Mac App Store Submission Guide

This guide provides information on:

- How to sign Electron apps on macOS;
- How to submit Electron apps to Mac App Store (MAS);
- The limitations of the MAS build.

Requirements

To sign Electron apps, the following tools must be installed first:

- Xcode 11 or above.
- The electron-osx-sign npm module.

You also have to register an Apple Developer account and join the Apple Developer Program.

Sign Electron apps

Electron apps can be distributed through Mac App Store or outside it. Each way requires different ways of signing and testing. This guide focuses on distribution via Mac App Store, but will also mention other methods.

The following steps describe how to get the certificates from Apple, how to sign Electron apps, and how to test them.

Get certificates

The simplest way to get signing certificates is to use Xcode:

1. Open Xcode and open "Accounts" preferences;
2. Sign in with your Apple account;
3. Select a team and click "Manage Certificates";
4. In the lower-left corner of the signing certificates sheet, click the Add button (+), and add following certificates:
   - "Apple Development"
   - "Apple Distribution"

The "Apple Development" certificate is used to sign apps for development and testing, on machines that have been registered on Apple Developer website. The method of registration will be described in Prepare provisioning profile.

Apps signed with the "Apple Development" certificate cannot be submitted to Mac App Store. For that purpose, apps must be signed with the "Apple Distribution" certificate instead. But note that apps signed with the "Apple Distribution" certificate cannot run directly, they must be re-signed by Apple to be able to run, which will only be possible after being downloaded from the Mac App Store.

Other certificates You may notice that there are also other kinds of certificates.

The "Developer ID Application" certificate is used to sign apps before distributing them outside the Mac App Store.

The "Developer ID Installer" and "Mac Installer Distribution" certificates are used to sign the Mac Installer Package instead of the app itself. Most Electron apps do not use Mac Installer Package so they are generally not needed.

The full list of certificate types can be found here.

Apps signed with "Apple Development" and "Apple Distribution" certificates can only run under App Sandbox, so they must use the MAS build of Electron. However, the "Developer ID Application" certificate does not have this restrictions, so apps signed with it can use either the normal build or the MAS build of Electron.

Legacy certificate names Apple has been changing the names of certificates during past years, you might encounter them when reading old documentations, and some utilities are still using one of the old names.

- The "Apple Distribution" certificate was also named as "3rd Party Mac Developer Application" and "Mac App Distribution".
- The "Apple Development" certificate was also named as "Mac Developer" and "Development".

Prepare provisioning profile

If you want to test your app on your local machine before submitting your app to the Mac App Store, you have to sign the app with the "Apple Development" certificate with the provisioning profile embedded in the app bundle.

To create a provisioning profile, you can follow the below steps:

1. Open the "Certificates, Identifiers & Profiles" page on the Apple Developer website.
2. Add a new App ID for your app in the "Identifiers" page.
3. Register your local machine in the "Devices" page. You can find your machine's "Device ID" in the "Hardware" page of the "System Information" app.
4. Register a new Provisioning Profile in the "Profiles" page, and download it to `/path/to/yourapp.provisionprofile`.

Enable Apple's App Sandbox

Apps submitted to the Mac App Store must run under Apple's App Sandbox, and only the MAS build of Electron can run with the App Sandbox. The standard darwin build of Electron will fail to launch when run under App Sandbox.

When signing the app with `electron-osx-sign`, it will automatically add the necessary entitlements to your app's entitlements, but if you are using custom entitlements, you must ensure App Sandbox capacity is added:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList
<plist version="1.0">
  <dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
  </dict>
</plist>
```

Extra steps without `electron-osx-sign` If you are signing your app without using `electron-osx-sign`, you must ensure the app bundle's entitlements have at least following keys:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList
<plist version="1.0">
  <dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.application-groups</key>
    <array>
      <string>TEAM_ID.your.bundle.id</string>
    </array>
  </dict>
</plist>
```

The `TEAM_ID` should be replaced with your Apple Developer account's Team ID, and the `your.bundle.id` should be replaced with the App ID of the app.

And the following entitlements must be added to the binaries and helpers in the app's bundle:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList
<plist version="1.0">
  <dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.inherit</key>
    <true/>
  </dict>
</plist>
```

And the app bundle's `Info.plist` must include `ElectronTeamID` key, which has

your Apple Developer account's Team ID as its value:

```
<plist version="1.0">
<dict>
  ...
  <key>ElectronTeamID</key>
  <string>TEAM_ID</string>
</dict>
</plist>
```

When using `electron-osx-sign` the `ElectronTeamID` key will be added automatically by extracting the Team ID from the certificate's name. You may need to manually add this key if `electron-osx-sign` could not find the correct Team ID.

Sign apps for development

To sign an app that can run on your development machine, you must sign it with the "Apple Development" certificate and pass the provisioning profile to `electron-osx-sign`.

