|  |  |  |
| --- | --- | --- |
|  | **2, LEBUH ACHEH, GEORGE TOWN**  **10300 GEORGE TOWN**  **PULAU PINANG**  **INFORMATION SHEET** | |
| **PROGRAM’S CODE & NAME** | J620-002-4:2020 FRONT-END SOFTWARE DEVELOPMENT | |
| **LEVEL** | FOUR (4) | |
| **COMPETENCY UNIT NO. AND TITLE** | J620-002-4:2020-C04: MOBILE APPLICATION WITH THIRD PARTY API DEVELOPMENT | |
| **WORK ACTIVITIES NO. AND STATEMENT** | 1. CREATE MOBILE APP DESIGN MOCK-UP ELEMENTS. 2. PLAN MOBILE APP DESIGN STRUCTURE. 3. TRANSFORM MOCK-UP TO MOBILE APP. 4. INTEGRATE MOBILE APP WITH DATA SOURCE. 5. VERIFY SUCCESSFUL API INTEGRATION 6. **VERIFY DEVELOPED MOBILE APP.** 7. VERIFY MOBILE APP ACCESSIBLE GLOBALLY. | |
| **CODE NO.** | J620-002-4:2020-C04/IS(13/15) | Page: 1 of22 |

**TITLE**:

**APPLICATION SIGNING AND PACKAGE**

**PURPOSE**:

This information sheet is intended to provide insight and knowledge to trainees with regards to the fundamentals of application signing and package.

**INFORMATION:**

This information sheet provides useful notes and explanations on fundamental concepts of application signing and package.

# **APPLICATION SIGNING**

Android requires that all APKs be digitally signed with a certificate before they are installed on a device or updated. If you use Android App Bundles, you need to sign only your app bundle before you upload it to the Play Console and Play App Signing takes care of the rest. However, you can also manually sign your app for upload to Google Play and other app stores.

The following is a high-level overview of the steps you might need to take to sign and publish a new app to Google Play:

* Generate an upload key and keystore
* Sign your app with your upload key
* Opt in to Play App Signing
* Upload your app to Google Play
* Prepare & roll out release of your app

If instead your app is already published to the Google Play Store with an existing app signing key, or you would like to choose the app signing key for a new app instead of having Google generate it, follow these steps:

* Sign your app with your app’s signing key and select the option to encrypt and export its signing key.
* Upload your app’s signing key to opt in to Play App Signing.
* (Recommended) Generate and register an upload certificate for future updates to your app
* Upload your app to Google Play
* Prepare & roll out release of your app

## Play App Signing

With Play App Signing, Google manages and protects your app's signing key for you and uses it to sign your APKs for distribution. And, because app bundles defer building and signing APKs to the Google Play Store, you need to opt in to Play App Signing before you upload your app bundle. Doing so lets you benefit from the following:

* Use the Android App Bundle and support Google Play’s advanced delivery modes. The Android App Bundle makes your app much smaller, your releases simpler, and makes it possible to use feature modules and offer instant experiences.
* Increase the security of your signing key and make it possible to use a separate upload key to sign the app bundle you upload to Google Play.

Opting in to Play App Signing applies for the lifetime of your app. In order to ensure security, after you opt in you cannot retrieve a copy of your app's signing key and you cannot delete it from Google's servers without deleting your app.

Play App Signing uses two keys: the app signing key and the upload key, which are described in further detail in the section about Keys and keystores. You keep the upload key and use it to sign your app for upload to the Google Play Store. By using a separate upload key, you can request an upload key reset if your key is ever lost or compromised. By comparison, if you’re not opted into app signing with by Google Play and you lose your app’s signing key, you lose the ability to update your app.

When you are ready to publish, you can sign your app using Android Studio upload it to Google Play. The key with which you sign your app becomes your app’s upload key. Google uses the upload certificate to verify your identity and signs your APK(s) with your app signing key for distribution as shown in figure 1.

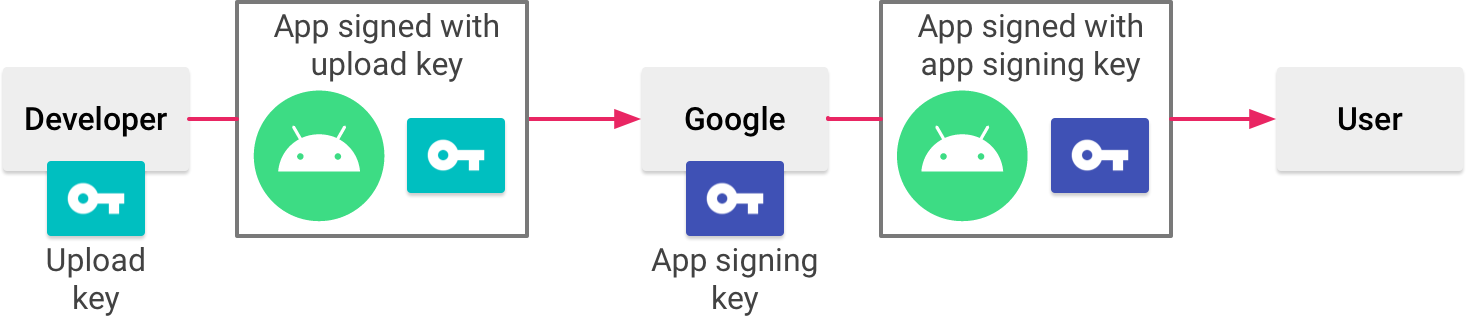


Figure 1. Signing an app with Play App Signing

If you do not already have an app signing key, you can generate one during the sign-up process. When you opt in to Play App Signing, you aren’t able to download the signing key from Google. If you want to use the same signing key across multiple stores, make sure to provide your own signing key when you opt in to Play App Signing, instead of having Google generate one for you.

Your keys are stored on the same infrastructure that Google uses to store its own keys, where they are protected by Google’s Key Management Service. You can learn more about Google’s technical infrastructure by reading the Google Cloud Security Whitepapers.

When you use Play App Signing, if you lose your upload key, or if it is compromised, you can contact Google to revoke your old upload key and generate a new one. Because your app signing key is secured by Google, you can continue to upload new versions of your app as updates to the original app, even if you change upload keys. To learn more, read Reset a lost or compromised private upload key.

