Signing

Sign your app

Android requires that all APKs be digitally signed with a certificate before they can be installed. This document describes how to sign your APKs using Android Studio, including creating and storing your certificate, signing different build configurations using different certificates, and configuring the build process to sign your APKs automatically.

Certificates and keystores

A public-key certificate, 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 of the key (for example, name and location). The owner of the certificate holds the corresponding private key.

When you sign an APK, the signing tool attaches the public-key certificate to the APK. The public-key certificate serves as a "fingerprint" that uniquely associates the APK to you and your corresponding private key. This helps Android ensure that any future updates to your APK are authentic and come from the original author. The key used to create this certificate is called the *app signing key*.

A keystore is a binary file that contains one or more private keys.

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. For more about the benefits of using the same certificate for all your apps throughout their lifespans, see Signing Considerations below.

Sign your debug build

When running or debugging your project from the IDE, Android Studio automatically signs your APK 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) will not accept an APK 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 an APK, 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.

For more information about how to build and run apps for debugging, see Build and Run Your App.

Expiry of the debug certificate

The self-signed certificate used to sign your APK for debugging has an expiration date of 365 days from its creation date. When the certificate expires, you will get a build error.

To fix this problem, simply delete the debug.keystore file. The file is stored in the following locations:

- ~/.android/ on OS X and Linux
- C:\Documents and Settings\<user>\.android\ on Windows XP
- C:\Users\<user>\.android\ on Windows Vista and Windows 7, 8, and 10

The next time you build and run the debug build type, the build tools will regenerate a new keystore and debug key. Note that you must run your app, building alone does not regenerate the keystore and debug key.

Manage your key

Because your app signing key is used to verify your identity as a developer and to ensure seamless and secure updates for your users, managing your key and keeping it secure are very important, both for you and for your users. You can choose either to opt in to use Google Play App Signing to securely manage and store your app signing key using Google's infrastructure or to manage and secure your own keystore and app signing key.

By opting in to Google Play App Signing, you will gain the following benefits:

- Ensure that the app signing key is not lost. Loss of the app signing key means that an app cannot be updated, so it is critical for it not to be lost.
- Ensure that the app signing key is not compromised. Compromise of the key would allow a malicious attacker to deploy a malicious version of your app as an update over an

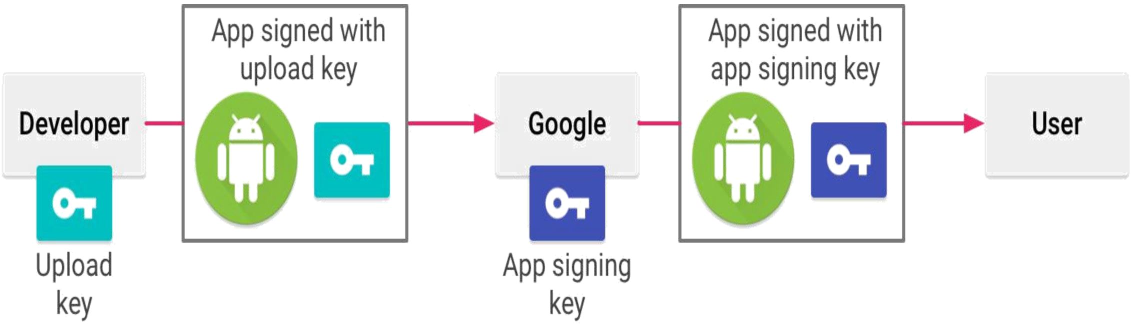

Figure 1. Signing an app with Google Play App Signing

existing install. With Play App Signing, developers only manage an upload key which can be reset in the case of loss and compromise. In the event of compromise, an attacker also needs access to the developer account to be able to do anything malicious.

Use Google Play App Signing

When using Google Play App Signing, you will use two keys: the *app signing key* and the *upload key*. Google manages and protects the app signing key for you, and you keep the upload key and use it to sign your apps for upload to the Google Play Store.

When you opt in to use Google Play App Signing, you export and encrypt your app signing key using the Play Encrypt Private Key tool provided by Google Play, and then upload it to Google's infrastructure. Then you create a separate upload key and register it with Google. When you are ready to publish, you sign your app using the upload key and upload it to Google Play. Google then uses the upload certificate to verify your identity, and re-signs your APK with your app signing key for distribution as shown in figure 1. (If you do not already have an app signing key, you can generate one during the sign-up process.)

When you use Google 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.

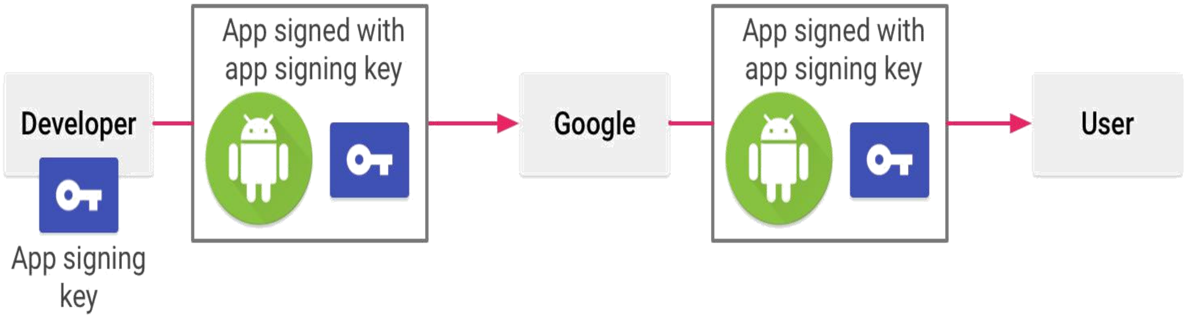
Developer
App signing key
App signed with app signing key
Google
App signed with app signing key
User

Figure 2. Signing an app when you manage your own app signing key

Manage your own key and keystore

Instead of using Google Play App Signing, you can choose to manage your own app signing key and keystore. If you choose to manage your own app signing key and keystore, you are responsible for securing the key and the keystore. You should 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 2.

Sign an APK

Regardless of how you choose to manage your key and keystore, you can use Android Studio to sign your APKs (with either the upload key or the app signing key), either manually, or by configuring your build process to automatically sign APKs.

If you choose to manage and secure your own app signing key and keystore, you will sign your APKs with your app signing key. If you choose to use Google Play App Signing to manage and secure your app signing key and keystore, you will sign your APKs with your upload key.