PKCS#11 Wrapper – APK Signing Integration Document

For APK signing, we use APKSigner, which is a command-line utility included in the Android SDK Build Tools that allows you to sign Android application package (APK) files and verify their signatures. It is an essential tool for Android developers to ensure the integrity and authenticity of their apps.

Encryption Consulting's PKCS#11 Wrapper exposes the keys held in your HSM through the standard PKCS#11 interface, which means APKSigner can use them through Java's built-in SunPKCS11 provider without any change to the tool itself. The keystore is given as type PKCS11 rather than a local file, so the private key never leaves the HSM and every signing operation is recorded centrally.

Section 1 covers the steps that are 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 sample APK file to sign.

1. Common Setup

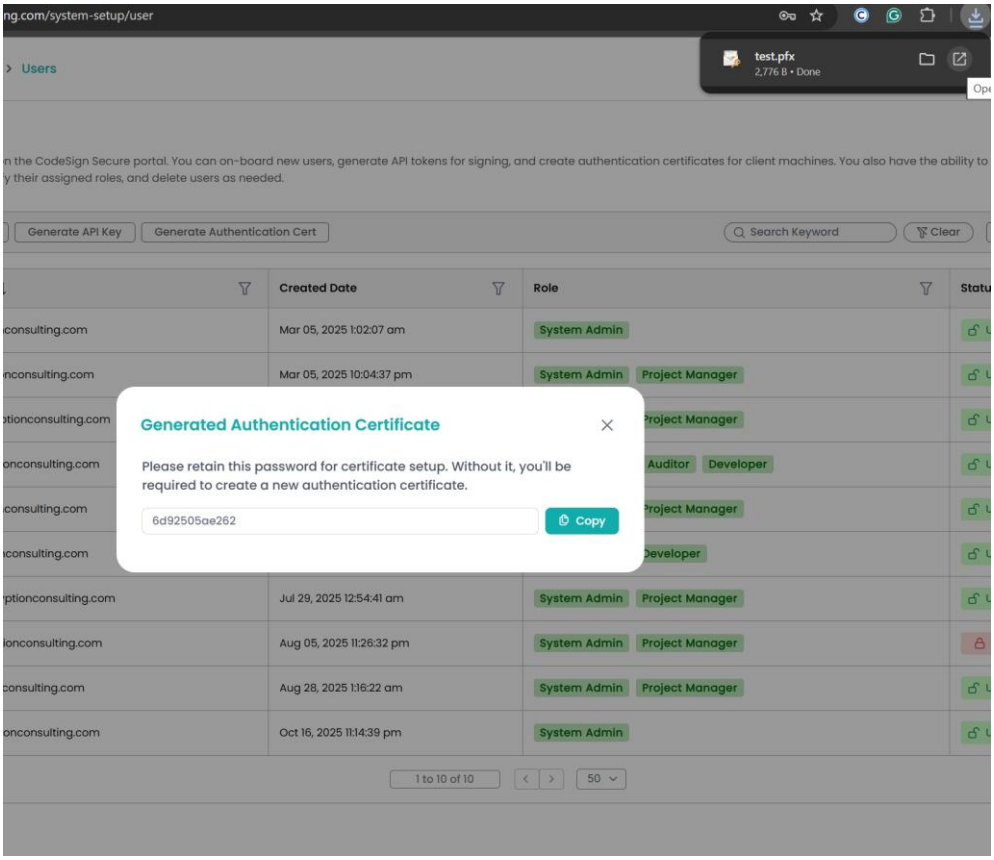
Two things are needed regardless of platform: a P12 authentication certificate that identifies your machine to CodeSign Secure, and the two configuration files that ship inside the PKCS#11 Wrapper download.

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 .pfx 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 referenced by the wrapper configuration 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:
  - **ec_pkcs11client.ini –** Points the wrapper at your CodeSign Secure server and at the P12 authentication certificate generated in Step 1.
  - **pkcs11properties.cfg –** The SunPKCS11 provider configuration that Java reads. Its path is supplied to APKSigner through the --provider-arg option.

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

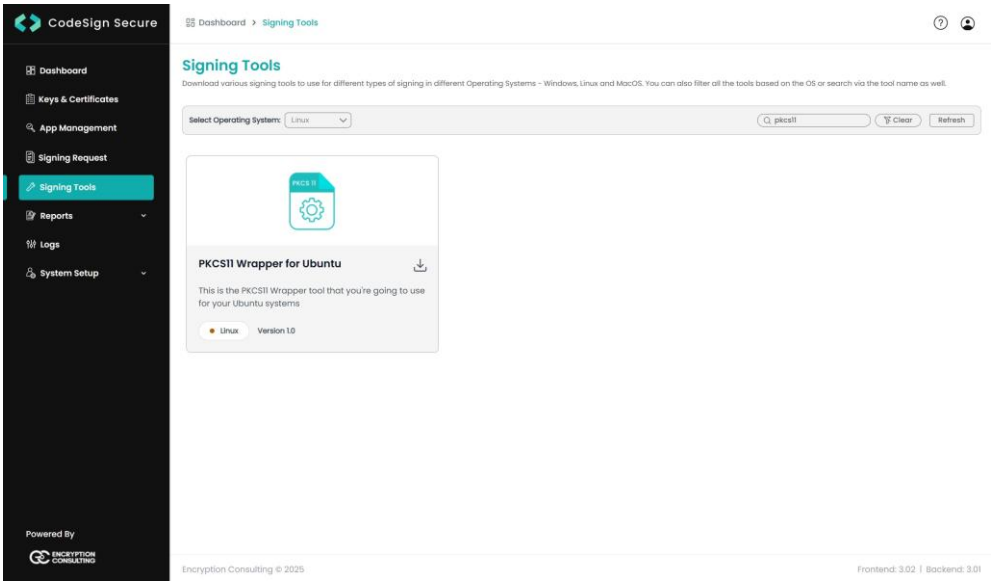
2. Linux (Ubuntu): Set Up the Wrapper

To sign your APK using APKSigner and our PKCS#11 wrapper in Ubuntu, please 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
```

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

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Locati
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To
```

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

```
source ~/.bashrc
```

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

- Check whether the variable has been set correctly:

```
echo $EC_INI_FILE_PATH
```

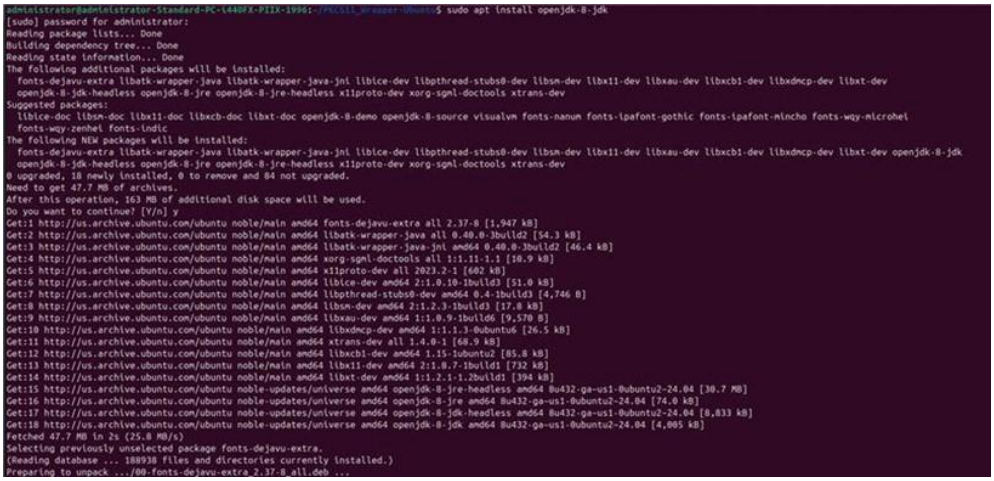
```
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$ echo $EC_INI_FILE_PATH
/home/administrator/PKCS11_Wrapper-Ubuntu/ec_pkcs11client.ini
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$
```