## b. Keys, certificates, and keystores

A public key certificate (.der or .pem files), also known as a digital certificate or an identity certificate, contains the public key of a public/private key pair, as well as some other metadata identifying the owner (for example, name and location) who holds the corresponding private key.

When signing your app, the signing tool attaches the certificate to your app. The certificate associates the APK or app bundle to you and your corresponding private key. This helps Android ensure that any future updates to your app are authentic and come from the original author. The key used to create this certificate is called the app signing key.

You can download the certificate for your app signing key and your upload key from the app signing page in the Play Console in order to register your key(s) with API providers. The certificate can be shared with anyone. It does not contain your private key. Every app must use the same certificate throughout its lifespan in order for users to be able to install new versions as updates to the app.

A certificate fingerprint is a short and unique representation of a certificate that is often requested by API providers alongside the package name to register an app to use their service. The MD5, SHA-1 and SHA-256 fingerprints of the upload and app signing certificates can be found on the app signing page of the Play Console. Other fingerprints can also be computed by downloading the original certificate (.der) from the same page.

Table 1 shows the different types of keys and keystores you should understand:

|  |  |
| --- | --- |
| App signing key | The key that is used to sign APKs that are installed on a user's device. As part of Android’s secure update model, the signing key never changes during the lifetime of your app. The app signing key is private and must be kept secret. You can, however, share the certificate that is generated using your app signing key. |
| Upload key | The key you use to sign the app bundle or APK before you upload it for app signing with Google Play. You must keep the upload key secret. However, you can share the certificate that is generated using your upload key. You may generate an upload key in one of the following ways:   * If you choose for Google to generate the app signing key for you when you opt in, then the key you use to sign your app for release is designated as your upload key. * If you provide the app signing key to Google when opting in your new or existing app, then you have the option to generate a new upload key during or after opting in for increased security. * If you do not generate a new upload key, you continue to use your app signing key as your upload key to sign each release. |
| Java keystore (.jks or .keystore) | A binary file that serves as a repository of certificates and private keys. |
| Play Encrypt Private Key (PEPK) tool | Use this tool to export private keys from a Java Keystore and encrypt them for transfer to Google Play. When providing the app signing key for Google to use, select the option to Export and upload a key from a Java keystore and follow the instructions to download and use the tool. Alternatively, select the option to Export and upload a key (not using a Java keystore) to download, review, and use the PEPK tool’s open-source code. |

Table 1. Types of keys and keystores

## c. Sign your debug build

When running or debugging your project from the IDE, Android Studio automatically signs your app with a debug certificate generated by the Android SDK tools. The first time you run or debug your project in Android Studio, the IDE automatically creates the debug keystore and certificate in $HOME/.android/debug.keystore, and sets the keystore and key passwords.

Because the debug certificate is created by the build tools and is insecure by design, most app stores (including the Google Play Store) do not accept apps signed with a debug certificate for publishing.

Android Studio automatically stores your debug signing information in a signing configuration, so you do not have to enter it every time you debug. A signing configuration is an object consisting of all of the necessary information to sign your app, including the keystore location, keystore password, key name, and key password. You cannot directly edit the debug signing configuration, but you can configure how you sign your release build.

## d. Generate an upload key and keystore

When you are ready to publish your app, you need to sign your app and upload it to an app store, such as Google Play. When publishing your app to Google Play, you should also opt in to Play App Signing. If you don't already have an upload key, which is useful when opting in to Play App Signing, you can generate one using Android Studio as follows:

1. In the menu bar, click Build > Generate Signed Bundle/APK.
2. In the Generate Signed Bundle or APK dialog, select Android App Bundle or APK and click Next.
3. Below the field for Key store path, click Create new.
4. On the New Key Store window, provide the information for your keystore and key, as shown in figure 2.
5. Once you complete the form, click OK.

