Necessary information for Play Market

- Category of your app
- 512x512px app icon of any shape (can be transparent)
- Several screenshots

Required information about the publisher

For this, you need to answer the following questions:

- What is your first and last name?
- What is the name of your organizational unit?
- What is the name of your organization? (may be the same as Profile Name)
- What is the name of your City or Locality?
- What is the name of your State or Province?
- What is the two-letter country code for this unit?
- What is your (support) phone number?
- What is your email? (may be the same as Dev Apple Id you used for deployment or Play Market Console account email)

Generating Signed APK

Android requires that all apps be digitally signed with a certificate before they can be installed, so to distribute your Android application via Google Play store, you'll need to generate a signed release APK. The Signing Your Applications page on Android Developers documentation describes the topic in detail. This guide covers the process in brief, as well as lists the steps required to package the JavaScript bundle.

Generating a signing key

You can generate a private signing key using `keytool`. On Windows `keytool` must be run from `C:\Program Files\Java\jdkx.x.x_x\bin`.

This command prompts you for passwords for the keystore and key and for the Distinguished Name fields for your key. It then generates the keystore as a file called `my-release-key.keystore`.

The keystore contains a single key, valid for 10000 days. The alias is a name that you will use later when signing your app, so remember to take note of the alias.

On Mac, if you're not sure where your jdk bin folder is, then perform the following command to find it:

It will output the directory of the jdk, which will look something like this:

Navigate to that directory by using the command `$ cd /your/jdk/path` and use the keytool command with sudo permission as shown below.

Note: Remember to keep your keystore file private and never commit it to version control.

Setting up gradle variables

1. Place the `my-release-key.keystore` file under the `android/app` directory in your project folder.
2. Edit the file `~/.gradle/gradle.properties` or `android/gradle.properties`, and add the following (replace `*****` with the correct keystore password, alias and key password),

These are going to be global gradle variables, which we can later use in our gradle config to sign our app.

Note about saving the keystore:

Once you publish the app on the Play Store, you will need to republish your app under a different package name (losing all downloads and ratings) if you

want to change the signing key at any point. So backup your keystore and don't forget the passwords.

Note about security: If you are not keen on storing your passwords in plaintext, and you are running OSX, you can also store your credentials in the Keychain Access app. Then you can skip the two last rows in `~/.gradle/gradle.properties`.

Adding signing config to your app's gradle config

Edit the file `android/app/build.gradle` in your project folder, and add the signing config,

Generating the release APK

Simply run the following in a terminal:

Gradle's `assembleRelease` will bundle all the JavaScript needed to run your app into the APK. If you need to change the way the JavaScript bundle and/or drawable resources are bundled (e.g. if you changed the default file/folder names or the general structure of the project), have a look at `android/app/build.gradle`to see how you can update it to reflect these changes.

Note: Make sure gradle.properties does not include *org.gradle.configureondemand=true* as that will make the release build skip bundling JS and assets into the APK.

The generated APK can be found under `android/app/build/outputs/apk/release/app-release.apk`, and is ready to be distributed.

Testing the release build of your app

Before uploading the release build to the Play Store, make sure you test it thoroughly. First uninstall any previous version of the app you already have installed. Install it on the device using:

Note that `--variant=release` is only available if you've set up signing as described above.

You can kill any running packager instances, since all your framework and JavaScript code is bundled in the APK's assets.

Split APKs by ABI to reduce file size

By default, the generated APK has the native code for both x86 and ARMv7a CPU architectures. This makes it easier to share APKs that run on almost all Android devices. However, this has the downside that there will be some unused native code on any device, leading to unnecessarily bigger APKs.

You can create an APK for each CPU by changing the following line in android/app/build.gradle:

Upload both these files to markets which support device targetting, such as Google Play and Amazon AppStore, and the users will automatically get the appropriate APK. If you want to upload to other markets, such as APKFiles, which do not support multiple APKs for a single app, change the following line as well to create the default universal APK with binaries for both CPUs.

Enabling Proguard to reduce the size of the APK (optional)

Proguard is a tool that can slightly reduce the size of the APK. It does this by stripping parts of the React Native Java bytecode (and its dependencies) that your app is not using.

IMPORTANT: Make sure to thoroughly test your app if you've enabled Proguard. Proguard often requires configuration specific to each native library you're using. See `app/proguard-rules.pro`.

To enable Proguard, edit `android/app/build.gradle`:

```
/**
 * Run Proguard to shrink the Java bytecode in release builds.
 */
def enableProguardInReleaseBuilds = true
```