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

Step 4: Install the Prerequisites

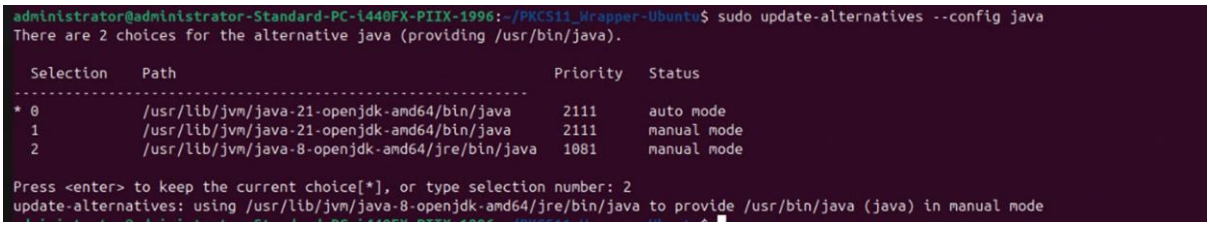
- Now, let us install some prerequisites in your client system to run the PKCS11 Wrapper.
  - Install Java 8:

```
sudo apt install openjdk-8-jdk
```



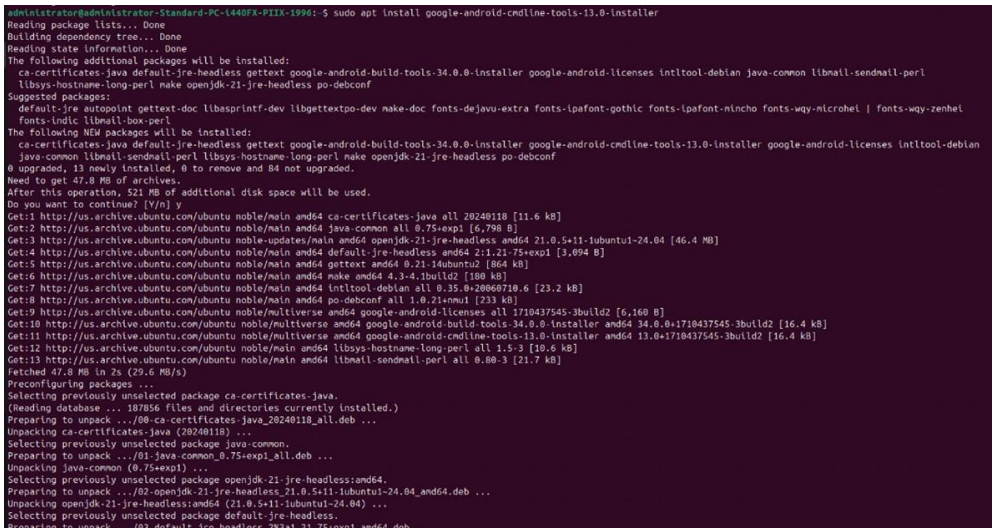
  - Set Java 8 as the active version:

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



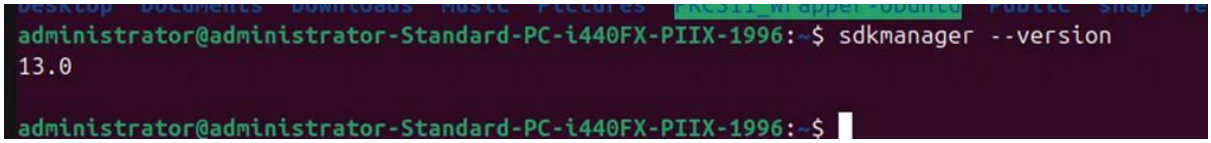
  - Install the Android SDK command-line tools:

```
sudo apt install google-android-cmdline-tools-13.0-installer
```



- Ensure that the SDK Manager for Android Studio has been properly installed:

```
sdkmanager --version
```



- Install Build tools using SDKManager, which contains the APKSigner:

```
sdkmanager "build-tools;34.0.0"
```

- Ensure that APKSigner is present:

```
apksigner --version
```

```
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~$ apksigner --version
0.9
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~$
```

- Two packages are required to run the PKCS11 Wrapper on your system. First, install liblog4cxx-dev:

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

```
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$ sudo apt-get install liblog4cxx-dev
[sudo] password for administrator:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
liblog4cxx-dev is already the newest version (1.1.0-1build3).
0 upgraded, 0 newly installed, 0 to remove and 81 not upgraded.
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$
```

- The last prerequisite is to install the curl package:

```
sudo apt-get install curl
```

```
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$ sudo apt-get install curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 81 not upgraded.
Need to get 227 kB of archives.
After this operation, 534 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu noble-updates/main amd64 curl amd64 8.5.0-2ubuntu10.5 [227 kB]
Fetched 227 kB in 0s (694 kB/s)
Selecting previously unselected package curl.
(Reading database ... 191536 files and directories currently installed.)
Preparing to unpack .../curl_8.5.0-2ubuntu10.5_amd64.deb ...
Unpacking curl (8.5.0-2ubuntu10.5) ...
Setting up curl (8.5.0-2ubuntu10.5) ...
Processing triggers for man-db (2.12.0-4build2) ...
administrator@administrator-Standard-PC-i440FX-PIIX-1996:~/PKCS11_Wrapper-Ubuntu$
```

3. Linux (Ubuntu): Sign and Verify

Now that all the configurations and prerequisites have been installed, let us perform the signing operation.

Steps:

Step 1: Run the Signing Command

- The signing command will look something like this. Ensure you run this command only inside the folder where your PKCS11 Wrapper is installed.

```
apksigner sign \
  --provider-class sun.security.pkcs11.SunPKCS11 \
  --provider-arg <path of the pkcs11properties.cfg file> \
  --ks NONE \
  --ks-type PKCS11 \
  --ks-pass pass:abcd1234 \
```

```
  --ks-key-alias <private key alias> \
  --in <path of the APK file you want to sign> \
  --out <path of the Signed APK file>
```

- A sample command is provided below:

```
apksigner sign \
  --provider-class sun.security.pkcs11.SunPKCS11 \
  --provider-arg \
    /home/administrator/PKCS11_Wrapper-Ubuntu/pkcs11properties.cfg \
  --ks NONE --ks-type PKCS11 --ks-pass pass:abcd1234 \
  --ks-key-alias gpg2 --in Sample.apk --out signed.apk
```

NOTE: When working with Luna HSM, append --min-sdk-version 28 to the signing command shown above.

Step 2: Option Reference

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

Option	Description
--provider-class	The Java security provider to use. This is always sun.security.pkcs11.SunPKCS11, the JDK's built-in PKCS#11 provider.
--provider-arg	The path to the pkcs11properties.cfg file in your system, which tells the provider how to reach the PKCS#11 Wrapper.
--ks	The keystore. This is NONE because the keys live in the HSM rather than in a keystore file.
--ks-type	The keystore type, which must be PKCS11.
--ks-pass	The keystore password, supplied in the form pass:<password>.
--ks-key-alias	The alias of the private key to sign with.
--in	The path of the APK file you want to sign.
--out	The path of the signed APK file to produce.