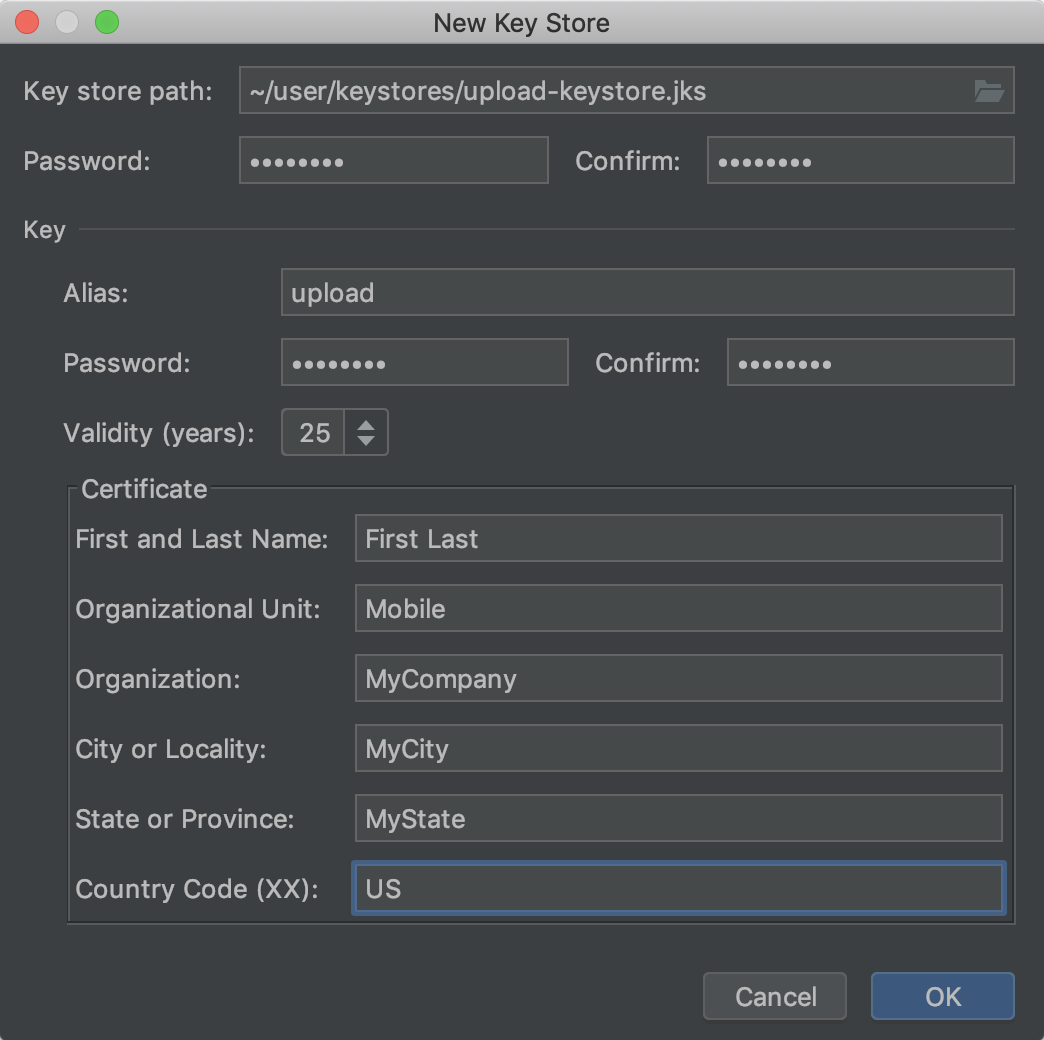


Figure 2. Create a new upload key and keystore in Android Studio.

.

## Sign your app with your key

If you already have an upload key, use it to sign your app. If instead your app is already signed and published to the Google Play store with an existing app signing key, use it to sign your app and make sure to encrypt and export it to opt your app in to Play App Signing. You can later generate a separate upload key and register your upload key’s public certificate with Google Play to sign and upload subsequent updates to your app.

To sign your app using Android Studio, and export an existing app signing key, follow these steps:

1. If you do not currently have the Generate Signed Bundle or APK dialog open, click Build > Generate Signed Bundle/APK.
2. In the Generate Signed Bundle or APK dialog, select either Android App Bundle or APK and click Next.
3. Select a module from the drop down.
4. Specify the path to your keystore, the alias for your key, and enter the passwords for both. If you have not yet prepared your upload keystore and key, first Generate an upload key and keystore and then return to complete this step.

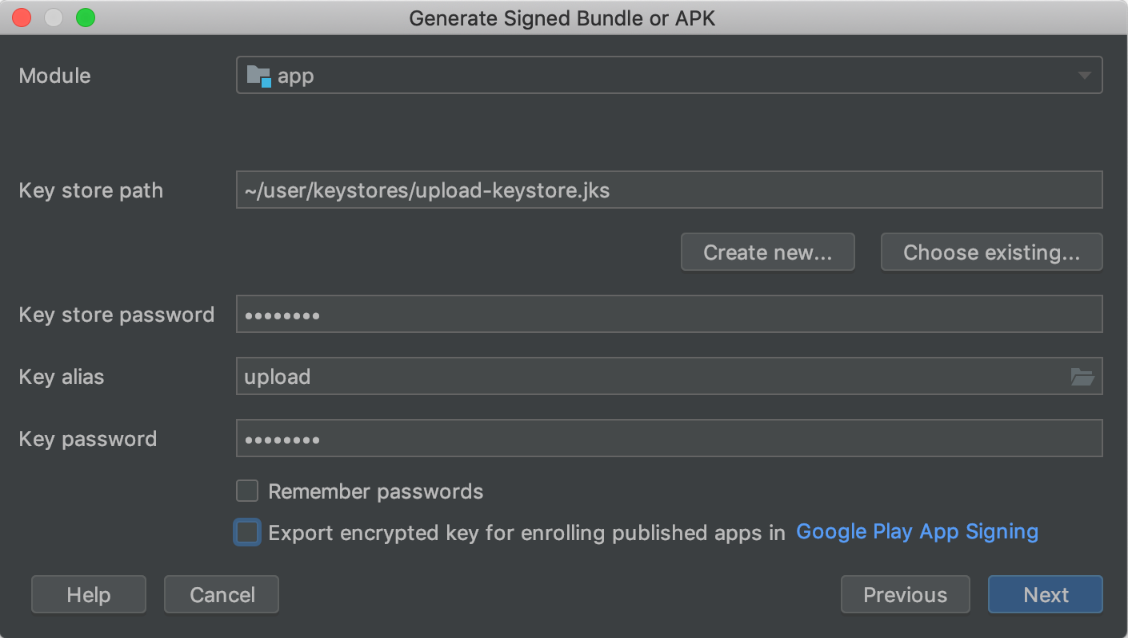


Figure 3. Sign your app with your upload key.

1. If you are signing an app bundle with an existing app signing key, and you would like to later opt your app into Play App Signing, check the box next to Export encrypted key and specify a path to save your signing key as an encrypted \*.pepk file. You can then use your encrypted app signing key to opt in an existing app into Play App Signing.
2. Click Next.
3. In the next window (shown in figure 4), select a destination folder for your signed app, select the build type, choose the product flavor(s) if applicable.
4. If you are building and signing an APK, you need to select which Signature Versions you want your app to support.
5. Click Finish.

