

Signature V2 Plugin External Document

version: 1.11.5

Disclaimer: this document is for the **PLUGIN** architecture

- [How to Read This Document?](#)
- [Client Parameters](#)
 - [Android](#)
 - [Extract your SHA1 Certificate Fingerprint](#)
 - [Extract your Internal App Sharing Certificate Fingerprint](#)
 - [Extract your Google Play App Signing Certificate Fingerprint](#)
 - [Extract your Huawei AppGallery App Signing Certificate Fingerprint](#)
 - [iOS](#)
- [Integration Guide](#)
 - [Android](#)
 - [iOS](#)
 - [Troubleshooting: Failed to verify bitcode Error](#)
 - [Troubleshooting: Code Signing failed](#)
 - [React Native](#)
 - [Cordova](#)
 - [Xamarin](#)
 - [Unity](#)
 - [Troubleshooting: "Unknown CPU Architecture in AdjustSigSdk.a" Error](#)
- [On Enforcing the SDK Signature in the Dashboard](#)
- [On Disabling/Enabling Signing For Testing](#)
- [Plugin -> Plugin Library Update Guide](#)
- [Verification](#)
 - [Step 1: Using sig_doctor](#)
 - [Usage](#)
 - [Step 2: Device Preparation](#)
 - [Step 3: App Generation](#)
 - [Step 4A: Native Android App](#)
 - [Step 4B: Native iOS App](#)
 - [Step 4C: React Native Android app](#)
 - [Step 4D: React Native iOS App](#)
 - [Step 4E: Unity Android App](#)
 - [Step 4F: Unity iOS App](#)
 - [Step 4G: Xamarin Android App](#)
 - [Step 4H: Cordova Android App](#)
 - [Step 4I: Cordova iOS App](#)
- [On Play Store Warnings](#)
- [Contact](#)

How to Read This Document?

- If you're a new client to this library:
 - i. Read the [Client Parameters](#) section
 - ii. Send those parameters to your Account Manager
 - iii. They will respond to you with the library
 - iv. Run through the [Integration Guide](#) section based on your platform

- v. Finally, verify your integration with the [Verification](#) section
- If you just want to integrate the library:
 - i. Read the [Integration Guide](#) section based on your platform
 - ii. Finally, verify your integration with the [Verification](#) section
- If you just want to verify the library's integration
 - i. Read the [Verification](#) section
- If you are an existing user of this library, and
 - i. Are using the plugin version of this library and wish to update to a newer version of the library, please refer to the [Plugin -> Plugin Library Update Guide](#)

Client Parameters

Android

To get started with the beta, Adjust needs three parameters from you. Apart from the app token, these parameters are public knowledge.

We require this information because every library is custom made; no two libraries are the same. Because of this, the library needs to perform some checks on the basic package and certificate fingerprint to be absolutely sure that the library is running on the app it was designed for.

Google servers, along with any device that downloads the app, will check your app's signing certificate fingerprint.

The three parameters we need are:

- Package name (to locate, look at the Google Play Store page link of your app)
 - Should look similar to 'com.adjust.sdk'
- Adjust app token
 - Provided by Adjust and should be easy for developer to find
- SHA1 Signing certificate fingerprint(s) (use the *keytool* command line tool shipped with the Android SDK from Google)

Extract your SHA1 Certificate Fingerprint

Find the .jks keystore file you use to sign the release version of your app. It is very important to use the **same** keystore you would use to publish the **release** version of your app to the Google Play Store.

Run the following command:

```
$ keytool -list -v -keystore <location/of/your/key.jks> -alias YOUR_KEY_ALIAS
```

Make sure to replace the information in the `<>` brackets with your *own* information. The developer should know the keystore alias, the location of the keystore and the keystore password. The above command will prompt you to enter your keystore password; make sure to have it handy.

The command output should look similar to this:

```
Alias name: Key0
Creation date: May 15, 2018
Entry type: PrivateKeyEntry
```

```
Certificate chain length: 1
Certificate[1]:
Owner: CN=aaaa
Issuer: CN=aaaa
Serial number: 642f1b62
Valid from: Tue May 15 09:46:06 CEST 2018 until: Sat May 09 09:46:06 CEST 2043
Certificate fingerprints:
MD5: E7:88:9F:8C:9D:F4:14:C1:CF:E8:4C:97:F3:F2:3A:E3
SHA1: C4:BD:07:91:BC:09:F8:B6:15:CD:BC:A3:3F:BC:68:8B:C2:EF:4F:F5
SHA256: 55:FB:97:0F:46:0F:94:EC:07:EA:01:69:50:5A:20:3F:A0:91:60:A4:F1:33:58:EA:76:DC:54:9E:A7:6A:B9:1A
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

Please send us the SHA1 fingerprint. In the code snippet above, it is:

```
C4:BD:07:91:BC:09:F8:B6:15:CD:BC:A3:3F:BC:68:8B:C2:EF:4F:F5 .
```

Your output **will** vary.

If you are using [Google Play Internal App Sharing](#), read the relevant section below.

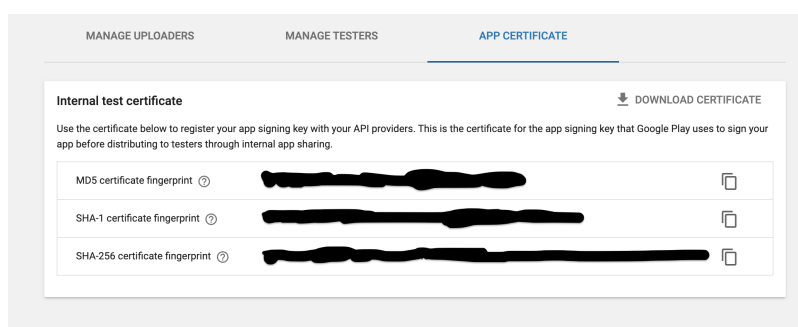
If you are using [Google Play App Signing](#), read the relevant section below.

Extract your Internal App Sharing Certificate Fingerprint

If you are using [Google Play Internal App Sharing](#), we'll need both **your organization's SHA-1 certificate fingerprint (see section above)** and your **“Internal test certificate fingerprint”** to make sure the integration is successful during testing and production.

Please complete the following steps to extract the fingerprints:

1. Navigate to the Google Play Console and log in
2. Select the application to sign
3. Select Release Management → App Releases → Manage internal app sharing → App certificates
4. Copy the SHA-1 certificate fingerprint from there, including the one from your organization's keystore that you used to sign your app



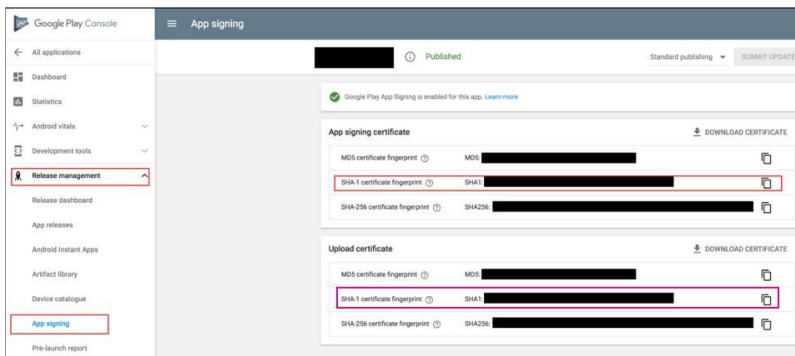
Extract your Google Play App Signing Certificate Fingerprint

If you are using [Google Play App Signing](#), we'll need both **your organization's SHA-1 certificate fingerprint (see relevant section above)** and your **“App signing certificate fingerprint”** to make sure the integration is successful during testing and production.

Please complete the following steps to extract the fingerprints:

1. Navigate to the Google Play Console and log in
2. Select the application to sign

3. Select Release Management → App Signing
4. Copy the SHA-1 certificate fingerprint from **both** sections



Extract your Huawei AppGallery App Signing Certificate Fingerprint

If you are using [Huawei AppGallery App Signing](#), we'll need both **your organization's SHA-1 certificate fingerprint** (see **relevant section above**) and your **“App signing certificate fingerprint”** to make sure the integration is successful during testing and production.

You'll find the SHA-1 app signing certificate fingerprint if you follow the steps outlined in "Question 6" of this link from Huawei's FAQ: https://developer.huawei.com/consumer/en/doc/development/AppGallery-connect-Guides/agc-app_bundle_faq

iOS

The parameters we'll need are:

- Adjust app token
- Bundle ID

Integration Guide

Our new SDK Signature solution – Signature V2 – is a drop-in plugin compatible with our public SDKs. Signature V2 is easier to implement and offers more security than its predecessor; here are the most significant changes:

- It contains encryption algorithms which secure communication with our servers.
- We utilize fortified obfuscation to combat network sniffers and analysis.
- Each library is customized; secrets and randomized encryption parameters are unique to your app and your app alone.
- We use public key encryption to prevent malicious tampering.

Signature V2 **effectively deprecates** Signature V1 (our 'app secret' solution). For this reason, we **strongly recommend** that if you have used the app secrets solution, you start transitioning to this solution and, **when you are sure that the vast majority of your traffic is from Signature V2** (two weeks after full roll out of the new app version to the store), you can deactivate your Signature V1 secrets from the dashboard.

NOTE: As with Signature V1, if you are utilising Adjust's FPS, you may still receive information on rejections with Signature V2 through callbacks.

NOTE: Prior to v4.21.1 (for all platforms), **Signature V2 library was bundled with the Adjust SDK**. This meant that the development team could not fetch the latest updates from the SDK and would have to wait until Adjust released a Signature V2 update including the necessary changes.

Android

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **not** function without them):

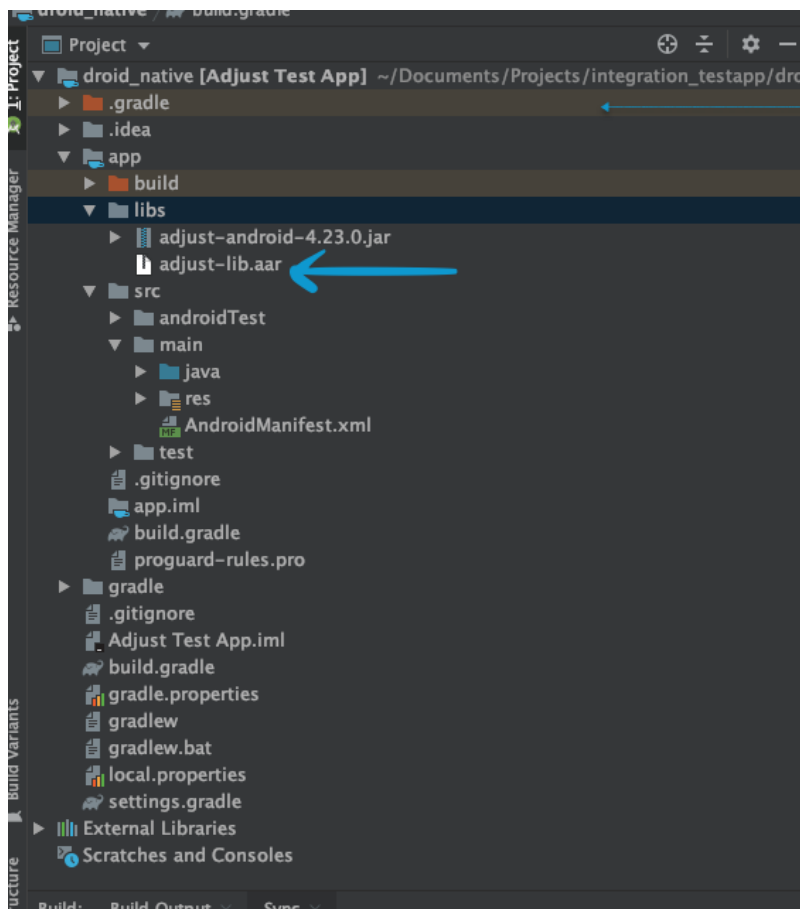
- Android API >= 18

NOTE: For the library to function Android Adjust SDK >= 4.21.1 is needed.

NOTE: If you are using ProGuard, you must use exactly the same Proguard configuration for Signature V2 as you use for the Adjust SDK.

To integrate the library:

1. Create a new **libs** directory inside your app module directory.
2. Copy the provided AAR library under the created **libs** directory.
 - **NOTE:** We'll use the name `adjust-lib.aar` to refer to your custom library for the rest of this section -- kindly replace with the name used in the deliverable you received.



3. Open your app-level `build.gradle` file and add the following, in their respective sections:

```
android {  
    defaultConfig {
```

```

        ndk.abiFilters 'armeabi-v7a', 'arm64-v8a', 'x86', 'x86_64'
    }
}

dependencies {
    implementation files('libs/adjust-lib.aar')
}

```

4. Click `sync project with Gradle files`.

iOS

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

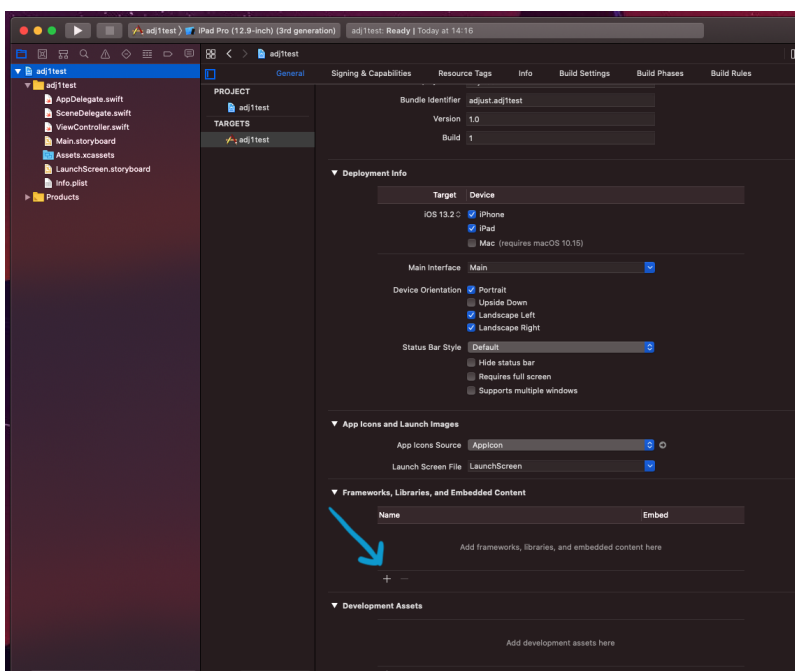
NOTE: For the library to function iOS Adjust SDK >= 4.21.1 is needed.

Dynamic Framework Integration

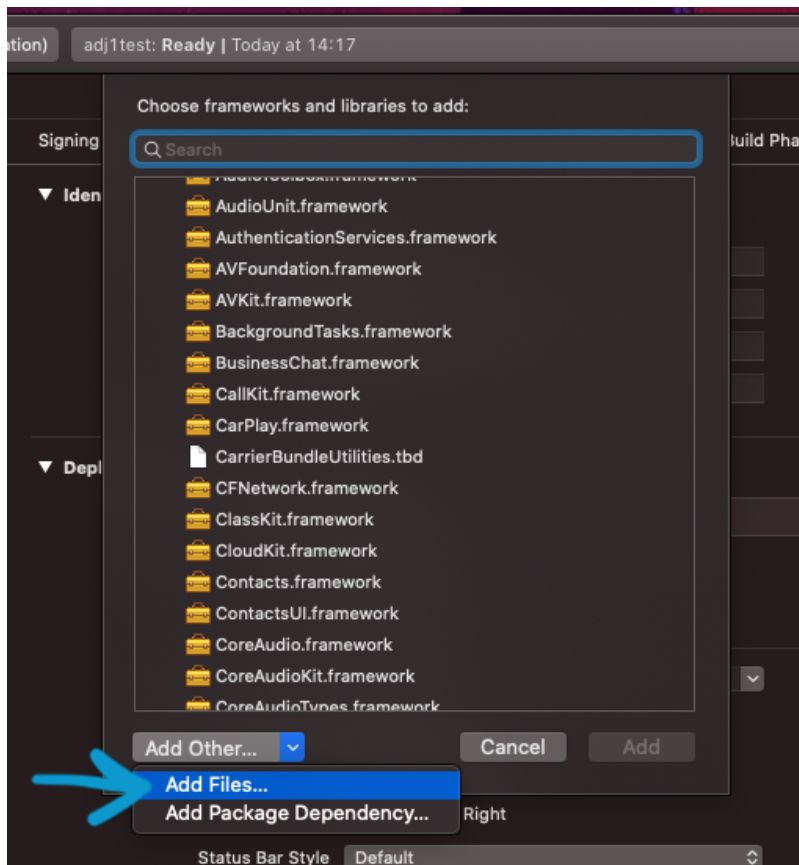
NOTE: We supply both a dynamic universal framework and a dynamic XCFramework. The steps to integrate **either are identical**. For documentation simplicity, I'll refer to the usage of the XCFramework below.

NOTE: We'll use the name `AdjustSigSdk` to refer to your custom library for the rest of this section -- kindly replace with the name used in the deliverable you received.

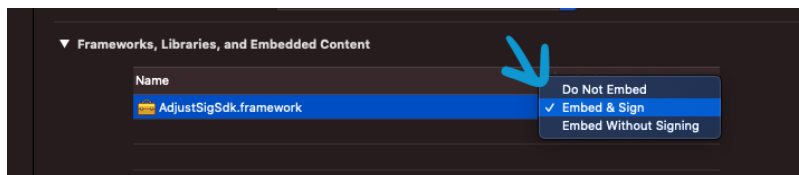
1. Copy the dynamic `AdjustSigSdk.xcframework` file to your project's directory
 2. In Xcode, select your project in the Project Navigator
 3. In the left-hand side of the main view, select your target
- For XCode >= 11:
 - i. Go to the `General` tab, expand the `Frameworks, Libraries and embedded Content` group.
 - ii. At the bottom of that section, select the `+` button



3. Click `Add Other > Add Files` , navigate to where you've put `AdjustSigSdk.xcframework` in your project and select it.



4. After adding, make sure to select `Embed & Sign` for `AdjustSigSdk.xcframework` .



- For XCode < 11:

NOTE: You can only use the universal dynamic framework for XCode < 11, not the XCFramework.

- In the `Build Phases` tab, expand the `Link Binary with Libraries` group
- At the bottom of that section, select the `+` button.
- Add `AdjustSigSdk.framework` and set it as "optional". Make sure to choose `Add Other...` and select the `AdjustSigSdk.framework` file, not the symbolic link provided by Xcode's framework selection pop-up.
- Go to the `General` tab, expand the `Embedded Binaries` group.
- If the library is not already there, click `Add Other...` and select `AdjustSigSdk.framework`. Make sure to choose `Add Other...` and select the `AdjustSigSdk.framework` file, not the symbolic link provided by Xcode's framework selection pop-up.
- You'll see a dialog with the title `Choose options for adding these files` , make sure to tick `Copy items if needed` . This will also add `AdjustSigSdk.framework` to the project navigator.

Static Framework Integration

We don't recommend you use static frameworks since a dynamic one is much better and lighter on your app. That being said, if you use the static framework, be sure to include `-force_load`

`$(PROJECT_DIR)/$(PROJECT_NAME)/AdjustSigSdk.framework/AdjustSigSdk` in your XCode project, under "Other Linker Flags".
If you don't know how to do this, please don't use a static framework and use a dynamic framework (instructions above).

Troubleshooting: Failed to verify bitcode Error

If you get an error such as the one below:

```
Failed to verify bitcode in
AdjustSigSdk.framework/AdjustSigSdk:
error: Cannot extract bundle from
/var/folders/qy/y1q4b9750lg6x21scz02hq5c0000gn/T/IDEDistributionOptionThinning.Xnl/Payload/AdjustExample-iOS.
```

You are using a dynamic *non* xcframework and will need to strip the binaries for the `i386` and `x86_64` architectures:

1. Select your project in the Project Navigator
2. In the left-hand side of the main view, select your target
3. Go to the `Build Phases` tab, press the `+` button and choose `New Run Script Phase`
4. A new Run Script will appear; name it "Strip Adjust Framework" and place it **below** your "Embed Frameworks" tab in "Build Phases"
5. Copy the code snippet in [this link](#) to the input area
6. Clean and rebuild

Alternatively, and if your build process allows, kindly consider switching to the dynamic xcframework.

Troubleshooting: Code Signing failed

This can happen if you included the "Strip Adjust Framework" run script (see section above) and misplaced the "Embed Frameworks" tab in "Build Phases".

The correct order of "Build Phases" should look like this (other phases can go in the middle of those):

```
Dependencies
Run Script
Compile Sources
Link Binary with Libraries
Copy Bundle Resources
Embed Pods Frameworks
Embed Frameworks
Strip Adjust Framework
```

If you added the above "Strip Adjust Framework" run script to your build phases, you should place "Embed Frameworks"

React Native

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **not** function without them):

- If you're using Android

- Android API ≥ 18
 - Android Adjust SDK $\geq 4.21.1$
- If you're using iOS
 - iOS Adjust SDK $\geq 4.21.1$

To integrate Signature V2 for React Native, first integrate the Adjust SDK following the steps in the Adjust SDK documentation. Signature V2 requires the Adjust SDK to be present.

Once the Adjust SDK is integrated, you can integrate Signature V2 for React Native by following the native [Android](#) and [iOS](#) integration steps listed previously in this document.

To be very clear: for this library, we only supply the native libraries, not a react native package, since the latter is **not needed**. There are no client-facing API methods for this library. Adjust SDK knows what to do with the native library if it finds it in the app, without any needed addition to client code.

Cordova

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **not** function without them):

- If you're using Android
 - Android API ≥ 18
 - Android Adjust SDK $\geq 4.21.1$
- If you're using iOS
 - iOS Adjust SDK $\geq 4.21.1$

NOTE: If you are using ProGuard, you must use exactly the same Proguard configuration for Signature V2 as you use for the Adjust SDK.

As for the actual integration steps:

1. Make a directory in your project and call it `ext`
2. Unzip the plugin library there, you should now have one directory inside `ext`, which is called `cordova-adjust-sig`
3. Run `cordova plugin add ext/cordova-adjust-sig`. Now, download the Adjust SDK through `cordova plugin add com.adjust.sdk`, or by any means through the [public README](#).
4. Open the iOS Xcode project in `platforms/ios` with Xcode. Select your project in the Project Navigator. In the left hand side of the main view, select your project.
5. Click the tab `Build Settings`
6. Double click `Other Linker Flags`
7. Click `+`
8. Paste the following: `-force_load`
`$(PROJECT_DIR)/$(PROJECT_NAME)/Plugins/com.adjust.sdk.sig/AdjustSigSdk.framework/AdjustSigSdk`

Xamarin

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **not** function without them):

- If you're using Android
 - Android API ≥ 18
 - Android Adjust SDK $\geq 4.21.1$
- If you're using iOS
 - iOS Adjust SDK $\geq 4.21.1$

