
```

```
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$ nano ~/.bashrc
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$
```

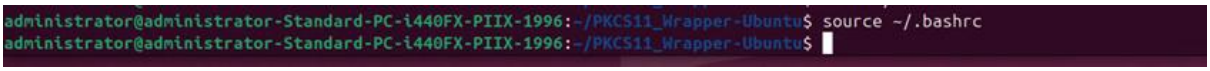
- Now add the EC_INI_FILE_PATH variable with the path of the .ini at the end of the bashrc file, like:

```
export EC_INI_FILE_PATH=\
/home/administrator/PKCS11_Wrapper-Ubuntu/ec_pkcs11client.ini
```



- Press Ctrl+X, then enter Y, and then press Enter to save.
- Reload the environment variables:

```
source ~/.bashrc
```



- Check whether the variable has been set correctly:

```
echo $EC_INI_FILE_PATH
```

NOTE: If you cannot see the file location from the echo command, open a new terminal and try again.

Step 4: Install and Activate Java

- You will also need to install Java (Java 8 to 17) on your Ubuntu machine for Jarsigner to work with our PKCS11 Wrapper.
- Run the following command to install Java 17 on your Ubuntu machine:

```
sudo apt install openjdk-17-jdk
```

```
aryan@ Sign-new:~$ sudo apt install openjdk-17-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  libllvm17t64
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-dejavu-extra libatk-wrapper-java libatk-wrapper-java-jni libice-dev
  libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev
  openjdk-17-jdk-headless openjdk-17-jre openjdk-17-jre-headless x11proto-dev
  xorg-sgml-doctools xtrans-dev
```

- To use Java 17, you will need to set it as the active version.

```
sudo update-alternatives --config java
```

```
aryan@ Sign-new:~$ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority   Status
------------------------------------------------------------
* 0            /usr/lib/jvm/java-21-openjdk-amd64/bin/java   2111      auto mode
  1            /usr/lib/jvm/java-17-openjdk-amd64/bin/java   1711      manual mode
  2            /usr/lib/jvm/java-21-openjdk-amd64/bin/java   2111      manual mode

Press <enter> to keep the current choice[*], or type selection number: 1
update-alternatives: using /usr/lib/jvm/java-17-openjdk-amd64/bin/java to provide /usr/bin/java
 (java) in manual mode
aryan@JSign-new:~$ S
```

- Now to check whether Java has been installed properly or not, run:

```
java -version
```

```
aryan@ Sign-new:~$ java -version
openjdk version "17.0.13" 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
aryan@ Sign-new:~$
```

Step 5: Set the JAVA_HOME Environment Variable

- Open the bashrc file again:

```
nano ~/.bashrc
```

- After running the above command, add these lines at the end of the file:

```
export JAVA_HOME=<Path of the Java 17 home folder>
export PATH=$JAVA_HOME/bin:$PATH
```

```
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
```

NOTE: JAVA_HOME must be the JDK home directory, not its bin subfolder, because the following line appends /bin to it. Pointing JAVA_HOME at bin would produce a bin/bin path and Java would not be found. A typical value is /usr/lib/jvm/java-17-openjdk-amd64.

- Press Ctrl+X, then Y to confirm, and then Enter to save.
- Reload the bashrc file using the below command:

```
source ~/.bashrc
```

- Check if the variable has been set:

```
echo $JAVA_HOME
```

- If not, then open a new terminal and try again.

Step 6: Install the Remaining Packages

- You would also need to install some other pre-requisite packages to perform signing using Jarsigner.

```
sudo apt-get install curl
sudo apt-get install liblog4cxx-dev
sudo apt-get install liblog4cxx12
```

NOTE: If the “sudo apt-get install liblog4cxx12” command gives an error, install the package directly for your architecture using the commands below.

- For amd architecture:

```
wget http://archive.ubuntu.com/ubuntu/pool/universe/l/log4cxx/\
liblog4cxx12_0.12.1-4_amd64.deb
sudo dpkg -i liblog4cxx12_0.12.1-4_amd64.deb
```

- For arm architecture:

```
wget http://ports.ubuntu.com/pool/universe/l/log4cxx/\
liblog4cxx12_0.12.1-4_arm64.deb
sudo dpkg -i liblog4cxx12_0.12.1-4_arm64.deb
```

- We are now ready with our Ubuntu environment to perform the signing using Jarsigner tool and PKCS11 Wrapper.

3. Linux (Ubuntu): Sign and Verify

The signing command must be run from the directory containing your configuration, so that the relative paths in the command resolve correctly.

Steps:

Step 1: Change to the Working Directory

- For signing, change the working directory of the terminal to that folder which contains your “ec_pkcs11client.ini” file.