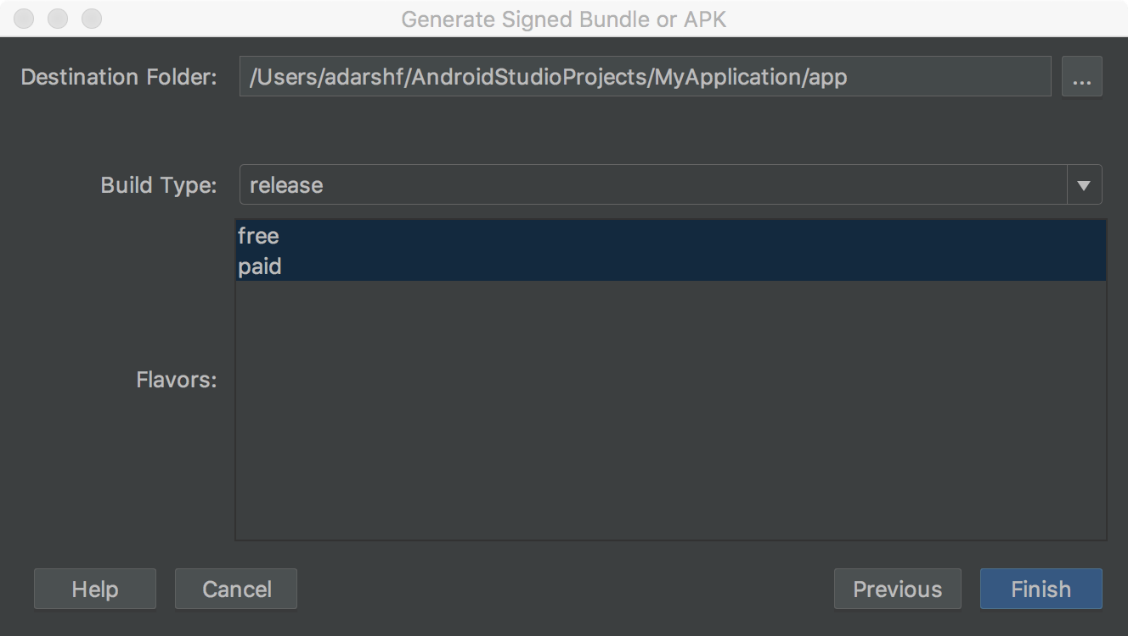


Figure 4. Generate a signed version of your app for the selected product flavors.

After Android Studio finishes building your signed app, you can either locate or analyze your app by clicking on the appropriate option in the pop-up notification. If you selected the option to export your signing key, you can quickly navigate to it by clicking the dropdown arrow in the bottom right corner of the popup to expand it and clicking Show Exported Key File, as shown in figure 5.

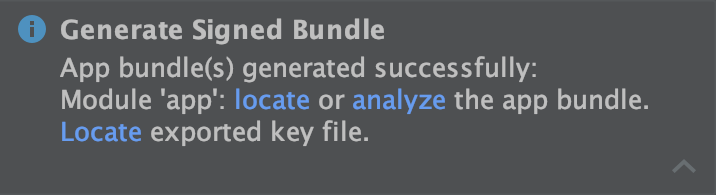


Figure 5. Click the link in the popup to analyze or locate your app bundle or locate your exported signing key.

## e. Opt in to Play App Signing

Play App Signing is the recommended way to sign your app for distribution through Google Play. The steps you need to take in order to opt in your app depends on whether your app has not yet been published to Google Play, or your app is already signed and published using an existing app signing key.

To opt in an app that has not yet been published to Google Play, proceed as follows:

1. If you have not already done so, generate an upload key and sign your app with that upload key.
2. Sign into your Play Console.
3. Follow the steps to prepare & roll out your release to create a new release.
4. After you choose a release track, configure app signing under the Let Google create and manage my app signing key section as follows:

* To have Google Play generate an app signing key for you and use it to sign your app, select Continue. The key you use to sign your first release becomes your upload key, and you should use it to sign future releases.
* To use the same key as another app on your developer account, select Advanced options > Use the same key as another app in this account, select an app, and then click Continue.
* To provide your own signing key for Google to use when signing your app, select Advanced options and select one of the options that lets you securely upload a private key and its public certificate.

In the section called Android App Bundles and APKs to add, click Browse files to locate and upload the app you signed using your upload key. For more information about releasing your app, refer to prepare & roll out your release. When you release your app after opting into Play App Signing, Google Play generates and manages your app’s signing key for you. Simply sign subsequent updates to your app using your app’s upload key before uploading it to Google Play.

If you are updating an app that is already published to Google Play using an existing app signing key, you can opt in to Play App Signing as follows:

1. If you have not already done so, sign your app with your existing app signing key and make sure to check the box next to Export encrypted key to save your signing key as an encrypted \*.pepk file. You will need this file in a later step.
2. Sign in to your Play Console and navigate to your app.
3. On the left menu, click Release management > App signing.
4. If applicable, review the Terms of Service and select Accept.
5. Select one of the options that best describes the signing key you want to upload to Google Play and follow the instructions that are shown. For example, if you used Android Studio to export your app’s signing key, as described on this page, select Upload a key exported from Android Studio and upload the \*.pepk file for your key.
6. Click Enroll.

You should now see a page with the details of your app’s signing and upload certificates. Google Play now signs your app with your existing key when deploying it to users. However, one of the most important benefits to Play App Signing is the ability to separate the key you use to sign the artifact you upload to Google Play from the key that Google Play uses to sign your app for distribution to users. So, consider following the steps in the next section to generate and register a separate upload key.

When you are publishing an app that is not signed by an upload key, the Google Play Console provides the option to register one for future updates to the app. Although this is an optional step, it is recommended that you publish your app with a key that is separate from the one Google Play uses to distribute your app to users. That way, Google keeps your signing key secure, and you have the option to reset a lost or compromised private upload key. This section describes how to create an upload key, generate an upload certificate from it, and register that certificate with Google Play for future updates of your app.

The following describes the situations in which you see the option to register an upload certificate in the Play Console:

* When you publish a new app that is signed with a signing key and opt it in to Play App Signing.
* When you are about to publish an existing app that is already opted in to Play App Signing, but it is signed using its signing key.

In some circumstances, you might want to change your app's signing key. For example, because you want a cryptographically stronger key, or your signing key has been compromised. However, because users can only update your app if the update is signed with the same signing key, it is difficult to change the signing key for an app that is already published.

If you publish your app to Google Play, you can upgrade the signing key for your published app through the Play Console—your new key is used to sign new installs and app updates, while your older app signing key is used to sign updates for users who installed your app before the key upgrade. If you lost your private upload key or your private key has been compromised, you can create a new one and contact the Google Play support team to reset the key. Resetting your upload key will not affect the app signing key that Google Play uses to re-sign APKs before delivering to users.