```
electron-osx-sign YourApp.app --identity='Apple Development' --provisioning-profile=/path/to
```

If you are signing without `electron-osx-sign`, you must place the provisioning profile to `YourApp.app/Contents/embedded.provisionprofile`.

The signed app can only run on the machines that registered by the provisioning profile, and this is the only way to test the signed app before submitting to Mac App Store.

Sign apps for submitting to the Mac App Store

To sign an app that will be submitted to Mac App Store, you must sign it with the "Apple Distribution" certificate. Note that apps signed with this certificate will not run anywhere, unless it is downloaded from Mac App Store.

```
electron-osx-sign YourApp.app --identity='Apple Distribution'
```

Sign apps for distribution outside the Mac App Store

If you don't plan to submit the app to Mac App Store, you can sign it the "Developer ID Application" certificate. In this way there is no requirement on App Sandbox, and you should use the normal darwin build of Electron if you don't use App Sandbox.

```
electron-osx-sign YourApp.app --identity='Developer ID Application' --no-gatekeeper-assess
```

By passing `--no-gatekeeper-assess`, the `electron-osx-sign` will skip the macOS GateKeeper check as your app usually has not been notarized yet by this step.

This guide does not cover App Notarization, but you might want to do it otherwise Apple may prevent users from using your app outside Mac App Store.

Submit Apps to the Mac App Store

After signing the app with the "Apple Distribution" certificate, you can continue to submit it to Mac App Store.

However, this guide do not ensure your app will be approved by Apple; you still need to read Apple's Submitting Your App guide on how to meet the Mac App Store requirements.

Upload

The Application Loader should be used to upload the signed app to iTunes Connect for processing, making sure you have created a record before uploading.

If you are seeing errors like private APIs uses, you should check if the app is using the MAS build of Electron.

Submit for review

After uploading, you should submit your app for review.

Limitations of MAS Build

In order to satisfy all requirements for app sandboxing, the following modules have been disabled in the MAS build:

- `crashReporter`
- `autoUpdater`

and the following behaviors have been changed:

- Video capture may not work for some machines.
- Certain accessibility features may not work.
- Apps will not be aware of DNS changes.

Also, due to the usage of app sandboxing, the resources which can be accessed by the app are strictly limited; you can read App Sandboxing for more information.

Additional entitlements

Depending on which Electron APIs your app uses, you may need to add additional entitlements to your app's entitlements file. Otherwise, the App Sandbox may prevent you from using them.

Network access Enable outgoing network connections to allow your app to connect to a server:

```
<key>com.apple.security.network.client</key>
<true/>
```

Enable incoming network connections to allow your app to open a network listening socket:

```
<key>com.apple.security.network.server</key>
<true/>
```

See the Enabling Network Access documentation for more details.

dialog.showOpenDialog

```
<key>com.apple.security.files.user-selected.read-only</key>
<true/>
```

See the Enabling User-Selected File Access documentation for more details.

dialog.showSaveDialog

```
<key>com.apple.security.files.user-selected.read-write</key>
<true/>
```

See the Enabling User-Selected File Access documentation for more details.

Cryptographic Algorithms Used by Electron

Depending on the countries in which you are releasing your app, you may be required to provide information on the cryptographic algorithms used in your software. See the encryption export compliance docs for more information.

Electron uses following cryptographic algorithms:

- AES - NIST SP 800-38A, NIST SP 800-38D, RFC 3394
- HMAC - FIPS 198-1
- ECDSA - ANS X9.62–2005
- ECDH - ANS X9.63–2001
- HKDF - NIST SP 800-56C
- PBKDF2 - RFC 2898
- RSA - RFC 3447
- SHA - FIPS 180-4
- Blowfish - https://www.schneier.com/cryptography/blowfish/
- CAST - RFC 2144, RFC 2612
- DES - FIPS 46-3
- DH - RFC 2631
- DSA - ANSI X9.30
- EC - SEC 1

- IDEA - “On the Design and Security of Block Ciphers” book by X. Lai
- MD2 - RFC 1319
- MD4 - RFC 6150
- MD5 - RFC 1321
- MDC2 - ISO/IEC 10118-2
- RC2 - RFC 2268
- RC4 - RFC 4345
- RC5 - https://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf
- RIPEMD - ISO/IEC 10118-3