JikGuard Android Game Protection Guide

Use the JikGuard command-line tool to protect an Android game build, sign an APK or Android App Bundle (AAB), or encrypt Unity and Cocos assets separately.

Set up the tool

Install a Java Development Kit (JDK) on Windows, macOS, or Linux. Keep `config.ini` and the `modules` folder next to the JAR. On macOS and Linux, **grant execute permission** to the platform executables in `modules`.

Open a terminal in the tool directory. The examples use `JikGuard.jar`. Substitute the JAR filename supplied with your release. File paths can be absolute or relative. Put paths containing spaces in double quotes.

Enter your game key

Open `config.ini` and enter the game key for your project. Register for a JikGuard account at user.jikguard.com.

```
[gamekey]
key=YOUR_GAME_KEY
```

Protect an APK or app bundle

Run the command for your build. The `-autoconfig` option loads your project's protection settings. The required `-inputfile` option identifies the build to protect.

```
java -jar JikGuard.jar -autoconfig -inputfile game.apk
java -jar JikGuard.jar -autoconfig -inputfile game.aab
```

By default, JikGuard writes `game_jikguard_protected.apk` or `game_jikguard_protected.aab` next to the input file. To choose another name or location, add `-outputfile` followed by the desired path.

Set a working directory or a custom engine library

To choose where JikGuard creates its temporary mini packages, create a writable directory and enter its path under `[packworkdir]`. With the default APK settings, the mini package is created beside the input build.

```
[packworkdir]
dir=C:\Builds\JikGuardWork
```

For a game engine other than Unity, Unreal Engine, or Cocos, enter the filename of its native shared library under `[engineso]`.

```
[engineso]
name=libMyEngine.so
```

Sign an Android build

You can sign the build during protection or sign an existing package in a separate step. JikGuard aligns APKs before signing. To sign an AAB, **add the JDK directory that contains `jarsigner` to your `PATH`**.

Configure the signing key

Set these four fields under `[signinfo]` in `config.ini`. Use only **ASCII characters** in the keystore path. The path can be relative or absolute.

```
[signinfo]
keystore-path=C:\Keys\game-release.jks
alias=release
password=YOUR_KEYSTORE_PASSWORD
alias-pwd=YOUR_KEY_PASSWORD
```

Use `password` for the keystore itself. Use `alias` to select the signing key inside it and `alias-pwd` for that key's password. Check all four values if signing fails.

For APKs, omitting `signver` uses the bundled signer's defaults for v1, v2, and v3 signing. If your release specifically requires v1 only, add `signver=v1` under `[signinfo]`.

Protect and sign in one step

Add `-sign` to the protection command for your package type.

```
java -jar JikGuard.jar -autoconfig -sign -inputfile game.apk
java -jar JikGuard.jar -autoconfig -sign -inputfile game.aab
```

Use Google Play App Signing

Use `-sha256` with the SHA-256 fingerprint of your **Google Play app signing key certificate**. In Play Console, copy the fingerprint from the app signing key certificate section. Replace `PLAY_CERT_SHA256` in this example with that value.

```
java -jar JikGuard.jar -autoconfig -sign -inputfile game.aab -sha256 PLAY_CERT_SHA256
```

Sign an existing APK or app bundle

Use the signing-only command for your file type. These commands use `[signinfo]` and accept a placeholder value for the game key.

```
java -jar JikGuard.jar -optype_sign_apk -inputfile game.apk
java -jar JikGuard.jar -optype_sign_jar -inputfile game.aab
```

The default outputs are `game_jikguard_signed.apk` and `game_jikguard_signed.aab` beside the input file. Use `-outputfile` to specify a different destination.

Encrypt game assets

You can encrypt assets separately from the APK or app bundle. Enter your project's game key in `config.ini` before running any of these commands.

Encrypt Unity AssetBundles

Pass a ZIP archive of AssetBundle files to `-optype_assetbundle_enc`. For example, this command writes `bundles_jikguard_protected.zip` beside `bundles.zip`.

```
java -jar JikGuard.jar -optype_assetbundle_enc -inputfile bundles.zip
```

You can also pass an AssetBundle directory. Use `-outputfile` to choose the output archive or directory.

Encrypt Unity DLLs

Use `-optype_dll_enc` for a single DLL or a ZIP archive containing Unity DLLs. This also supports DLLs distributed as HybridCLR hot updates.

```
java -jar JikGuard.jar -optype_dll_enc -inputfile HotUpdate.dll
java -jar JikGuard.jar -optype_dll_enc -inputfile assemblies.zip
```

The default output is a ZIP archive next to the input, such as `HotUpdate_jikguard_protected.zip` or `assemblies_jikguard_protected.zip`. Use `-outputfile` to choose another path.

Encrypt Cocos assets by file extension

In `config.ini`, add an `[encAsset]` section. List one extension per line and number the entries consecutively, starting at 1. Include the leading dot and **match the case of the filenames**. The following example selects JavaScript and PNG files.

```
[encAsset]
1=.js
2=.png
```

For a resource directory, specify a separate output directory so it is easy to locate the encrypted assets.

```
java -jar JikGuard.jar -optype_cocos_asset_enc -inputfile assets -outputfile assets-encrypted
```

For a ZIP archive, supply the archive as the input. This example creates `assets_jikguard_protected.zip` beside the original. Add `-outputfile` to choose a different ZIP file.

```
java -jar JikGuard.jar -optype_cocos_asset_enc -inputfile assets.zip
```