In Android Studio, you can configure your project to sign the release version of your app automatically during the build process by creating a signing configuration and assigning it to your release build type. A signing configuration consists of a keystore location, keystore password, key alias, and key password. To create a signing configuration and assign it to your release build type using Android Studio, complete the following steps:

1. In the Project window, right click on your app and click Open Module Settings.
2. On the Project Structure window, under Modules in the left panel, click the module you would like to sign.
3. Click the Signing tab, then click Add.
4. Select your keystore file, enter a name for this signing configuration (as you may create more than one), and enter the required information.

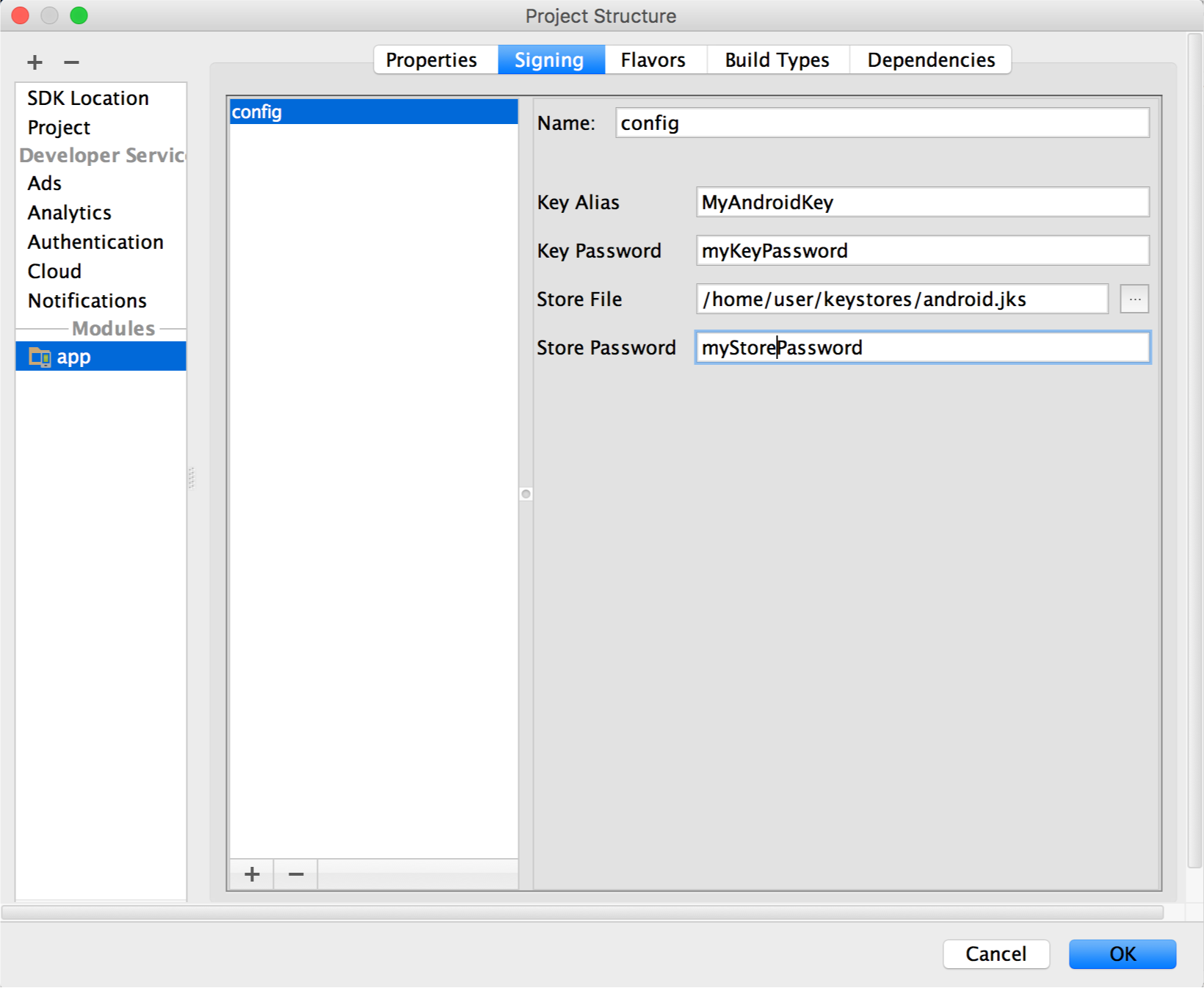


Figure 6. The window for creating a new signing configuration.

1. Click the Build Types tab.
2. Click the release build.
3. Under Signing Config, select the signing configuration you just created.