NOTE: If you are using ProGuard, you must use exactly the same Proguard configuration for Signature V2 as you use for the Adjust SDK.

As for the actual integration steps:

- Android:
 - i. Copy the received `AdjustSigSdk.Xamarin.Android.dll` file under your Android project (e.g. `MyProject/MyProject.Android/AdjustSigSdk.Xamarin.Android.dll`)
 - ii. In the Solution Explorer, right-click on the project name and select `Add > Reference...`
 - iii. In the `.Net Assembly` tab, click `Browse...` to select the copied file and click `OK`
- iOS:
 - i. Copy the received `libAdjustSigSdk.iOS.a` file under your iOS project (e.g. `MyProject/MyProject.iOS/libAdjustSigSdk.iOS.a`)
 - ii. In the Solution Explorer, right-click on the project name and select `Add > Add Native Reference` to select the copied file
 - iii. Under `Native References`, right-click `libAdjustSigSdk.iOS` and select `Properties`
 - Tick `Force Load`
 - Add to `Linker Flags` the following `-L$(ProjectDir)`

Unity

Signature V2 is non-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **not** function without them):

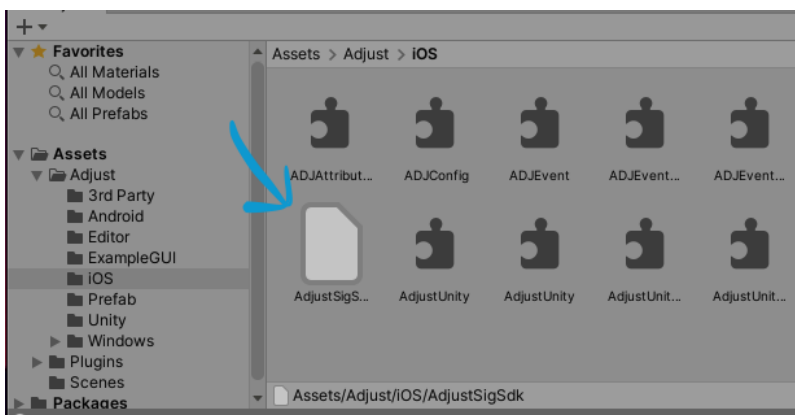
- If you're using Android
 - Android API ≥ 18
 - Android Adjust SDK $\geq 4.21.1$
- If you're using iOS
 - iOS Adjust SDK $\geq 4.21.1$ (recommended $\geq 4.32.1$, which fixes common integration issues)

NOTE: If you are using ProGuard, you must use exactly the same Proguard configuration for Signature V2 as you use for the Adjust SDK.

As for the actual integration steps:

1. Drop the `.aar` file in `Assets/Adjust/Android`
2. Drop the `.a` file in `Assets/Adjust/iOS`
3. If you're running Adjust Unity SDK 4.23.1 and above, the integrated `post-build process` will take care of the rest.
4. If you're running Adjust Unity SDK **lower** than 4.23.1, do the following:

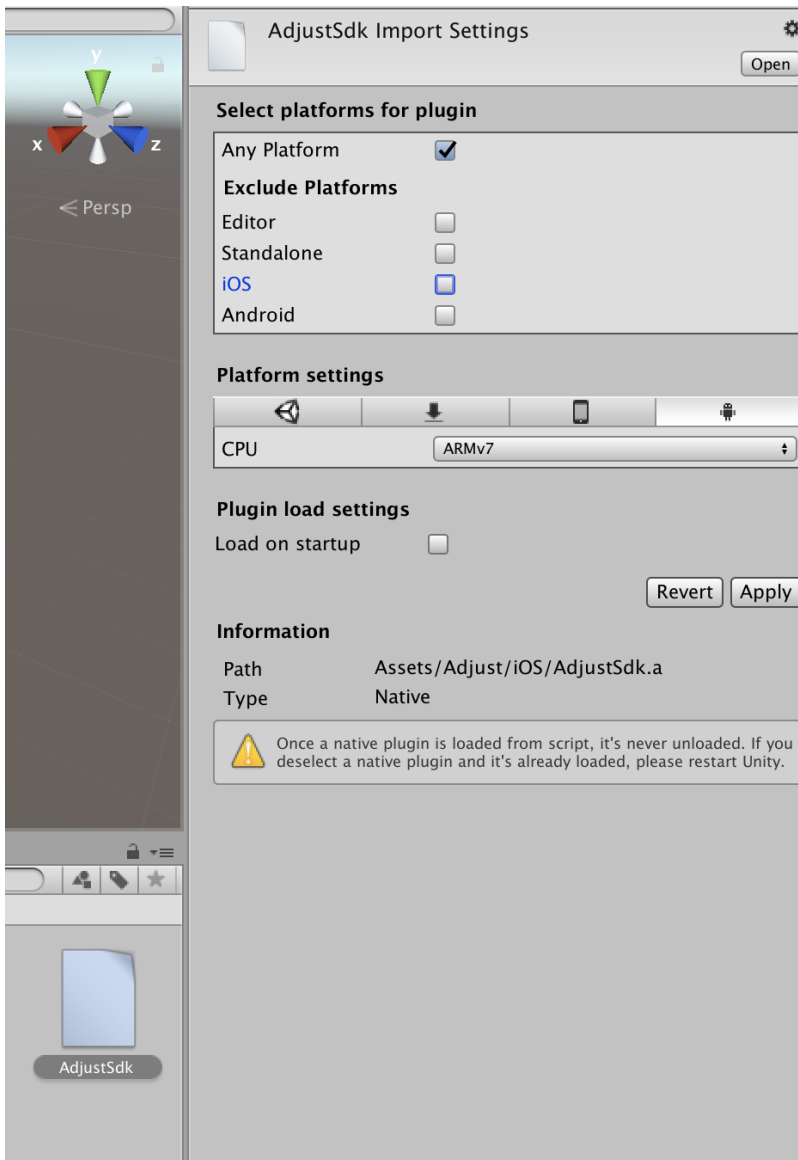
- i. Open your Unity iOS Xcode project with Xcode.
- ii. Select your project in the Project Navigator.
- iii. In the left hand side of the main view, select your project.
 - Click the tab `Build Settings`
 - Double click `Other Linker Flags`
 - Click `+`
 - Paste the following: `-force_load $(PROJECT_DIR)/Libraries/Adjust/iOS/AdjustSigSdk.a`



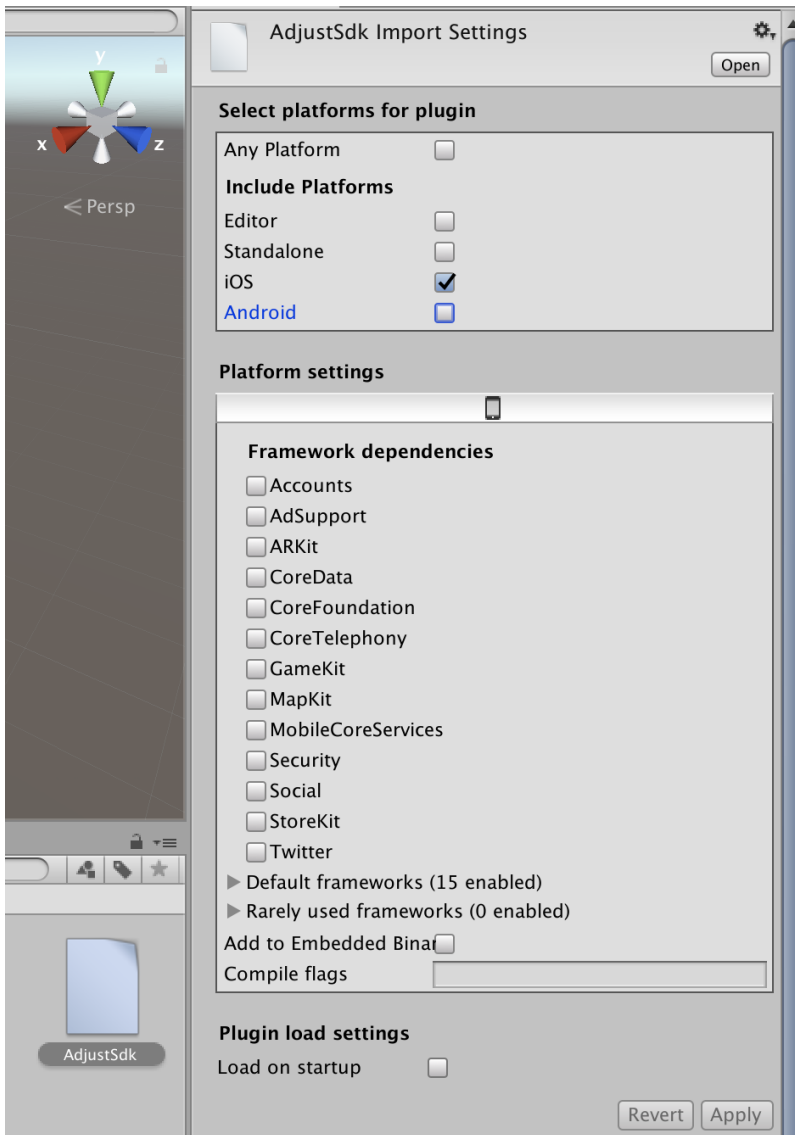
Troubleshooting: “Unknown CPU Architecture in AdjustSigSdk.a” Error

Unity Editor 2018 and above sometimes reads the Adjust iOS static library `AdjustSigSdk.a` as supported by “all architectures” and not just for `iOS`.

- Select the `AdjustSigSdk.a` file in `Assets/Adjust/iOS` as shown below:



- Now, toggle off `Any Platform` and select `iOS` platform as shown below:



On Enforcing the SDK Signature in the Dashboard

See [this page](#), please.

NOTE: While the signature is not "enforced" within the dashboard settings, all installs will be accepted, including those that carry no signature or an invalid signature. Once you "enforce" the Signature, Adjust servers will start rejecting all installs that do not carry a valid signature. Therefore, it's crucial to ensure that the `Enforce toggle` is set to `ON` once the vast majority of your incoming install requests originate from the app version containing Signature V2.

On Disabling Signing For Testing

Signing is enabled by default. That being said, sometimes during unit tests, you might want to **disable** signing inside your test suite. If you disable signing, your Adjust requests will not be signed. Below is how you can do that.

To repeat, please make sure to **only use this feature in your test suite** and not in production.

IMPORTANT NOTE: `Enforce SDK Signature` toggle in Adjust Dashboard **does affect Signature V2**. If `disableSigning()` is called and `Enforce SDK Signature` is toggled **ON**, **all traffic will be rejected** regardless of whether the environment is set to

`SANDBOX` or otherwise.

- Android:

```
// Make sure you're on SANDBOX mode when testing
AdjustConfig config = new AdjustConfig(this, appToken, AdjustConfig.ENVIRONMENT_SANDBOX);
AdjustFactory.disableSigning()
```

- iOS:

```
// Make sure you're in SANDBOX mode when testing
ADJConfig*adjustConfig = [ADJConfig configWithAppToken:yourAppToken environment:ADJEnvironmentSandbox];
[ADJAdjustFactory disableSigning]
```

Plugin -> Plugin Library Update Guide

Upon wishing to update to a newer version of the Signature V2 library (updates recommended every six (6) months), kindly do the below:

1. Request a new Signature V2 library for the frameworks and platforms interested from your Account Manager
2. Upon receiving the new Signature V2 library, kindly consult the documentation bundled with the Signature V2 library
3. To avoid any issues, *fully* remove the previous Signature V2 library from your app
4. Follow the integration and verification guide as outlined in the attached documentation of the new Signature V2 library received

Verification

Testing the new SDK solution is a bit different than testing the regular Adjust SDK. Please follow the steps below to confirm that the integration was successful.

Step 1: Using `sig_doctor`

`sig_doctor` is a standalone tool to verify a successful Signature V2 integration. It is bundled in the Signature V2 zip file that you received.

There's a `sig_doctor` binary for all three major operating systems (Windows, MacOS, and Linux). Furthermore, it is meant to test both iOS IPAs/frameworks and Android APKs/AARs, for all supported cross-platforms as well (Unity, Cordova, etc.)

Dependencies

The binary is compiled statically, so there're no dependencies.

Usage

- For Android: Build a **release** APK, made with your release keystore.
 - This should be the same exact build you would send to the app store, right before publishing.
 - If you're using *Android App Bundles*, the process is identical to that of generating an APK. Please test with your release APK, not AAB. It is furthermore possible to use `bundletool` to generate an APK out of an AAB file, while supplying `--mode=universal` also as an argument.

- For iOS: Build an IPA (doesn't matter if it is `development` , `adhoc` or `enterprise`).
- Double-click on the `adjust_sig_doctor` binary based on your platform (Windows, OSX, Linux).
- Follow the instructions in the dialog box.

Using the CLI

`sig_doctor` is also a CLI tool. Just run it with `-h` to show the help.

Step 2: Device Preparation

1. Use a physical device and not an emulator or simulator
2. Fully delete the app from the device
3. Make sure the device is "forgotten". You can do that either by using Testing Console or by visiting the following URL in a browser:
 - For Android: `http://app.adjust.com/forget_device?app_token={yourAppToken}&gps_adid={gpsAdidValue}`
 - Replace `yourAppToken` and `gps_adid` accordingly
 - For iOS: `http://app.adjust.com/forget_device?app_token={yourAppToken}&idfa={idfaValue}`
 - Replace `yourAppToken` and `idfaValue` accordingly
4. Connect the device to your development machine
5. Lastly, and for testing purposes, the device's Advertising ID is needed. To retrieve it, the Adjust Insights app ([Android](#) | [iOS](#)) can be used to extract said identifier

Step 3: App Generation

NOTE: Using the library in a debug environment (inside either Android Studio or Xcode) will trigger the library's detection mechanism and will flag the install as 'untrusted'. You **must** conduct the test with a packaged app. For Android, this means signing the APK with a release keystore, then installing it on a real device. For iOS, this means archiving the project and generating a 'Development' IPA, which you should install on a real device.

Step 4A: Native Android App

- Build a release version of the app and sign it with the same keystore used to generate the SHA1 certificate fingerprint you sent to Adjust.