```
aryan@test:~/PKCS_Wrapper_Ubuntu$ cd ..
aryan@test:~$ cd PKCS_Wrapper_Ubuntu/
aryan@test:~/PKCS_Wrapper_Ubuntu$ ls
ec_pkcs11client.ini  ec_pkcs11_client.log  EC_PKCS11_CLIENT-LogConfig.xml  ec_pkcs11client.so  Pkcs11Test_luna.pfx  Pkcs11Test.pfx
aryan@test:~/PKCS_Wrapper_Ubuntu$
```

Step 2: Run the Signing Command

- Now, run the signing command from this directory.

```
<Path of Jarsigner tool> \
  -keystore NONE \
  -storepass NONE \
  -storetype PKCS11 \
  -sigalg SHA256withRSA \
  -providerClass sun.security.pkcs11.SunPKCS11 \
  -providerArg <Path of pkcs11properties.cfg> \
  -signedjar <Path of the file after signing> \
  <Path of the file to be signed> \
  <Key alias of the signing certificate> \
  -tsa http://timestamp.digicert.com
```

- A sample command is provided below:

```
jarsigner \
  -keystore NONE \
  -storepass NONE \
  -storetype PKCS11 \
  -sigalg SHA256withRSA \
  -providerClass sun.security.pkcs11.SunPKCS11 \
  -providerArg pkcs11properties.cfg \
  -signedjar helloworld_signed.jar \
  helloworld.jar gpg2 \
  -tsa http://timestamp.digicert.com
```

NOTE: The JAR file and the key alias are positional arguments, given in that order and without option names. Keep them together and in that sequence.

Step 3: Verify the Signature

- For verification, run the following command:

```
<Path of Jarsigner tool> -verify \
  <Path of the file after signing> \
  -certs -verbose
```

- A sample command is provided below:

```
jarsigner -verify helloworld_signed.jar -certs -verbose
```

Step 4: Option Reference

- The following are the options and their meaning in the commands:

Option	Description
-keystore NONE	The keystore. This is NONE because the keys live in the HSM rather than in a keystore file.
-storepass NONE	The keystore password. This is NONE because access is controlled by the wrapper configuration.
-storetype PKCS11	The keystore type, which must be PKCS11.
-sigalg	The signature algorithm to use when signing, for example SHA256withRSA.
-providerClass	The Java security provider to use. This is always sun.security.pkcs11.SunPKCS11, the JDK's built-in PKCS#11 provider.
-providerArg	The path of the pkcs11properties.cfg file, which tells the provider how to reach the PKCS#11 Wrapper.
-signedjar	The path of the signed JAR file to produce.
<jar file>	The JAR file to be signed, given as a positional argument with no option name.
<alias>	The key alias of the signing certificate, given as a positional argument immediately after the JAR file.
-tsa	The URL of the timestamping authority, for example http://timestamp.digicert.com.

Option	Description
`-verify`	Checks the signature on an already signed JAR.
`-certs -verbose`	Shows the signer certificates and detailed output when verifying.

Step 5: Cross-Check in CodeSign Secure

- Open the CodeSign Secure portal and navigate to Reports > Signing Request Report, where every signing request performed through the wrapper is recorded for audit purposes.

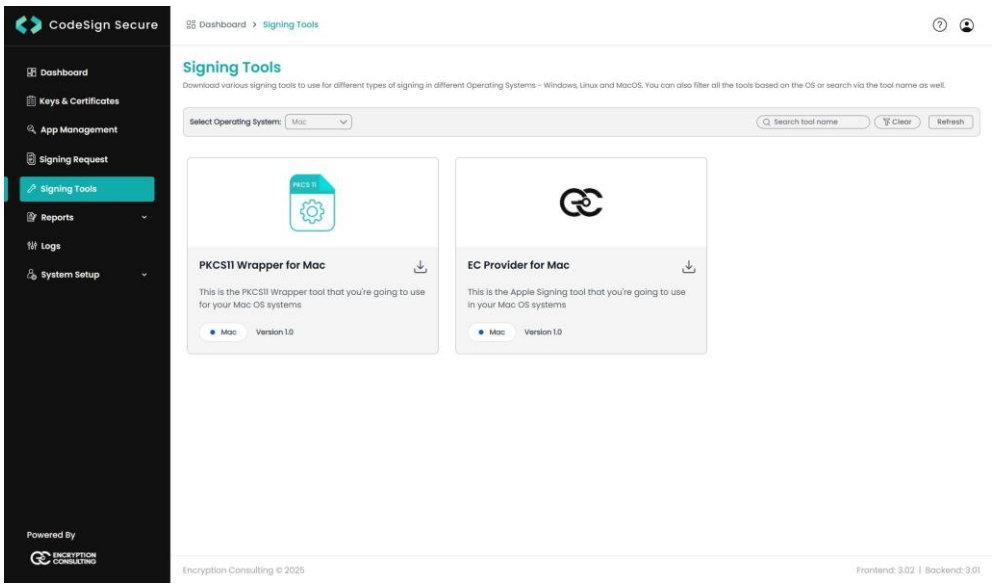
4. macOS: Install and Configure

To perform signing using Jarsigner and our PKCS11 Wrapper in a MacOS machine, you would need to follow the below steps.

Steps:

Step 1: Download the PKCS#11 Wrapper

- Download our PKCS11 Wrapper Tool from the CodeSign Secure portal in the Signing Tools section in your MacOS.



Step 2: Update the Configuration Files

- Go to your PKCS11 Wrapper directory (../PKCS11_Wrapper-Mac) and update the “pkcs11properties.cfg” and “ec_pkcs11client.ini” with the required details.
- For “pkcs11properties.cfg”, update the library path.

NOTE: When using Luna HSM, add the slot number in the pkcs11properties.cfg file as shown below.