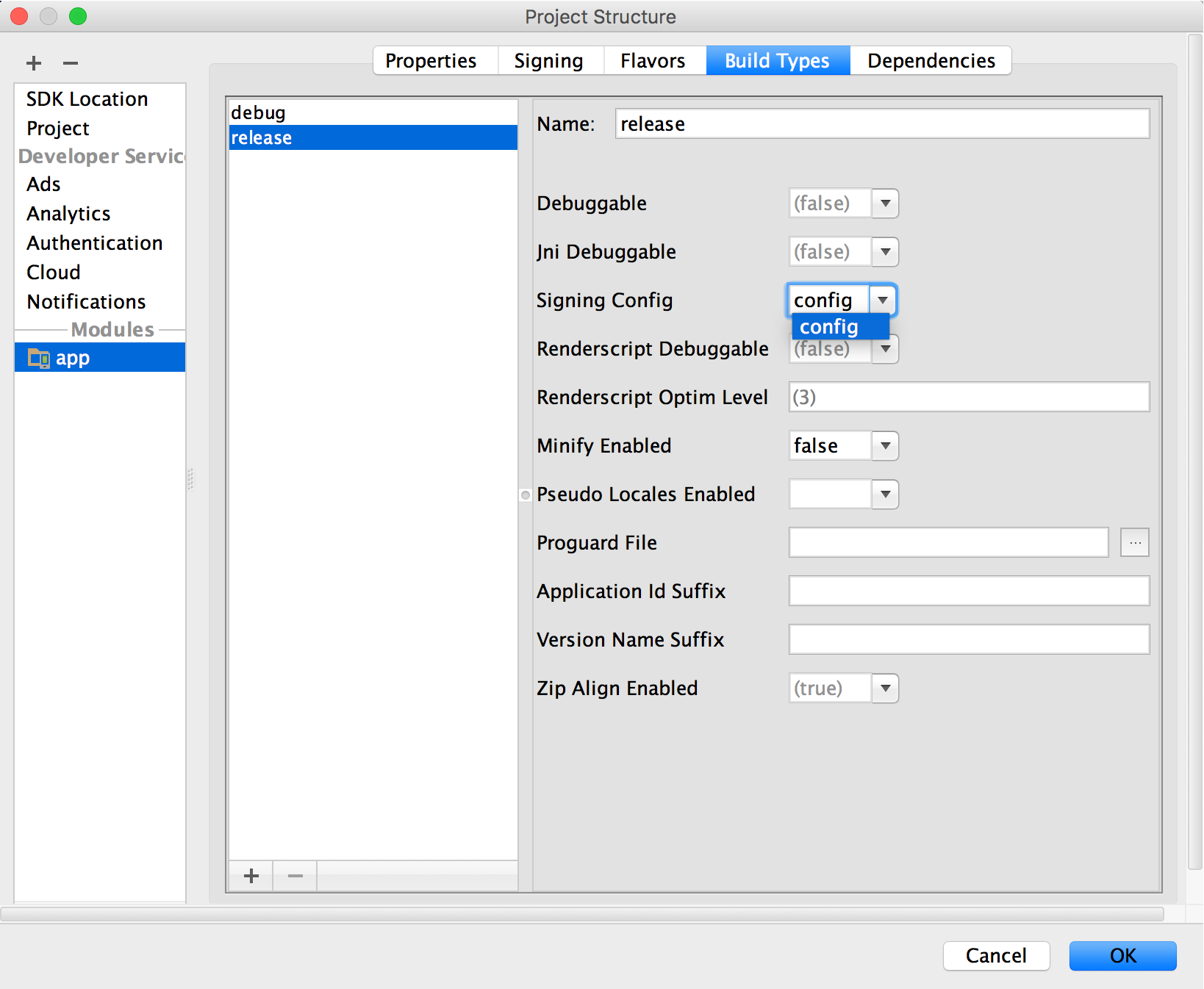


Figure 7. Select a signing configuration in Android Studio.

1. Click OK.

Now every time you build your release build type by selecting an option under Build > Build Bundle(s) / APK(s) in Android Studio, the IDE will sign your app automatically, using the signing configuration you specified. You can find your signed APK or app bundle in the build/outputs/ directory inside the project directory for the module you are building.

When you create a signing configuration, your signing information is included in plain text in your Gradle build files. If you are working in a team or sharing your code publicly, you should keep your signing information secure by removing it from the build files and storing it separately.

If your app uses product flavors and you would like to sign each flavor differently, you can create additional signing configurations and assign them by flavor:

1. In the Project window, right click on your app and click Open Module Settings.
2. On the Project Structure window, under Modules in the left panel, click the module you would like to sign.
3. Click the Signing tab, then click Add.
4. Select your keystore file, enter a name for this signing configuration (as you may create more than one), and enter the required information.

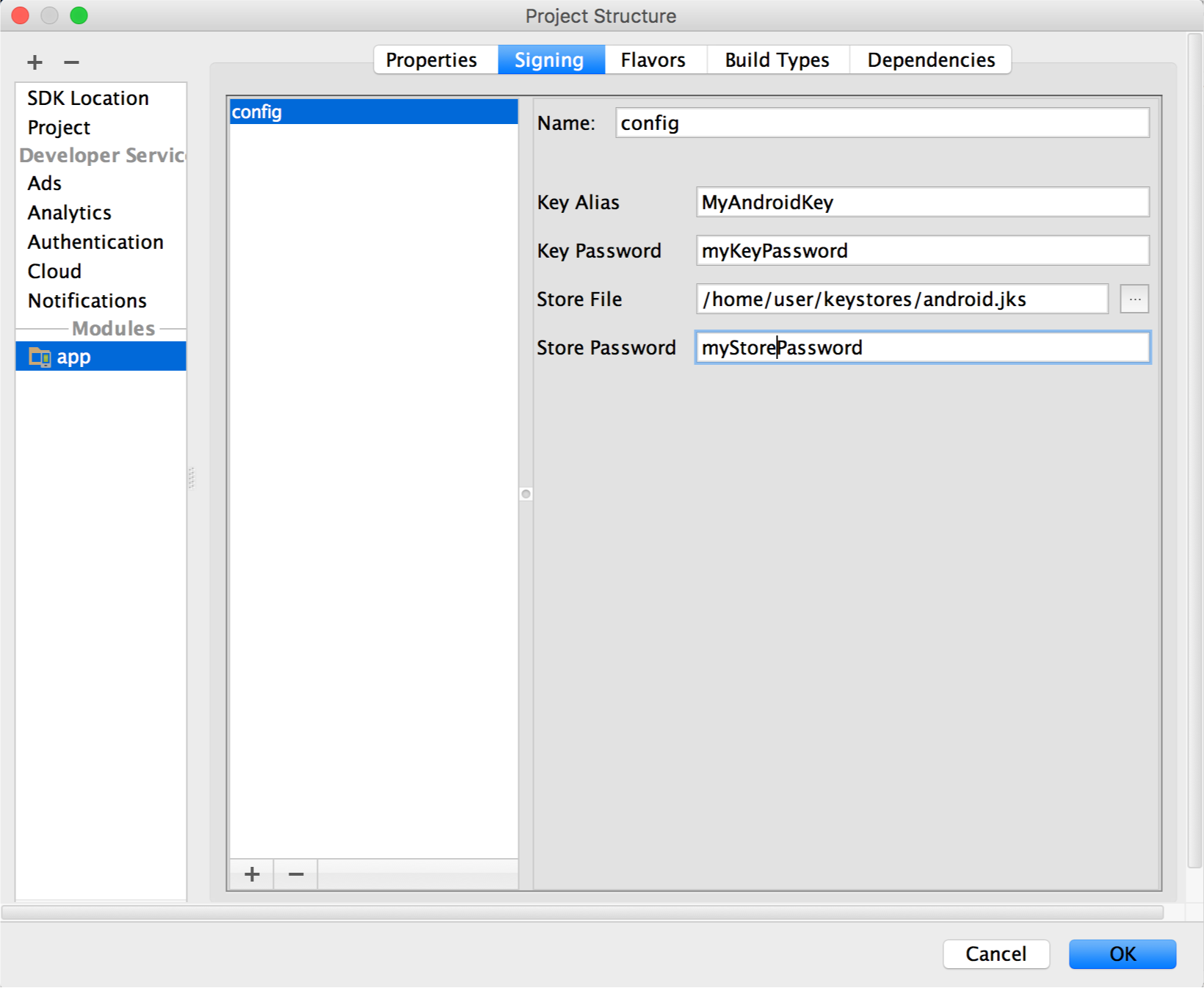


Figure 8. The window for creating a new signing configuration.