- Install your app through ADB on the device you prepared (see above).
- Run it so that an install is sent to Adjust servers.
- Use [Testing Console](#) along with your device's Google Advertising ID or IDFA to validate that `SignatureVerificationResult` is `Valid Signature`

NOTE: If the `SignatureVerificationResult` is anything but 'Valid Signature', please do not publish the app. Contact your Account Manager / Sales Engineer. We'll need to take a look at your integration and investigate further.

NOTE: Using the library in a debug environment (inside either Android Studio or Xcode) will trigger the library's detection mechanism and will flag the install as 'untrusted'. You **must** conduct the test with a packaged app. For Android, this means signing the APK with a release keystore, then installing it on a real device. For iOS, this means archiving the project and generating a 'development' IPA, which you should install on a real device.

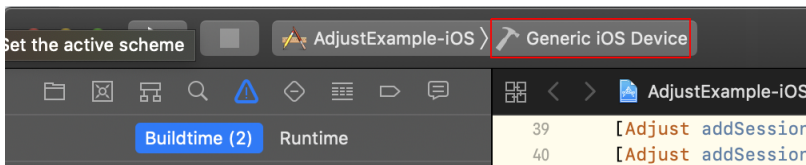
Step 4B: Native iOS App

The steps required are:

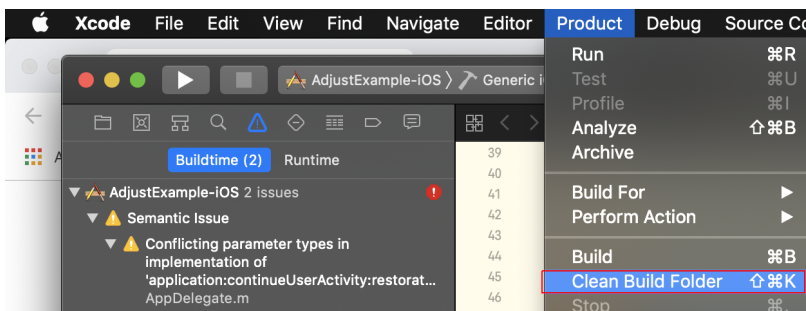
1. Archive a development version of your app as an IPA file
2. Sideload your archived app through Xcode

In detail:

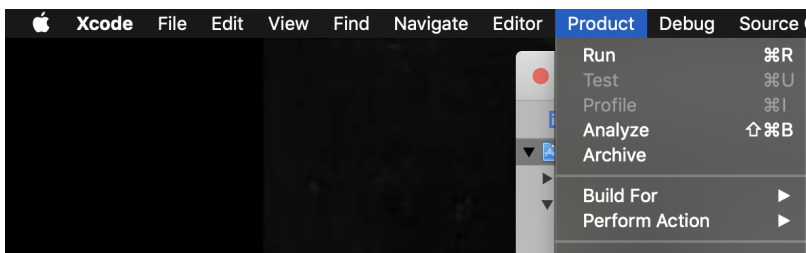
- Select `Generic iOS Device` from the device selection drop-down menu (you'll find it to the right of the debug and stop icons in the main toolbar).



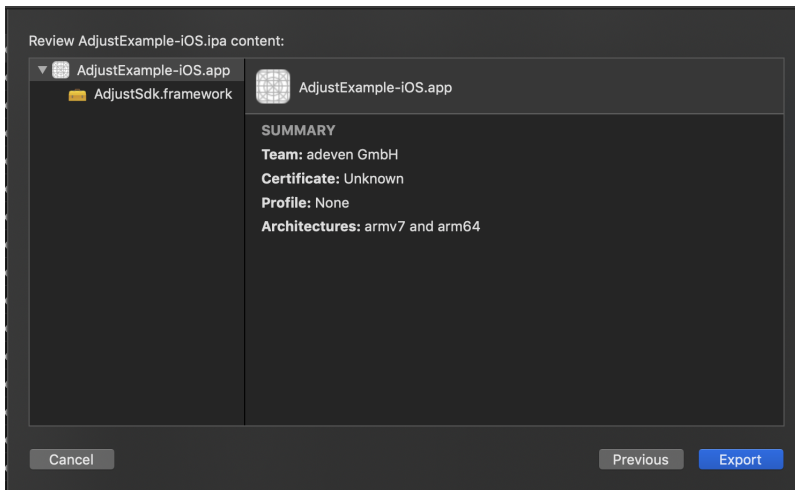
- Click `Product` → `Clean Build Folder` (or `Clean`) in your Xcode toolbar.



- Click `Product` → `Archive` in your Xcode toolbar.



- Select the first entry and:
 - If you have Xcode < 10, click `Export`
 - If you have Xcode \geq 10, click `Distribute App`
 - Choose `Development` and continue
 - **NOTE:** if you receive a *Failed to verify bitcode* error, please refer to this [troubleshooting section](#) in this document in the “iOS Integration Guide” section.
 - You'll see the Development distribution options dialog menu; keep all of the options as-is, then click `Next` on this and the following menu
 - You should receive a pop-up like this:



- Click `Export` and save the archive
- Insert a physical iOS device through your USB port
- Click `Windows` → `Devices and Simulators` in the Xcode toolbar
- You should see your test device; click the `+` icon at the bottom of 'Installed Apps' and select the `.ipa` file from the directory you archived
- Congratulations, you sideloaded your app through Xcode! Run the app from the physical iOS device and check `Test Console` with the device's IDFA to find out if the Signature V2 integration was successful.
- You can also view the device logs, either from the `Devices and Simulators` window or through MacOS's Console app, which comes pre-installed with MacOS.

Use [Testing Console](#) along with your device's Google Advertising ID or IDFA to validate that `SignatureVerificationResult` is `Valid Signature`

NOTE: If the `SignatureVerificationResult` is anything but 'Valid Signature', please do not publish the app. Contact your Account Manager / Sales Engineer. We'll need to take a look at your integration and investigate further.

NOTE: Using the library in a debug environment (inside either Android Studio or Xcode) will trigger the library's detection mechanism and will flag the install as 'untrusted'. You **must** conduct the test with a packaged app. For Android, this means signing the APK with a release keystore, then installing it on a real device. For iOS, this means archiving the project and generating a 'development' IPA, which you should install on a real device.

Step 4C: React Native Android app

Follow the same steps as in [Section 4A](#). You must also run `react-native bundle` in order to build the Javascript layer for offline use. Run the following command and substitute the `--entry-file` and `--assets-dest` parameters with their counterparts in your project:

```
$ react-native bundle --platform android --dev false --entry-file
index.js --bundle-output
android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res
```

The final output should look similar to this:

```
[1] % react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index
.android.bundle --assets-dest android/app/src/main/res
Loading dependency graph, done.
Loading dependency graph... bundle: Writing bundle output to: android/app/src/main/assets/index.android.bundle
bundle: Done writing bundle output
```

NOTE: If the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact your Account Manager. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's Google Advertising ID.

Step 4D: React Native iOS App

Follow the same steps as in [Section 4B](#).

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's IDFA.

Step 4E: Unity Android App

Follow the same steps as in [Section 4A](#).

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's Google Advertising ID.

Step 4F: Unity iOS App

Follow the same steps as in [Section 4B](#).

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's IDFA.

Step 4G: Xamarin Android App

Follow the same steps as in [Section 4A](#).

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's Google Advertising ID.

Step 4H: Cordova Android App

The same steps in [Section 4A](#) apply.

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's Google Advertising ID.

Step 4I: Cordova iOS App

Follow the same steps as in [Section 4B](#) *after* you:

- Build in Cordova
- Import Build in Xcode

NOTE: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Mobile Security Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app ([Android](#) | [iOS](#)) to extract the device's IDFA.

On Play Store Warnings

Upon uploading your app to the Play Store, you may encounter the following warning:

This App Bundle contains native code, and you've not uploaded debug symbols. We recommend you upload a symbol file to make your crashes and ANRs easier to analyze and debug.

The lack of debug symbols is intended, and the warning may be safely ignored.

Contact

If you have any issues, please contact your Account Manager. They will help you immediately debug any issues.

If you have any suggestions on improving the testing process, please contact your Account Manager as well. We truly appreciate any comments that would make life easier for both of us.

– Adjust Mobile Security Team