```
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac % cat pkcs11properties.cfg
name=signingmanager
library=/Users/subhayuroy/PKCS11_Wrapper-Mac/ec_pkcs11client.dylib
slot=3
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac %
```

- For the "ec_pkcs11client.ini" file, update the six fields listed in Section 1, Step 2.
- The initialization and configuration files should look like this with all the correct file paths and settings.

```
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac % cat ec_pkcs11client.ini
[logconfig_file]
name=/Users/subhayuroy/PKCS11_Wrapper-Mac/EC_PKCS11_CLIENT-LogConfig.xml

[server_url]
;url=https://lunacodesignsecure.encryptionconsulting.com
url=https://devcodesignsecure.encryptionconsulting.com
;url=http://127.0.0.1:6666

[pfxfile_path]
path=/Users/subhayuroy/PKCS11_Wrapper-Mac/AuthCert-nCipher.pfx

[pfxfile_passwd]
passwd=f4550251e262

[login_token]
username=Sam@encryptionconsulting.com
passcode=8@!$U3QXS5Xtu2vf
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac % cat pkcs11properties.cfg
name=signingmanager
library=/Users/subhayuroy/PKCS11_Wrapper-Mac/ec_pkcs11client.dylib
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac %
```

Step 3: Install and Activate Java

- You will also need to install Java (Java 8 to 17) on your MacOS machine for Jarsigner to work with our PKCS11 Wrapper.
- To install Java 17 on your MacOS machine, run the following command:

```
brew install openjdk@17
```

- Find the location where Java 17 is installed on your machine:

```
brew info openjdk@17
```

- Now set Java 17 as the active version. For Zsh:

```
nano ~/.zshrc
```

- For Bash:

```
nano ~/.bash_profile
```

- After running the above command, add these lines:

```
export PATH=<Path of Java 17 bin folder>:$PATH
export JAVA_HOME=<Path of Java 17 bin folder>
```

NOTE: Here PATH is set directly to the bin folder rather than being derived from JAVA_HOME, which is why this track differs from the Linux one in Section 2. If other Java tooling on the machine relies on JAVA_HOME, point it at the JDK home directory instead.

- Reload the environment variables using the below commands. For Zsh:

```
source ~/.zshrc
```

- For Bash:

```
source ~/.bash_profile
```

Step 4: Install the Remaining Packages

- You would also need to install some other pre-requisite packages to perform signing using Jarsigner.

```
brew install log4cxx
brew install curl
```

- We are now ready with our MacOS environment to perform the signing using Jarsigner tool and PKCS11 Wrapper.

5. ## macOS: Sign and Verify

As on Linux, the signing command is run from the directory containing your configuration files.

Steps:

Step 1: Change to the Working Directory

- For signing, change the working directory of the terminal to that folder which contains your “ec_pkcs11client.ini” file.

Step 2: Run the Signing Command

- Now, run the signing command from this directory.