1. Repeat steps 3 and 4 as necessary until you have created all your signing configurations.
2. Click the Flavors tab.
3. Click the flavor you would like to configure, then select the appropriate signing configuration from the Signing Config dropdown menu.

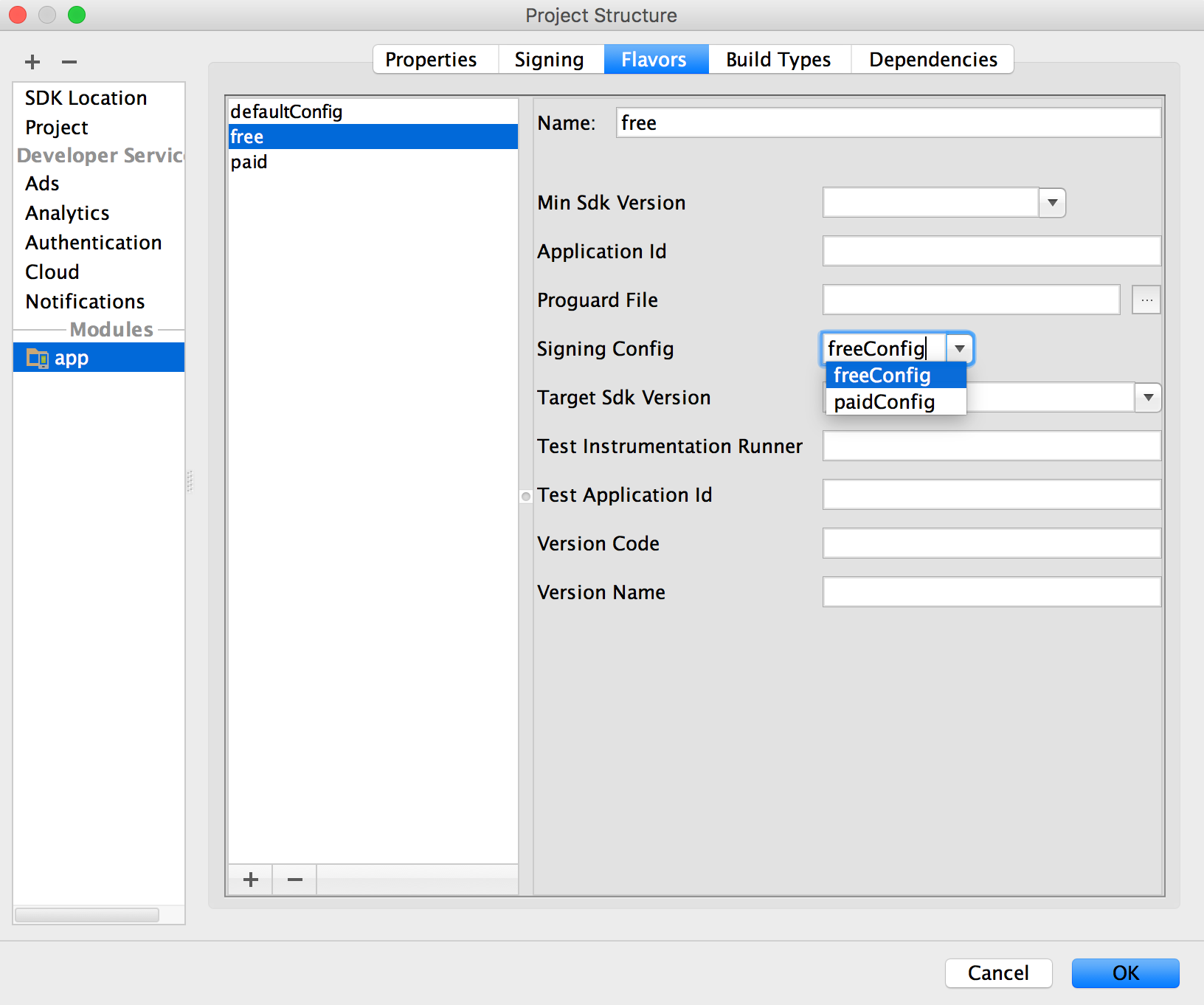


Figure 9. Configure signing settings by product flavor.

Repeat to configure any additional product flavors.

1. Click OK.

## f. Manage your own signing key

If you choose not to opt in to Play App Signing, you can manage your own app signing key and keystore. Keep in mind, you are responsible for securing the key and the keystore. Additionally, your app will not be able to support Android App Bundles, Play Feature Delivery and Play Asset Delivery.

When you are ready to create your own key and keystore, make sure you first choose a strong password for your keystore and a separate strong password for each private key stored in the keystore. You must keep your keystore in a safe and secure place. If you lose access to your app signing key or your key is compromised, Google cannot retrieve the app signing key for you, and you will not be able to release new versions of your app to users as updates to the original app. For more information, see Secure your key, below.

If you manage your own app signing key and keystore, when you sign your APK, you will sign it locally using your app signing key and upload the signed APK directly to the Google Play Store for distribution as shown in figure 10.

