Application Signing with 4D

By Timothy Aaron Penner, Senior Technical Services Engineer, 4D Inc.

Technical Note 18-15

Table of Contents

Introduction

Application signing is an integral part of distribution cycle of applications. It is sort of like a tamper-resistant bottles or a 'void if removed' sticker – it serves as a way for the consumer to identify if the item has been tampered with. The signed application is only valid if application has not been tampered with. In contrast, making changes to the binary file after signing the application will invalidate the signature. Although this is an extra step in the in final stages of development, the added development time equates to better peace of mind for the end users.

Why sign an application?

There are many advantages to signing an application before distributing it to customers, and possibly only a few disadvantages. Generally speaking, the advantages outweigh the disadvantages; but it is up to the developer to decide what is right for their own development and deployment practice. For example, a developer providing in-house development work for internal database may not find this to be as important as it would be for someone developing a vertical market application. Although, application signing could actually be beneficial in both situations. Let's explore the pros and cons.

Advantages of signing an app

Signing the application can be advantageous for both the developer and the end user. Here are a few examples of the advantages:

- Protects binary from being manipulated
- Proves the application has not been tampered with
- Proves the application is coming from a trusted source
- Gets rid of the "Unknown Publisher" message when running on Windows

Disadvantages of signing an app

There are a few potential disadvantages to signing an application. Here are some examples of things to consider:

- It is an additional step in the process or preparing the application
  - The development process will become slightly longer
- It requires a certificate from a trusted Certificate Authority ($$$)
  - The development cost will be slightly more
- The first time the application is launched the signature is verified
  - This takes a little bit of extra time upon that initial launch

Obtaining a signing certificate

Before an application can be signed, the developer must obtain a signing certificate. There are some differences in how to obtain the certificate for Mac and Windows so this section of the document is broken into two sections, one specific to Windows and one specific to Mac.

Windows

The signing certificate can be obtained from numerous online SSL Certificate vendors. For this tech note a self-signed certificate is used; if this were for production then it would be recommended to use a certificate from a trusted source instead of a self-signed certificate.

Create a Self-Signed Certificate

The following command and parameters will be used to create a self-signed certificate on Windows:

Command: **New-SelfSignedCertificate**
Parameter 1: **-Type** Custom
Parameter 2: **-Subject** "CN=4D, O=4D Inc, C=US"
Parameter 3: **-KeyUsage** DigitalSignature
Parameter 4: **-FriendlyName** "4D, Inc"
Parameter 5: **-CertStoreLocation** "Cert:\LocalMachine\My"

The values used for this tech note are filled in but the developer will want to use custom data for the **Subject** and **FriendlyName** parameters. The **CertStoreLocation** parameter can also use custom data but this location is used later on so if using something other than what is listed above, it will need to be substituted in later on.

Once the values are updates, put it all together on a single line and run it from an Elevated PowerShell window like this:

```
New-SelfSignedCertificate -Type Custom -Subject "CN=4D, O=4D Inc, C=US" -
KeyUsage DigitalSignature -FriendlyName "4D, Inc" -CertStoreLocation
"Cert:\LocalMachine\My"
```

If the command is executed properly the output may look like this:

```
   PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My

Thumbprint                                Subject
----------                                -------
A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304  CN=4D, O=4D Inc, C=US
```

The thumbprint of the certificate is important and will be used later on. In this example the thumbprint is A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304 but the thumbprint for

those that are following along at home will be different. Be sure to include the correct thumbprint for the examples in the following sections.

Note: This process needs to be done from a PowerShell window with Elevated Privileges.

Export the certificate to PFX

The next step is to export the certificate to a PFX file, but in order to do this a password must be set. The PowerShell command ConvertTo-SecureString is used to create the secure password and store it into a session variable ($pwd). The PowerShell command ExportPfxCertificate is used to export the certificate in PFX format using the $pwd session variable.