```
<Path of Jarsigner tool> \
  -keystore NONE \
  -storepass NONE \
  -storetype PKCS11 \
  -sigalg SHA256withRSA \
  -providerClass sun.security.pkcs11.SunPKCS11 \
  -providerArg <Path of pkcs11properties.cfg> \
  -signedjar <Path of the file after signing> \
  <Path of the file to be signed> \
  <Key alias of the signing certificate> \
  -tsa http://timestamp.digicert.com
```

- A sample command is provided below:

```
jarsigner \
  -keystore NONE \
  -storepass NONE \
  -storetype PKCS11 \
  -sigalg SHA256withRSA \
  -providerClass sun.security.pkcs11.SunPKCS11 \
  -providerArg pkcs11properties.cfg \
  -signedjar helloworld_signed.jar \
  helloworld.jar gpg2 \
  -tsa http://timestamp.digicert.com
```

Step 3: Verify the Signature

- For verification, run the following command:

```
<Path of Jarsigner tool> -verify \
  <Path of the file after signing> \
  -certs -verbose
```

- A sample command is provided below:

```
jarsigner -verify helloworld_signed.jar -certs -verbose
```

Step 4: Option Reference

- The following are the options and their meaning in the commands:

Option	Description
`-keystore NONE`	The keystore. This is NONE because the keys live in the HSM rather than in a keystore file.
`-storepass NONE`	The keystore password. This is NONE because access is controlled by the wrapper configuration.
`-storetype PKCS11`	The keystore type, which must be PKCS11.
`-sigalg`	The signature algorithm to use when signing, for example SHA256withRSA.
`-providerClass`	The Java security provider to use. This is always sun.security.pkcs11.SunPKCS11, the JDK's built-in PKCS#11 provider.