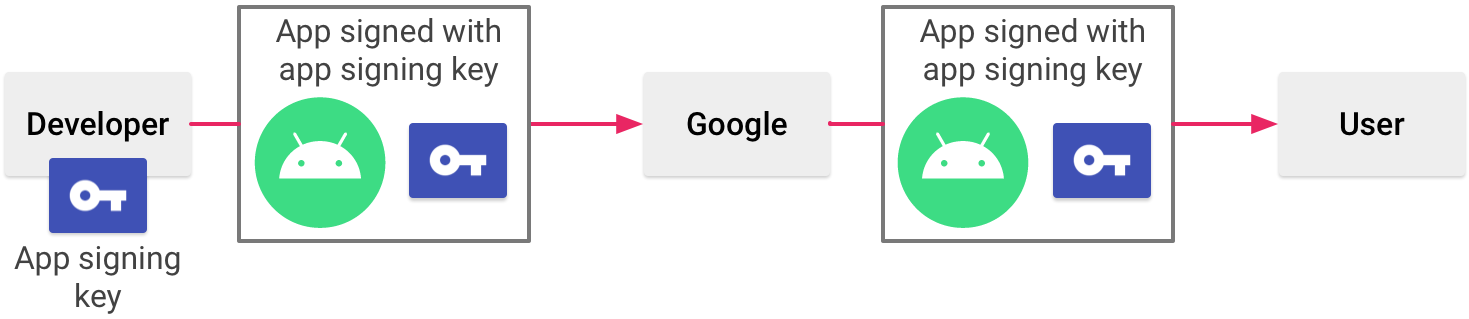


Figure 6. Signing an app when you manage your own app signing key

When you use Play App Signing, Google keeps your signing key safe, and ensures your apps are correctly signed and able to receive updates throughout their lifespans. However, if you decide to manage your app signing key yourself, there are a few considerations you should keep in mind. You should sign your app with the same certificate throughout its expected lifespan. There are several reasons why you should do so:

* App upgrade: When the system is installing an update to an app, it compares the certificate(s) in the new version with those in the existing version. The system allows the update if the certificates match. If you sign the new version with a different certificate, you must assign a different package name to the app—in this case, the user installs the new version as a completely new app.
* App modularity: Android allows APKs signed by the same certificate to run in the same process, if the apps so request, so that the system treats them as a single app. In this way you can deploy your app in modules, and users can update each of the modules independently.
* Code/data sharing through permissions: Android provides signature-based permissions enforcement, so that an app can expose functionality to another app that is signed with a specified certificate. By signing multiple APKs with the same certificate and using signature-based permissions checks, your apps can share code and data in a secure manner.

If you plan to support upgrades for an app, ensure that your app signing key has a validity period that exceeds the expected lifespan of that app. A validity period of 25 years or more is recommended. When your key's validity period expires, users will no longer be able to seamlessly upgrade to new versions of your app.

If you choose to manage and secure your app signing key and keystore yourself (instead of opting in to Play App Signing), securing your app signing key is of critical importance, both to you and to the user. If you allow someone to use your key, or if you leave your keystore and passwords in an unsecured location such that a third-party could find and use them, your authoring identity and the trust of the user are compromised.

If you use Play App Signing, your app signing key is kept secure using Google's infrastructure. You should keep your upload key secure as described below. If your upload key is compromised, you can contact Google to revoke it and receive a new upload key.

If a third party should manage to take your key without your knowledge or permission, that person could sign and distribute apps that maliciously replace your authentic apps or corrupt them. Such a person could also sign and distribute apps under your identity that attack other apps or the system itself, or corrupt or steal user data.

Your private key is required for signing all future versions of your app. If you lose or misplace your key, you will not be able to publish updates to your existing app. You cannot regenerate a previously generated key. Your reputation as a developer entity depends on securing your app signing key properly, at all times, until the key is expired. Here are some tips for keeping your key secure:

* Select strong passwords for the keystore and key.
* Do not give or lend anyone your private key, and do not let unauthorized persons know your keystore and key passwords.
* Keep the keystore file containing your private key in a safe, secure place.

In general, if you follow common-sense precautions when generating, using, and storing your key, it will remain secure.

# **METHOD TO UPLOAD YOUR APP TO THE PLAY CONSOLE (APPLICATION PACKAGE)**

After you [build and sign the release version of your app](https://developer.android.com/studio/publish/app-signing#sign-apk), the next step is to upload it to Google Play to inspect, test, and publish your app. Before you get started, you might want to make sure you satisfy the following:

1. If you haven't already done so, enroll into Play App Signing, which is the recommended way to upload and sign your app. If you build and upload an Android App Bundle, you must enroll in app Play App Signing.
2. Google Play supports compressed app downloads of only 150 MB or less. To learn more, read Compressed download size restriction.

After you've met the requirements above, go ahead and upload your app to the Play Console. This page also describes how you can test and update your app bundle after it's been uploaded.

## Inspecting APKs using bundle explorer

If you upload your app as an Android App Bundle, the Play Console automatically generates split APKs and multi-APKs for all device configurations your app supports. In the Play Console, you can use the App Bundle Explorer to see all APK artifacts that Google Play generates, inspect data such as supported devices and APK size savings, and download generated APKs to deploy and test locally.

To learn more, read the Play Console help topic about Reviewing your app bundle details.

## Test using the internal test track

The easiest and fastest way to test your app after you upload an app bundle is by using Play Console’s internal test track. This new test track is aimed towards quickly sharing your app with a limited number of internal testers. And, because the only way to test downloading and installing feature modules on demand is through Google Play, the internal test track is where you'll want to upload your app bundle to fine-tune this type of user experience.

## Updating your app bundle

After you upload your app to the Play Console, updating your app requires you to increase the version code you include in the base module, and build and upload a new app bundle. Google Play then generates updated APKs with new version codes and serves them to users as needed.

**QUESTIONS:**

1. Explain the term “public key certificate” in the google play store.

Answer:

A public-key certificate (der or .pem files), also known as a digital certificate or an identity certificate, contains the public key of a public/private key pair, as well as some other metadata identifying the owner (for example, name and location) who holds the corresponding private key.

**REFERENCE:**

1. https://source.android.com/devices/tech/debug
2. https://source.android.com/devices/tech/debug/understanding-logging
3. https://source.android.com/devices/tech/debug/scoped-vendor-logging
4. https://source.android.com/devices/tech/debug/native-crash
5. https://source.android.com/devices/tech/debug/gdb