The following 2 lines PowerShell code will create a password in the $pwd session variable and then use the $pwd session variable in the exporting of the certificate.

```
$pwd = ConvertTo-SecureString -String "thePassword" -Force -AsPlainText
Export-PfxCertificate -cert
Cert:\LocalMachine\My\A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304 -FilePath
C:\signing\cert.pfx -Password $pwd
```

Here is the breakdown of the first command and parameters used:

Return value stored in: $pwd
Command 1: ConvertToSecureString
Parameter 1: -String “thePassword”
Parameter 2: -Force
Parameter 3: -AsPlainText

The first command written out in the terminal looks like this:

```
$pwd = ConvertTo-SecureString -String "thePassword" -Force -AsPlainText
```

This stores the password (“thePassword”) as a secure string within the $pwd session variable to be used in the next line of PowerShell code. The developer will want to modify the value in the password parameter and then substitute the correct password in for the following examples.

Here is the breakdown of the second command and parameters used:

Command 2: ExportPfxCertificate
Parameter 1: -cert
Cert:\LocalMachine\My\A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304
Parameter 2: -FilePath C:\signing\cert.pfx
Parameter 3: -Password $pwd

The second command written out in the terminal looks like this (due to the length of the line and the width of this document, the single line is depicted on multiple lines):

```
Export-PfxCertificate -cert
Cert:\LocalMachine\My\A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304 -FilePath
C:\signing\cert.pfx -Password $pwd
```

This exports the certificate iditified by the thumbprint and exports it to the path specified by the FilePath parameter, using the password defined in the $pwd session variable. The developer will need to substitute the correct thumbprint when running this command.

Once all of the data points have been updated the developer can run the two commands like this:

```
$pwd = ConvertTo-SecureString -String "thePassword" -Force -AsPlainText
Export-PfxCertificate -cert
Cert:\LocalMachine\My\A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304 -FilePath
C:\signing\cert.pfx -Password $pwd
```

This will place the password protected PFX certificate into the location specified. This PFX file will be used later for signing the application, and the password is required to use the PFX file.

The output observed in PowerShell may look like this:

```
    Directory: C:\signing


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         8/9/2018   3:06 PM           2701 cert.pfx
```

This indicates that the file has been exported and provides a short summary.

MacOS

Signing Certificates for the MacOS platform can be created from either inside of XCode or from the Apple Developer website. In both cases, an active membership to the Apple Developer Program is required.

Obtaining the certificate from Apple's website

The Apple Developer website is available from the following URL: https://developer.apple.com/account/

The steps for obtaining the certificate from Apple's website have changed over the years. Currently, the developer needs to use the iTunes connect portal to obtain the certificate.

Obtaining the certificate using XCode

To create the certificate from within XCode, first launch XCode.