`-providerArg`	The path of the pkcs11properties.cfg file, which tells the provider how to reach the PKCS#11 Wrapper.
`-signedjar`	The path of the signed JAR file to produce.
`<jar file>`	The JAR file to be signed, given as a positional argument with no option name.
`<alias>`	The key alias of the signing certificate, given as a positional argument immediately after the JAR file.
`-tsa`	The URL of the timestamping authority, for example http://timestamp.digicert.com.
`-verify`	Checks the signature on an already signed JAR.
`-certs -verbose`	Shows the signer certificates and detailed output when verifying.

Step 5: Cross-Check in CodeSign Secure

- Open the CodeSign Secure portal and navigate to Reports > Signing Request Report, where every signing request performed through the wrapper is recorded for audit purposes.

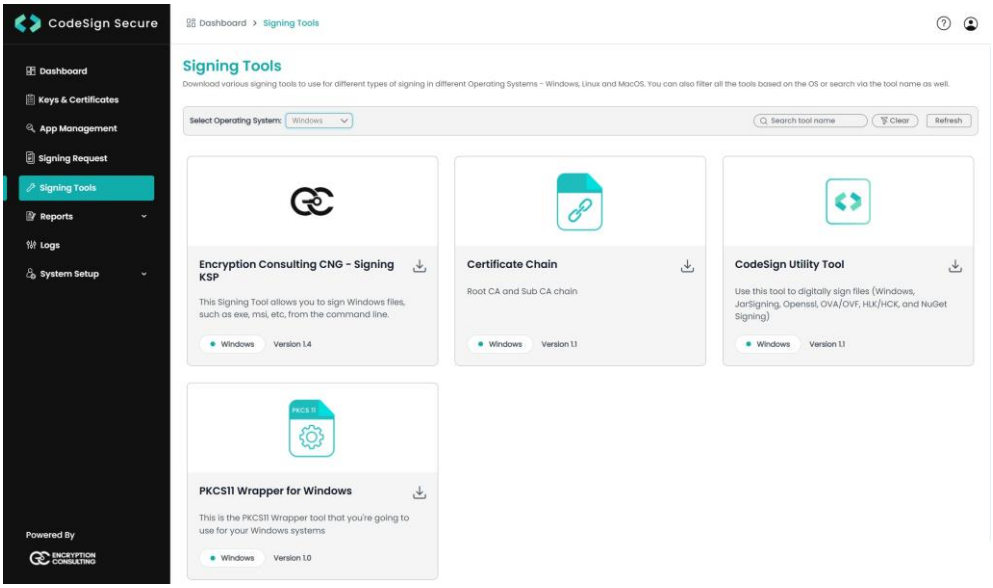
6. Windows: Install and Configure

To perform signing using the Jarsigner Tool and our PKCS11 Wrapper in a Windows machine, you would need to follow the below steps.

Steps:

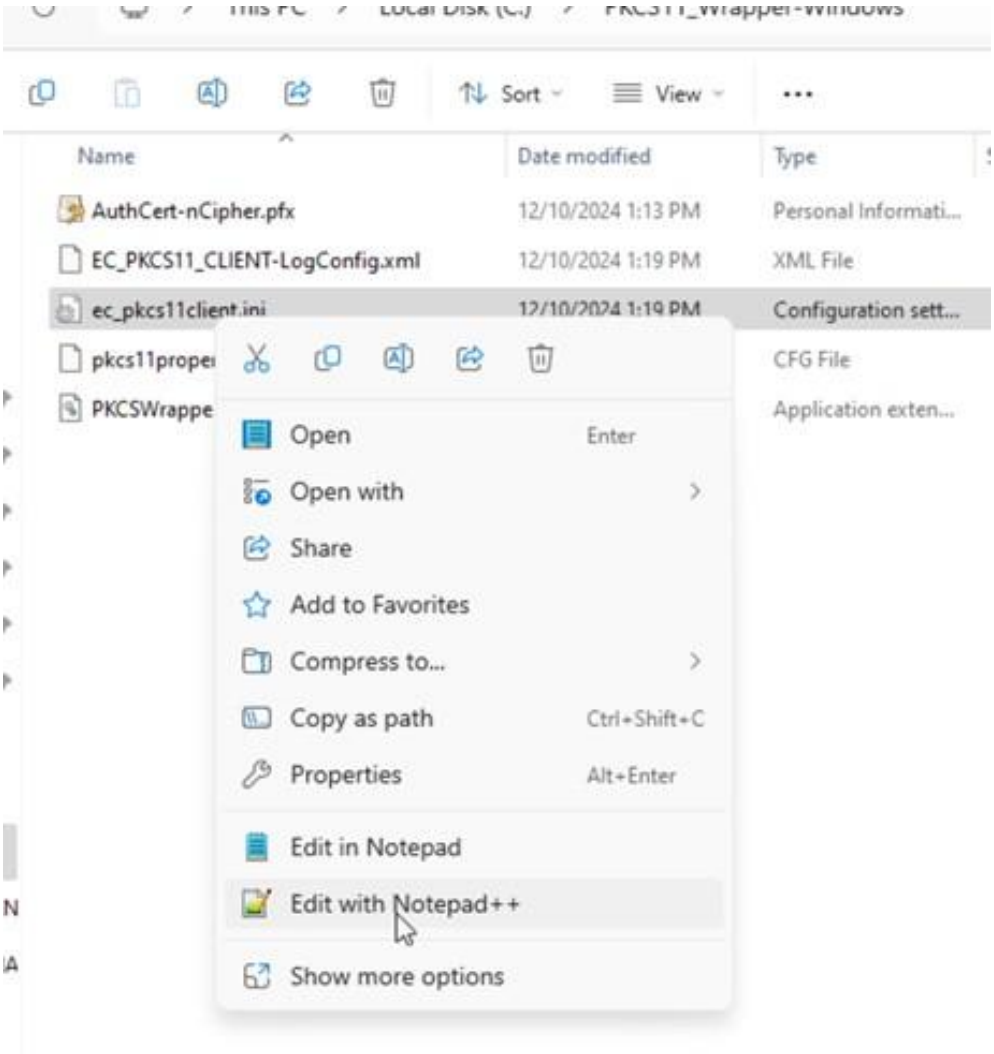
Step 1: Download the PKCS#11 Wrapper for Windows

- Go to EC CodeSign Secure v3.02’s Signing Tools section and download the PKCS11 Wrapper for Windows.



Step 2: Edit the Configuration Files

- Go to your Windows client system and edit the configuration files (ec_pkcs11client.ini and pkcs11properties.cfg) downloaded in the PKCS11 Wrapper.



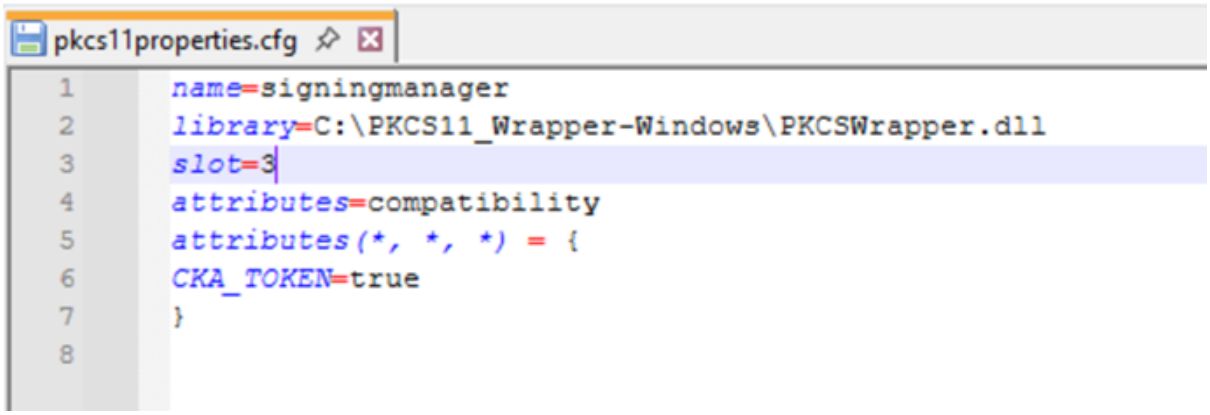
ec_pkcs11client.ini