Option	Description
`--min-sdk-version`	The lowest Android API level the signature must be valid for. Required when working with a Luna HSM.

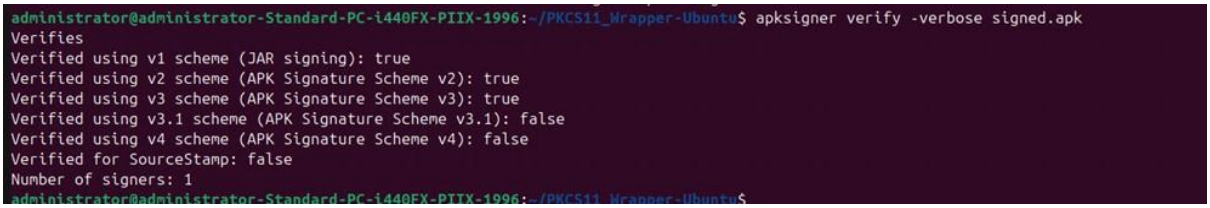
Step 3: Verify the Signature

- After successfully signing the APK, let us verify it using this command:

```
apksigner verify -verbose <path of the signed APK file>
```

- A sample command is provided below:

```
apksigner verify -verbose signed.apk
```



NOTE: When working with Luna HSM, use: apksigner verify -verbose --min-sdk-version 28 <path of the signed APK file>

Step 4: 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.

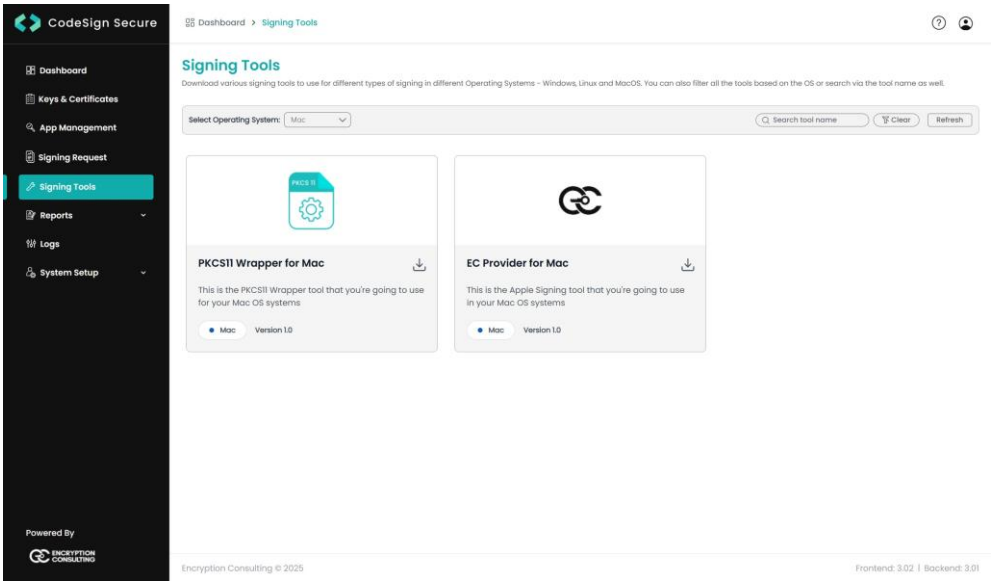
4. macOS: Set Up the Wrapper

To perform APK Signing using the APKSigner tool and our PKCS11 Wrapper on a macOS machine, you would need to follow the below steps.

Steps:

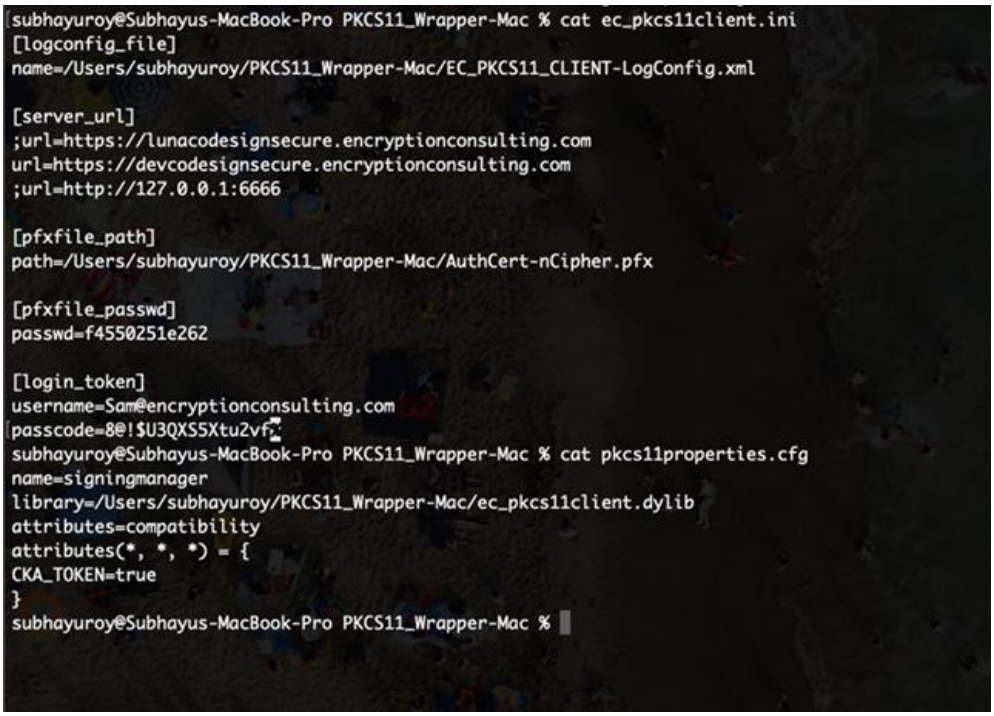
Step 1: Download the PKCS#11 Wrapper for macOS

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



Step 2: Edit the Configuration Files

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



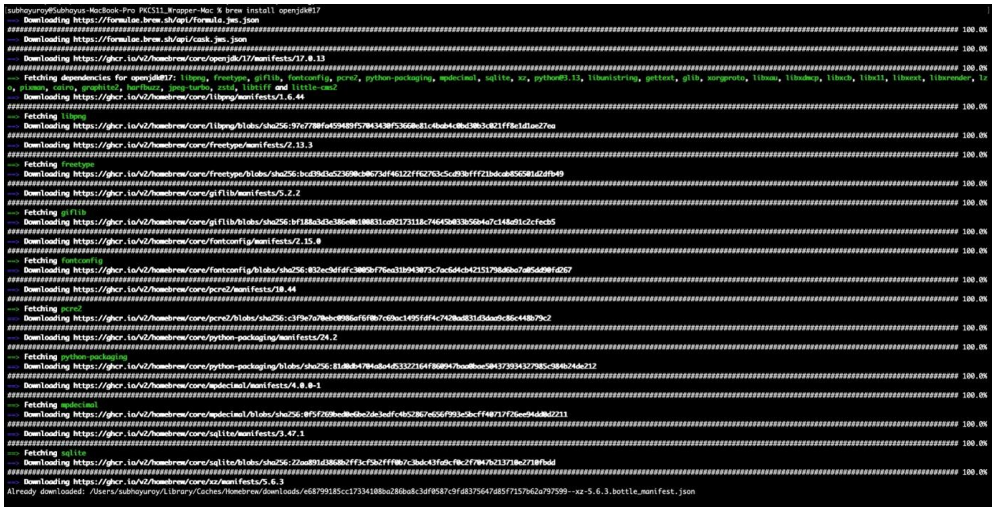
NOTE: When using Luna HSM, add the slot number in the pkcs11properties.cfg file as shown below.

```
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 %
```