```
[logconfig_file]
name=C:\PKCS11_Wrapper-Windows\EC_PKCS11_CLIENT-LogConfig.xml

[server_url]
url=https://devcodesignsecure.encryptionconsulting.com

[pfxfile_path]
path=C:\PKCS11_Wrapper-Windows\Pkcs11Test.pfx

[pfxfile_passwd]
passwd=bb881220628e

[login_token]
username=Aryan@encryptionconsulting.com
passcode=8@!$U3QXS5Xtu2vf
```

pkcs11properties.cfg

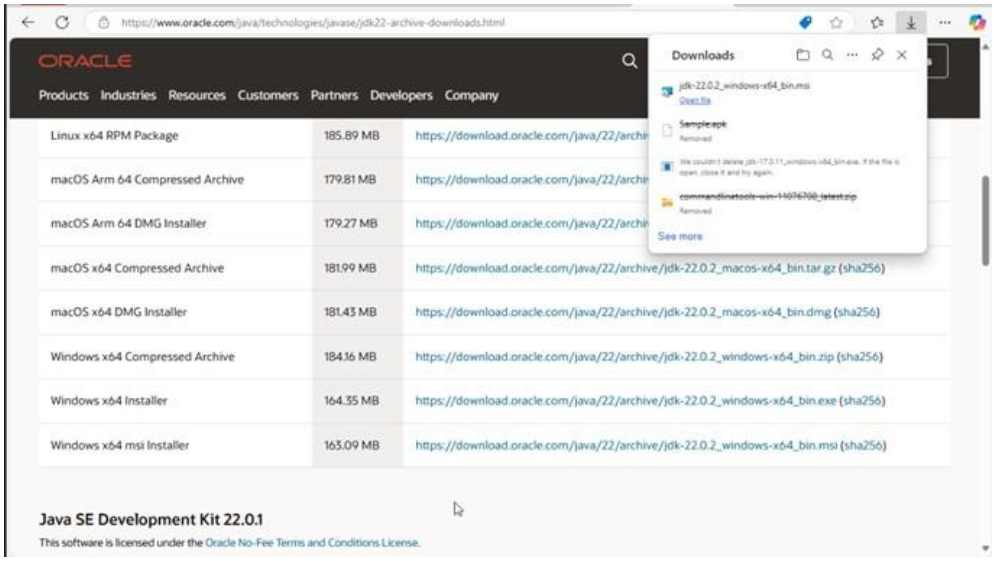
```
name=signingmanager
library=C:\PKCS11_Wrapper-Windows\PKCSWrapper.dll
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
```

NOTE: When using Luna HSM, add the slot number in the pkcs11properties.cfg file as shown below.



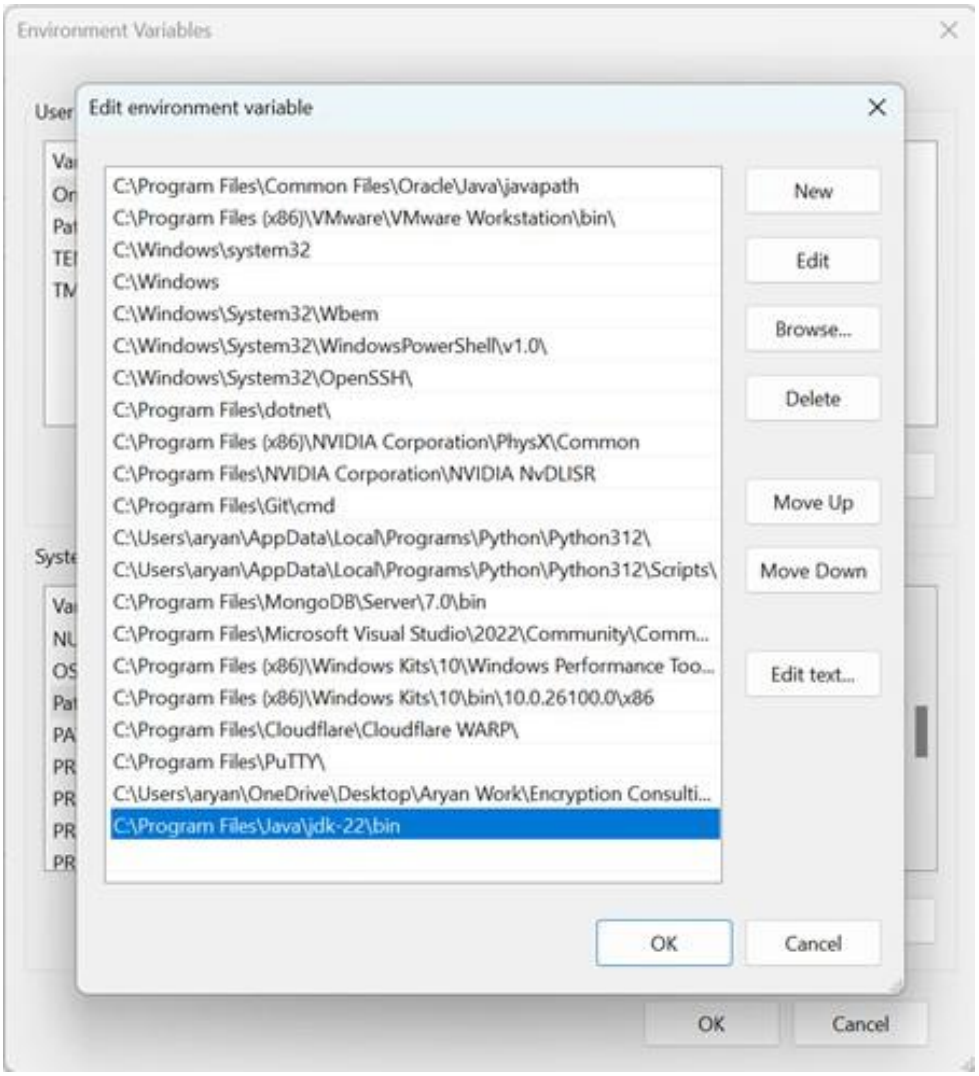
Step 3: Install and Activate Java

- Now, let us install some prerequisites (Java 8 to 22) in your client system to run the PKCS11 Wrapper.
- Install Java 22 from Oracle's official site, and follow the instructions in the msi file.


https://www.oracle.com/java/technologies/javase/jdk22-archive-downloads.html
ORACLE
Products Industries Resources Customers Partners Developers Company
Downloads
jdk-22.0.2_windows-x64_bin.msi
Open file
Sample.apk
Removed
See more
Linux x64 RPM Package 185.89 MB
macOS Arm 64 Compressed Archive 179.81 MB
macOS Arm 64 DMG Installer 179.27 MB
macOS x64 Compressed Archive 181.99 MB https://download.oracle.com/java/22/archive/jdk-22.0.2_macos-x64_bin.tar.gz (sha256)
macOS x64 DMG Installer 181.43 MB https://download.oracle.com/java/22/archive/jdk-22.0.2_macos-x64_bin.dmg (sha256)
Windows x64 Compressed Archive 184.16 MB https://download.oracle.com/java/22/archive/jdk-22.0.2_windows-x64_bin.zip (sha256)
Windows x64 Installer 164.35 MB https://download.oracle.com/java/22/archive/jdk-22.0.2_windows-x64_bin.exe (sha256)
Windows x64 msi Installer 163.09 MB https://download.oracle.com/java/22/archive/jdk-22.0.2_windows-x64_bin.msi (sha256)
Java SE Development Kit 22.0.1
This software is licensed under the Oracle No-Fee Terms and Conditions License.


Java(TM) SE Development Kit 22.0.2 (64-bit) - Setup
Java
ORACLE
Welcome to the Installation Wizard for Java SE Development Kit 22.0.2
This wizard will guide you through the installation process for the Java SE Development Kit 22.0.2.
Next >
Cancel

- Set Java 22 as the active version by storing the bin path in the PATH variable.



NOTE: The supported Java range differs by platform in the product documentation: Java 8 to 17 for Linux and macOS, and Java 8 to 22 for Windows.

- We are now ready with our Windows environment to perform the signing using Jarsigner tool and PKCS11 Wrapper.

7. Windows: Sign and Verify

As on the other platforms, the signing command is run from the directory containing your configuration files.

Steps:

Step 1: Change to the Working Directory

- For signing, change the working directory of the terminal to that folder which contains your “ec_pkcs11client.ini” file.