Step 3: Install and Activate Java 17

- Install Java 17:

```
brew install openjdk@17
```



- Set Java 17 as the active version. For Zsh:

```
nano ~/.zshrc
```

```
 UW PICO 5.09                                                    File: /Users/subhayuroy/.zshrc

export PATH=/usr/local/bin:$PATH
export PATH=/Users/subhayuroy/flutter/bin:$PATH
export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export PATH="$PATH":"$HOME/.pub-cache/bin"
export PATH=/Users/subhayuroy/PKCS11_Wrapper-Mac/cmdline-tools/bin:$PATH
export JAVA_HOME=$(/usr/libexec/java_home -v 17)
export PATH=$JAVA_HOME/bin:$PATH
```

- For Bash:

```
nano ~/.bash_profile
```

```
subhayuroy@Subhayus-MacBook-Pro bin % /usr/libexec/java_home -v 17

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home
subhayuroy@Subhayus-MacBook-Pro bin % nano ~/.zshrc
subhayuroy@Subhayus-MacBook-Pro bin % source ~/.zshrc
subhayuroy@Subhayus-MacBook-Pro bin %
```

- Add these lines:

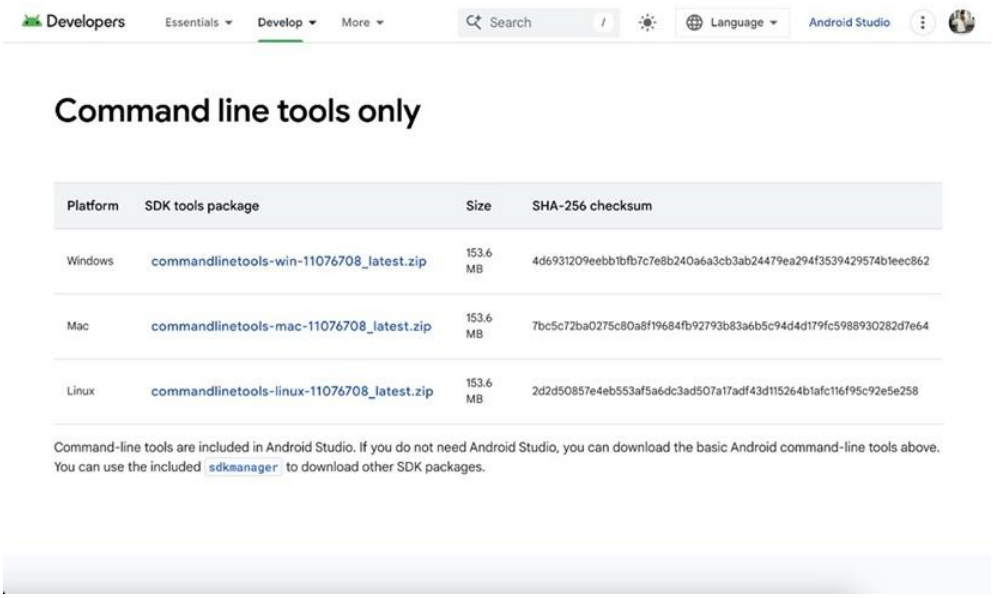
```
export JAVA_HOME=$(/usr/libexec/java_home -v 17)
export PATH=$JAVA_HOME/bin:$PATH
```

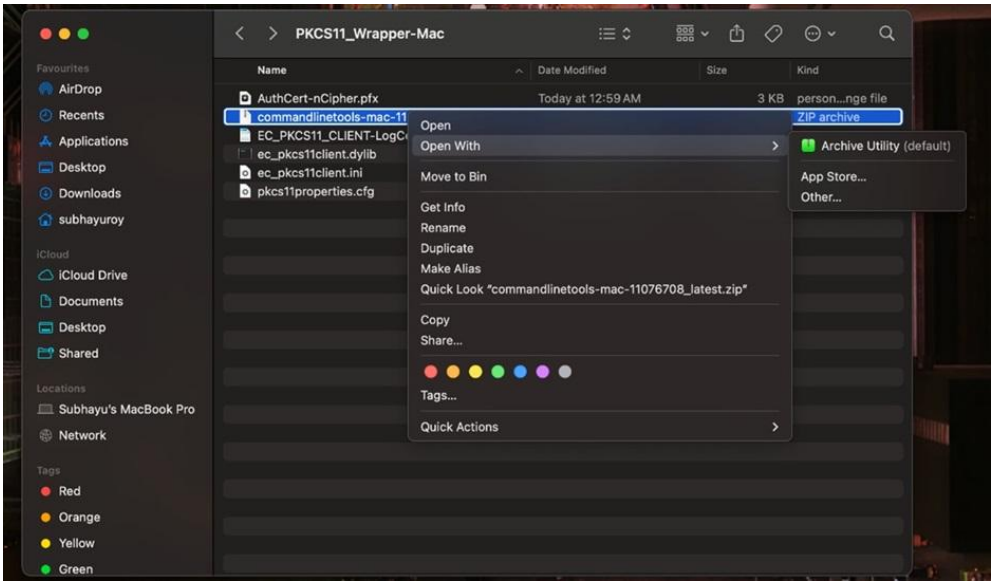
- And then run one of the following, matching your shell:

```
source ~/.zshrc
source ~/.bash_profile
```

Step 4: Install the Android SDK Command-Line Tools

- Install the Android SDK command-line tools from this site.





- Ensure that the SDK Manager for Android Studio has been properly installed:

```
sdkmanager --sdk_root=/Users/subhayuroy/PKCS11_Wrapper-Mac \
  --version
```

```
subhayuroy@Subhayus-MacBook-Pro bin % sdkmanager --sdk_root=/Users/subhayuroy/PKCS11_Wrapper-Mac --version

12.0

subhayuroy@Subhayus-MacBook-Pro bin %
```

- Install Build tools using SDKManager, which contains the APKSigner:

```
sdkmanager --sdk_root=/Users/subhayuroy/PKCS11_Wrapper-Mac \
  "build-tools;34.0.0"
```

```
subhayuroy@Subhayus-MacBook-Pro bin % sdkmanager --sdk_root=/Users/subhayuroy/PKCS11_Wrapper-Mac "build-tools;34.0.0"
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
License android-sdk-license:          ] 10% Computing updates...
---------------------------------------
Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the Licens
e Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.

1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which m
ay be updated from time to time; and (ii) successfully passes the Android Compatibility Test Suite (CTS).

1.4 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States.


2. Accepting the License Agreement

2.1 In order to use the SDK, you must first agree to the License Agreement. You may not use the SDK if you do not accept the License Agreement.

2.2 By clicking to accept, you hereby agree to the terms of the License Agreement.

2.3 You may not use the SDK and may not accept the License Agreement if you are a person barred from receiving the SDK under the laws of the United States or other countries, including the country in which you are resident or from which y
ou use the SDK.

2.4 If you are agreeing to be bound by the License Agreement on behalf of your employer or other entity, you represent and warrant that you have full legal authority to bind your employer or such entity to the License Agreement. If you do
 not have the requisite authority, you may not accept the License Agreement or use the SDK on behalf of your employer or other entity.
```

- Ensure that APKSigner is present:

```
ls /Users/subhayuroy/PKCS11_Wrapper-Mac/build-tools/34.0.0/\
apksigner
```

```
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac % ls /Users/subhayuroy/PKCS11_Wrapper-Mac/build-tools/34.0.0/apksigner

/Users/subhayuroy/PKCS11_Wrapper-Mac/build-tools/34.0.0/apksigner
subhayuroy@Subhayus-MacBook-Pro PKCS11_Wrapper-Mac %
```

Step 5: Install the Remaining Packages

- Two packages are required to run the PKCS11 Wrapper on your system. First, install the log4cxx library:

```
brew install log4cxx
```

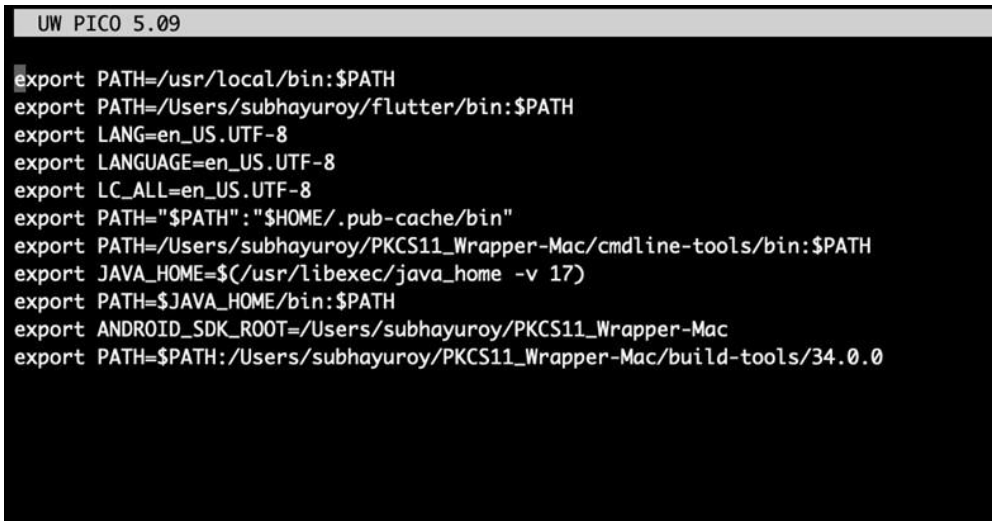
- The last prerequisite is to install the curl package:

```
brew install curl
```

Step 6: Add the Relative Paths to Your PATH

- You need to ensure all the relative paths are added to your PATH variable in the ~/.zshrc file:

```
export PATH=/Users/subhayuroy/PKCS11_Wrapper-Mac/cmdline-tools/\
bin:$PATH
export JAVA_HOME=$(/usr/libexec/java_home -v 17)
export PATH=$JAVA_HOME/bin:$PATH
export ANDROID_SDK_ROOT=/Users/subhayuroy/PKCS11_Wrapper-Mac
export PATH=$PATH:/Users/subhayuroy/PKCS11_Wrapper-Mac/\
build-tools/34.0.0
```



5. **macOS: Sign and Verify**

Now that all the configurations and prerequisites have been installed, let us perform the signing operation. On macOS the signing command invokes the APKSigner jar through java directly, so that the PKCS#11 module can be exported to the tool.

Steps:

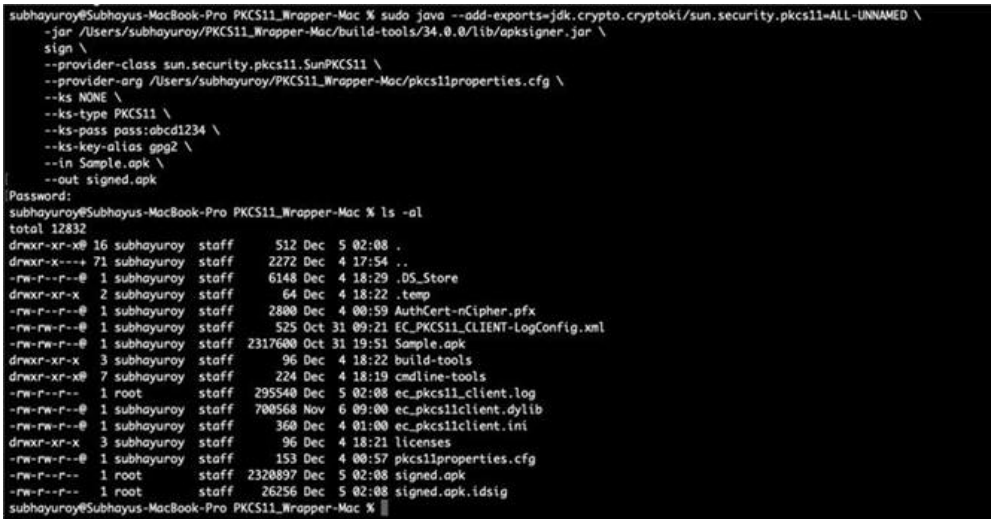
Step 1: Run the Signing Command

- The signing command will look something like this. Ensure you run this command only inside the folder where your PKCS11 Wrapper is installed.

```
java \
  --add-exports=jdk.crypto.cryptoki/sun.security.pkcs11=ALL-UNNAMED \
  -jar <path of the apksigner.jar in your system> sign \
  --provider-class sun.security.pkcs11.SunPKCS11 \
  --provider-arg <path of the pkcs11properties.cfg file> \
  --ks NONE \
  --ks-type PKCS11 \
  --ks-pass pass:abcd1234 \
  --ks-key-alias <private key alias> \
  --in <path of the APK file you want to sign> \
  --out <path of the Signed APK file>
```

- A sample command is provided below:

```
java \
  --add-exports=jdk.crypto.cryptoki/sun.security.pkcs11=ALL-UNNAMED \
  -jar /Users/subhayuroy/PKCS11_Wrapper-Mac/build-tools/34.0.0/\
lib/apksigner.jar \
  sign \
  --provider-class sun.security.pkcs11.SunPKCS11 \
  --provider-arg \
    /Users/subhayuroy/PKCS11_Wrapper-Mac/pkcs11properties.cfg \
  --ks NONE \
  --ks-type PKCS11 \
  --ks-pass pass:abcd1234 \
  --ks-key-alias gpg2 \
  --in Sample.apk \
  --out signed.apk
```



NOTE: The --add-exports option is required because the JDK used here keeps the SunPKCS11 classes in an encapsulated module. The Linux track does not need it because it runs on Java 8, which predates the module system.

Step 2: Verify the Signature

- After successfully signing the APK, let us verify it using this command:

```
apksigner verify -verbose <path of the signed APK file>
```

- A sample command is provided below:

```
apksigner verify -verbose signed.apk
```

Step 3: 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.