```
C:\>cd PKCS11_Wrapper-Windows

C:\PKCS11_Wrapper-Windows>dir
 Volume in drive C has no label.
 Volume Serial Number is 865B-2F9D

 Directory of C:\PKCS11_Wrapper-Windows

02/06/2025  11:48 AM    <DIR>          .
02/06/2025  11:56 AM             6,122 build_project.ps1
02/04/2025  06:00 PM               248 ec_pkcs11client.ini
02/04/2025  05:36 PM               525 EC_PKCS11_CLIENT-LogConfig.xml
02/06/2025  11:56 AM           843,332 ec_pkcs11_client.log
02/04/2025  05:36 PM         2,104,682 jsign-7.0.jar
02/04/2025  06:08 PM               136 pkcs11properties.cfg
02/04/2025  05:39 PM             2,800 Pkcs11Test.pfx
02/06/2025  11:47 AM         2,285,056 PKCSWrapper.dll
               8 File(s)      5,242,901 bytes
               1 Dir(s)     384,016,384 bytes free
```

Step 2: Run the Signing Command

- Now, run the signing command from this directory.

```
<Path of Jarsigner tool> ^
  -keystore NONE ^
  -storepass NONE ^
  -storetype PKCS11 ^
  -sigalg SHA256withRSA ^
  -providerClass sun.security.pkcs11.SunPKCS11 ^
  -providerArg <Path of pkcs11properties.cfg> ^
  -signedjar <Path of the file after signing> ^
  <Path of the file to be signed> ^
  <Key alias of the signing certificate> ^
  -tsa http://timestamp.digicert.com
```

- A sample command is provided below:

```
jarsigner ^
  -keystore NONE ^
  -storepass NONE ^
  -storetype PKCS11 ^
  -sigalg SHA256withRSA ^
  -providerClass sun.security.pkcs11.SunPKCS11 ^
  -providerArg pkcs11properties.cfg ^
  -signedjar helloworld_signed.jar ^
  helloworld.jar gpg2 ^
  -tsa http://timestamp.digicert.com
```

NOTE: The caret characters are Command Prompt line continuations. You may keep them or join the command onto a single line.

Step 3: Verify the Signature

- For verification, run the following command:

```
<Path of Jarsigner tool> -verify ^
  <Path of the file after signing> ^
  -certs -verbose
```

- A sample command is provided below:

```
jarsigner -verify helloworld_signed.jar -certs -verbose
```

Step 4: Option Reference

- The following are the options and their meaning in the commands:

Option	Description
-keystore NONE	The keystore. This is NONE because the keys live in the HSM rather than in a keystore file.
-storepass NONE	The keystore password. This is NONE because access is controlled by the wrapper configuration.
-storetype PKCS11	The keystore type, which must be PKCS11.
-sigalg	The signature algorithm to use when signing, for example SHA256withRSA.
-providerClass	The Java security provider to use. This is always sun.security.pkcs11.SunPKCS11, the JDK's built-in PKCS#11 provider.
-providerArg	The path of the pkcs11properties.cfg file, which tells the provider how to reach the PKCS#11 Wrapper.
-signedjar	The path of the signed JAR file to produce.
<jar file>	The JAR file to be signed, given as a positional argument with no option name.
<alias>	The key alias of the signing certificate, given as a positional argument immediately after the JAR file.
-tsa	The URL of the timestamping authority, for example http://timestamp.digicert.com.
-verify	Checks the signature on an already signed JAR.
-certs -verbose	Shows the signer certificates and detailed output when verifying.

Step 5: Cross-Check in CodeSign Secure

- Open the CodeSign Secure portal and navigate to Reports > Signing Request Report, where every signing request performed through the wrapper is recorded for audit purposes.