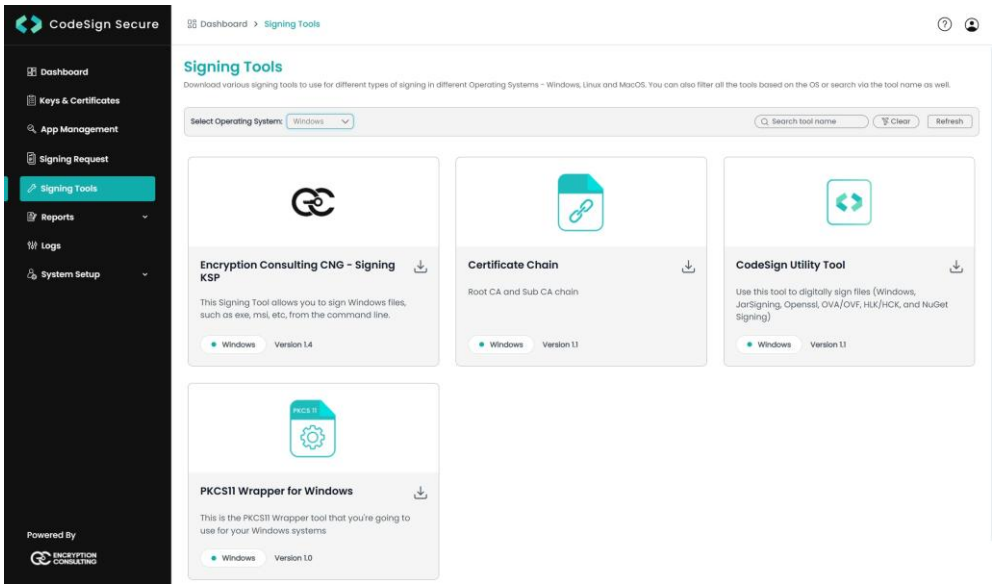
6. Windows: Set Up the Wrapper

To perform APK Signing using the APKSigner tool and our PKCS11 Wrapper on a Windows machine, you would need to follow the below steps.

Steps:

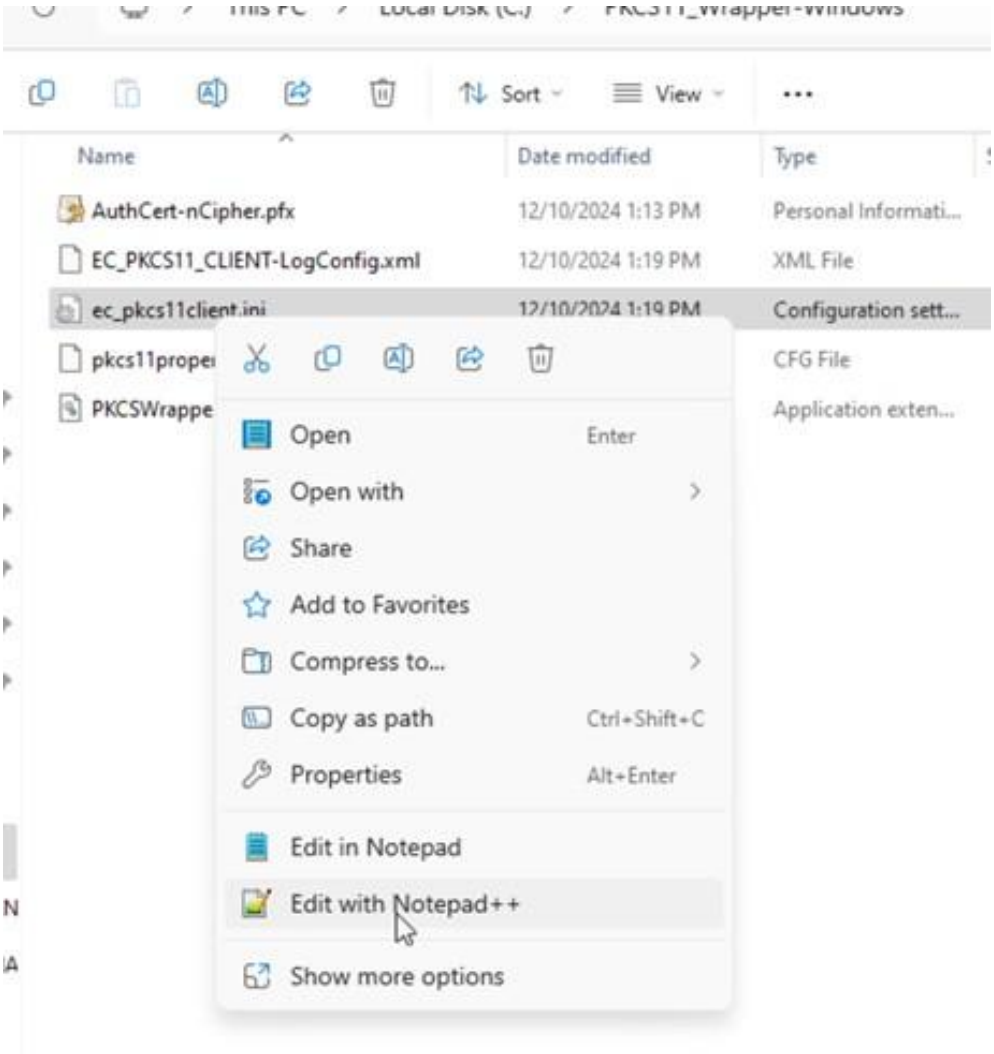
Step 1: Download the PKCS#11 Wrapper for Windows

- 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
}
```

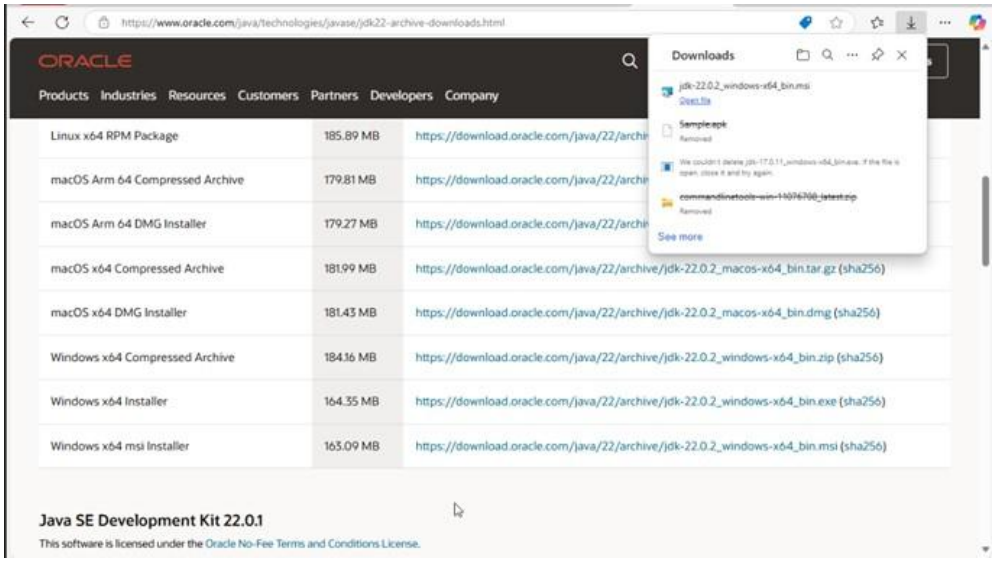
NOTE: When using Luna HSM, add the slot number in the pkcs11properties.cfg file as shown below.

pkcs11properties.cfg

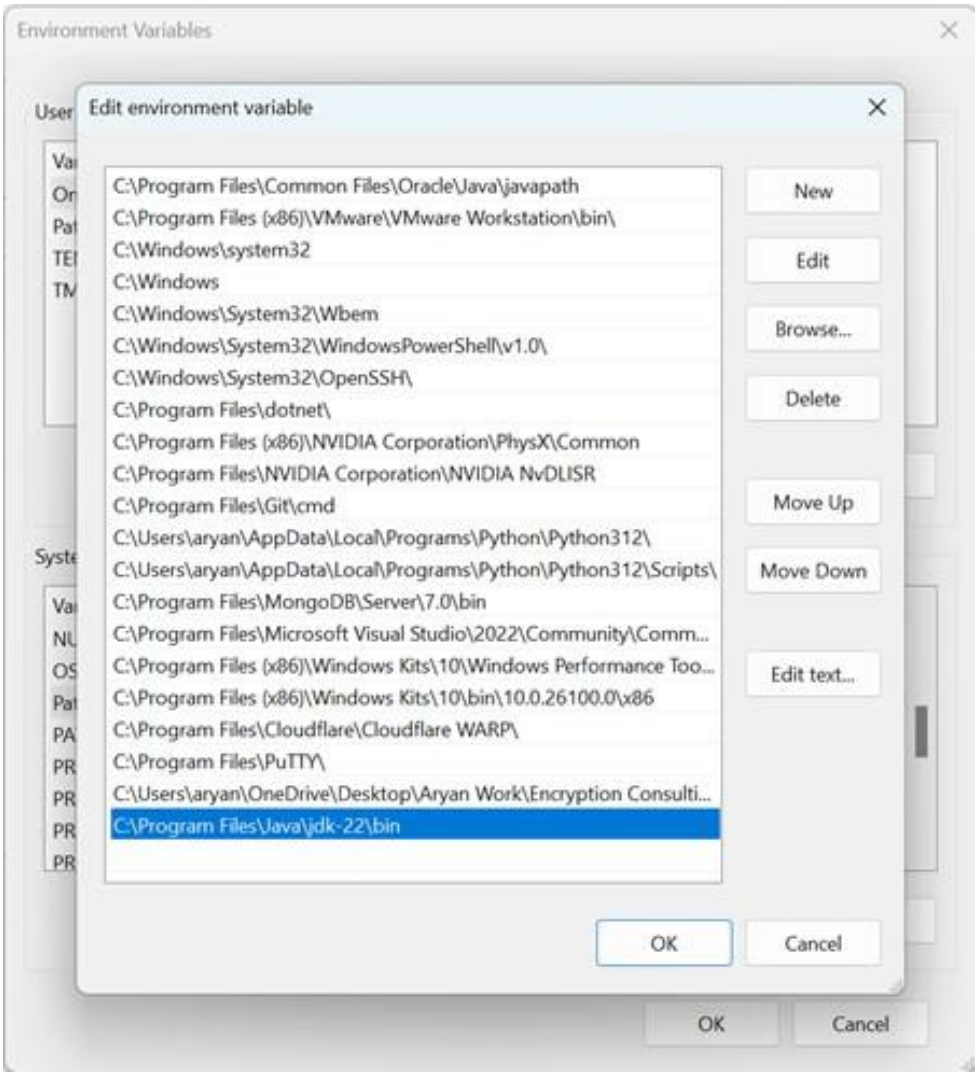
```
name=signingmanager
library=C:\PKCS11_Wrapper-Windows\PKCSWrapper.dll
slot=3
attributes=compatibility
attributes(*, *, *) = {
CKA_TOKEN=true
}
```

Step 3: Install and Activate Java 22

- Install Java 22 from Oracle's official site, and follow the instructions in the msi file.

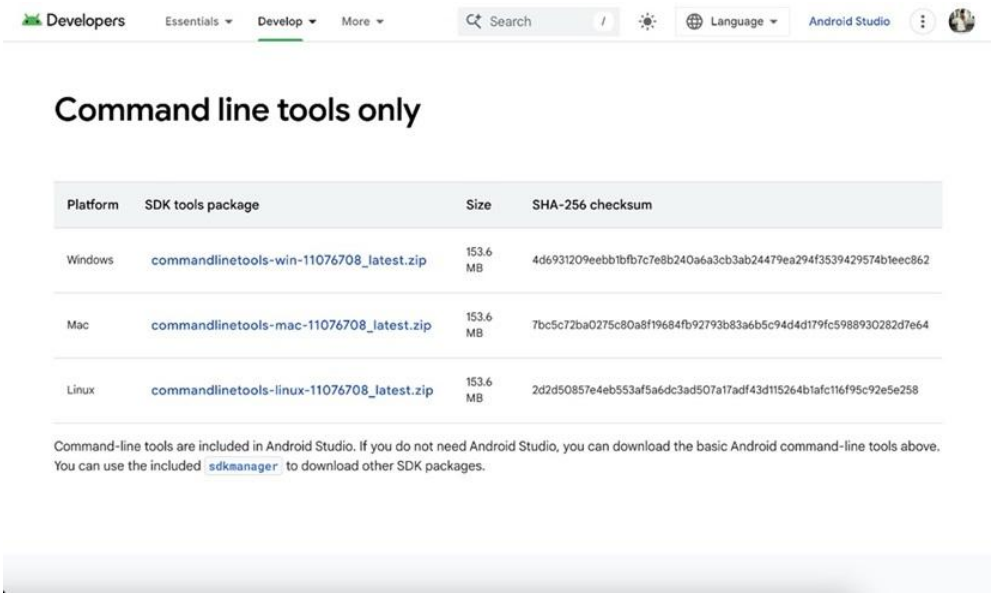


- Set Java 22 as the active version by storing the bin path in the PATH variable.

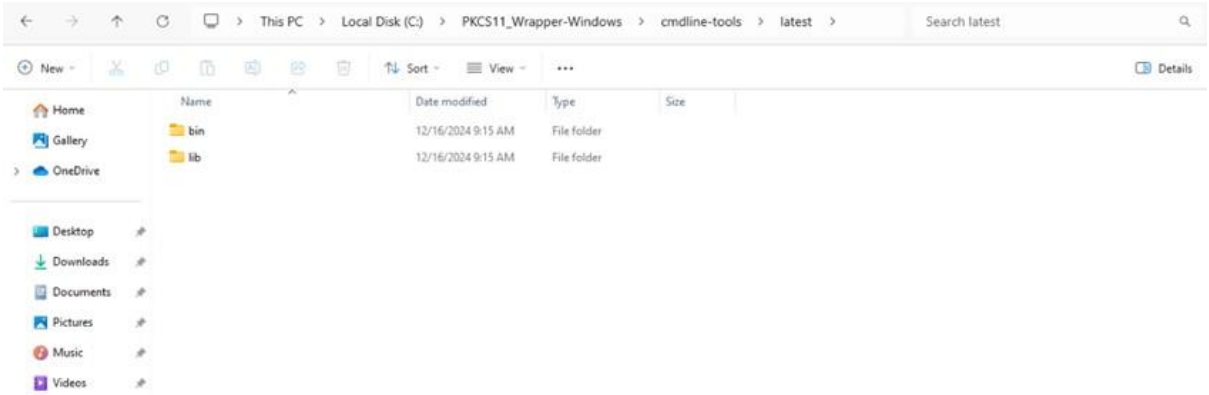


Step 4: Install the Android SDK Command-Line Tools

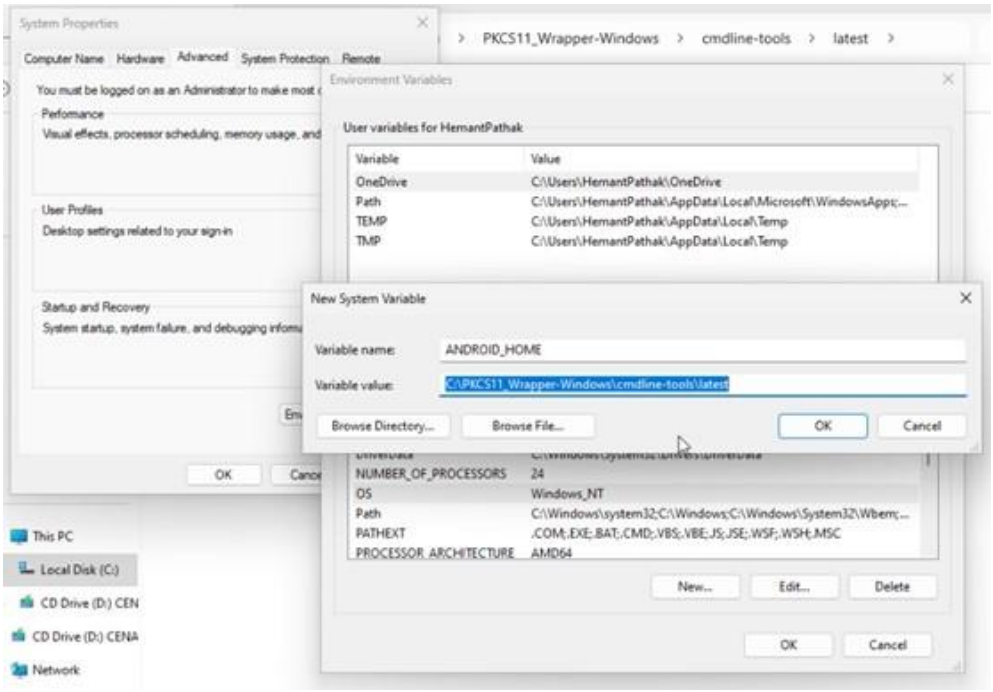
- Install the Android SDK command-line tools from this link here.



- Extract the files into a “cmdline-tools” folder and create a subfolder named latest. Now, move the bin and lib folders into the latest folder.



- Set an environment variable called ANDROID_HOME and set it to the path where you extracted the command line tools.



- Install Build tools using SDKManager, which contains the APKSigner:

```
.\bin\sdkmanager --channel=0 --install "build-tools;34.0.0"
```

```
C:\PKCS11_Wrapper-Windows\cmdline-tools\latest>.\bin\sdkmanager.bat --channel=0 --install "build-tools;34.0.0"
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
Warning: Errors during XML parse:
Warning: Additionally, the fallback loader failed to parse the XML.
[=====================                  ] 52% Unzipping... android-14/renderscri
```

- Ensure that APKSigner is present:

```
Apksigner.bat --version
```

```
C:\PKCS11_Wrapper-Windows\build-tools\31.0.0>apksigner.bat --version
0.9
C:\PKCS11_Wrapper-Windows\build-tools\31.0.0>
```

Step 5: Patch Apksigner.bat for the PKCS#11 Module

- Open the Apksigner.bat file and note its current last line.

```
    if "%a:~0,2%" NEQ "-J" goto notJ
        set javaOpts=%javaOpts% -%a:~2%
        shift /1
        goto firstArg

    :notJ
    set params=%params% %1
    shift /1
    goto firstArg

:endArgs

set javaOpts=%javaOpts% %defaultXmx% %defaultXss%
call "%java_exe%" %javaOpts% -jar "%jarpath%" %params%
```

- Update the last line to the below command:

```
call "%java_exe%" --add-exports ^
  jdk.crypto.cryptoki/sun.security.pkcs11=ALL-UNNAMED ^
  %javaOpts% -jar "%jarpath%" %params%
```

```
    if "%a:~0,5%" NEQ "-JXss" goto notXss
        set defaultXss=
    :notXss

    if "%a:~0,2%" NEQ "-J" goto notJ
        set javaOpts=%javaOpts% -%a:~2%
        shift /1
        goto firstArg

    :notJ
    set params=%params% %1
    shift /1
    goto firstArg

:endArgs

set javaOpts=%javaOpts% %defaultXmx% %defaultXss%
call "%java_exe%" --add-exports jdk.crypto.cryptoki/sun.security.pkcs11=ALL-UNNAMED %javaOpts% -jar "%jarpath%" %params%
```

NOTE: This is a single logical line in the batch file. The caret characters shown above are batch line continuations, so you may either keep them or join the three lines into one.

NOTE: This edit performs the same job as the --add-exports option used in the macOS command, so the Windows signing command itself stays short.

7. Windows: Sign and Verify

Now that all the configurations and prerequisites have been installed, let us perform the signing operation.

Steps:

Step 1: Run the Signing Command

- The signing command will look something like this. Ensure you run this command only inside the folder where your PKCS11 Wrapper is installed.

```
apksigner sign ^
  --provider-class sun.security.pkcs11.SunPKCS11 ^
  --provider-arg <path of the pkcs11properties.cfg file> ^
  --ks NONE ^
  --ks-type PKCS11 ^
  --ks-pass pass:abcd1234 ^
  --ks-key-alias <private key alias> ^
  --in <path of the APK file you want to sign> ^
  --out <path of the Signed APK file>
```

- A sample command is provided below:

```
apksigner sign ^
  --provider-class sun.security.pkcs11.SunPKCS11 ^
  --provider-arg ^
    C:\Users\riley\Downloads\PKCS11_Wrapper-Windows\pkcs11properties.cfg ^
  --ks NONE --ks-type PKCS11 --ks-pass pass:secretpassword ^
  --ks-key-alias gpg2 --in Sample.apk --out signed.apk
```

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

NOTE: The --ks-pass value must match the password configured for your wrapper. The sample above uses a different value from the Linux and macOS samples, so use whichever applies to your own configuration.

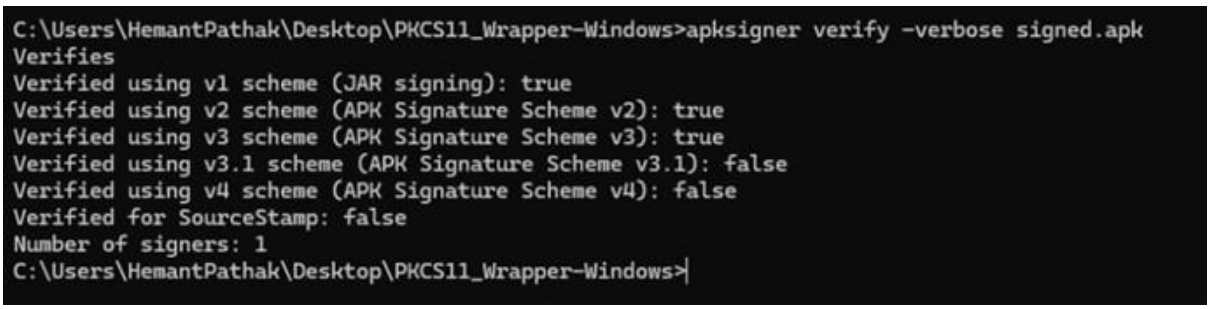
Step 2: Verify the Signature

- After successfully signing the APK, let us verify it using this command:

```
apksigner verify -verbose <path of the signed APK file>
```

- A sample command is provided below:

```
apksigner verify -verbose signed.apk
```



Step 3: 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.