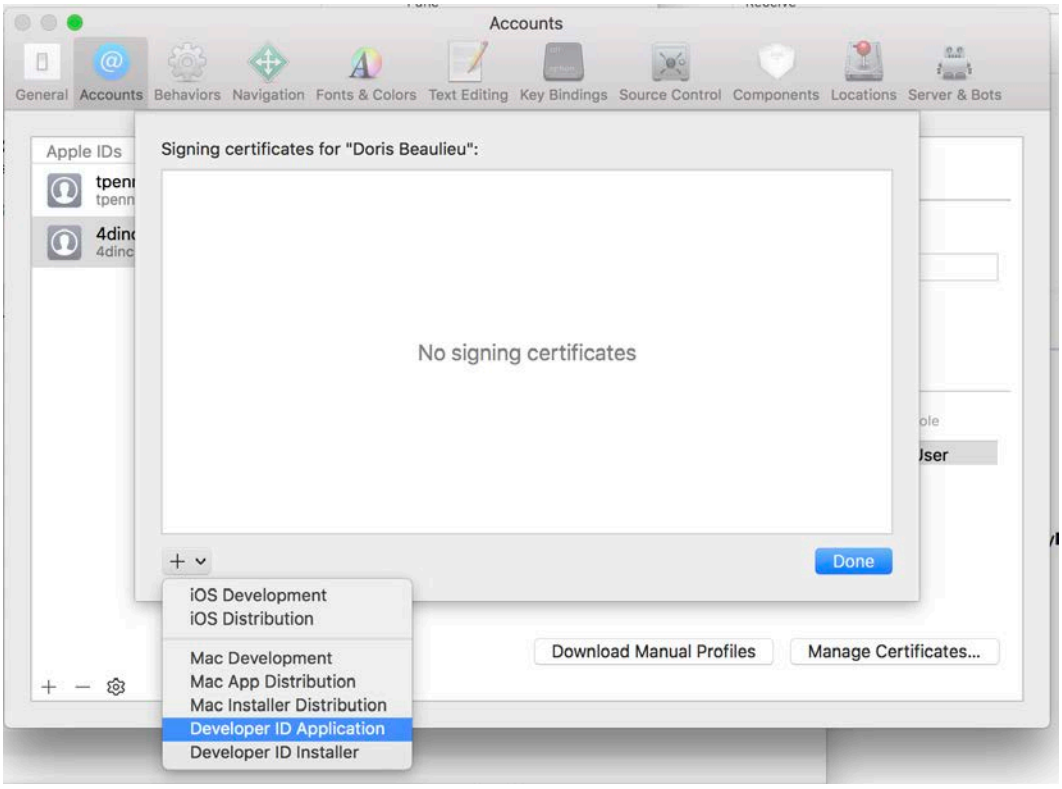
From the XCode drop down menu choose the Preferences menu item:

Choose the Accounts tab.

If the AppleID account is not already added, click on the + button to add the account (this account must have an active membership to the Apple Developer Program).

Once the Apple ID account is added, select the account from the left side of the accounts window,. Then click on the Manage Certificates button.

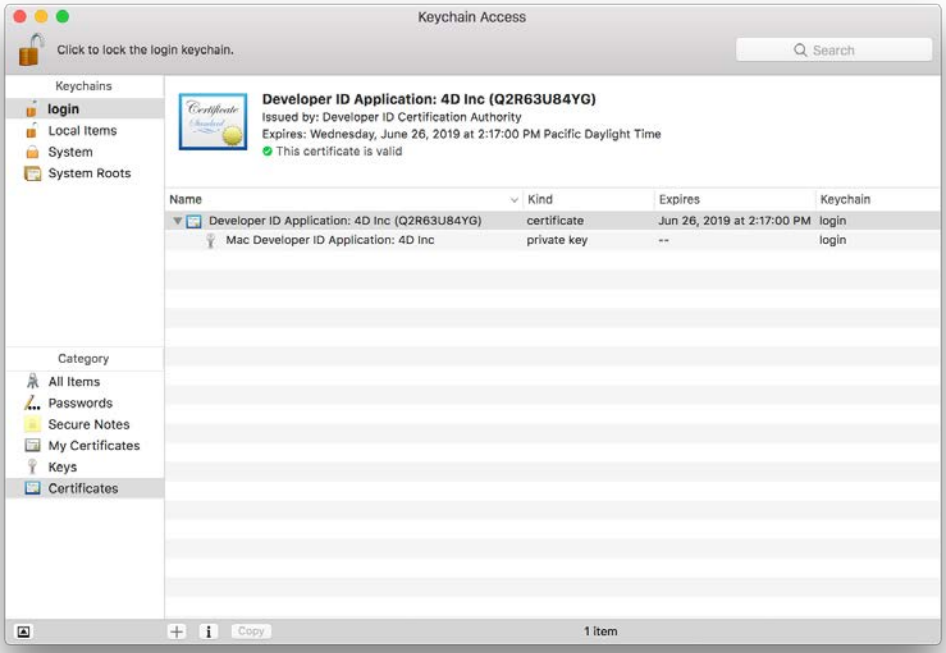
From the manage certificates dialog, click on the + and choose "Developer ID Application".



Follow the prompts to complete the process.

Locating the certificate in Keychain Access

Once complete, the certificate should be listed in the Keychain Access application under the certificates section.



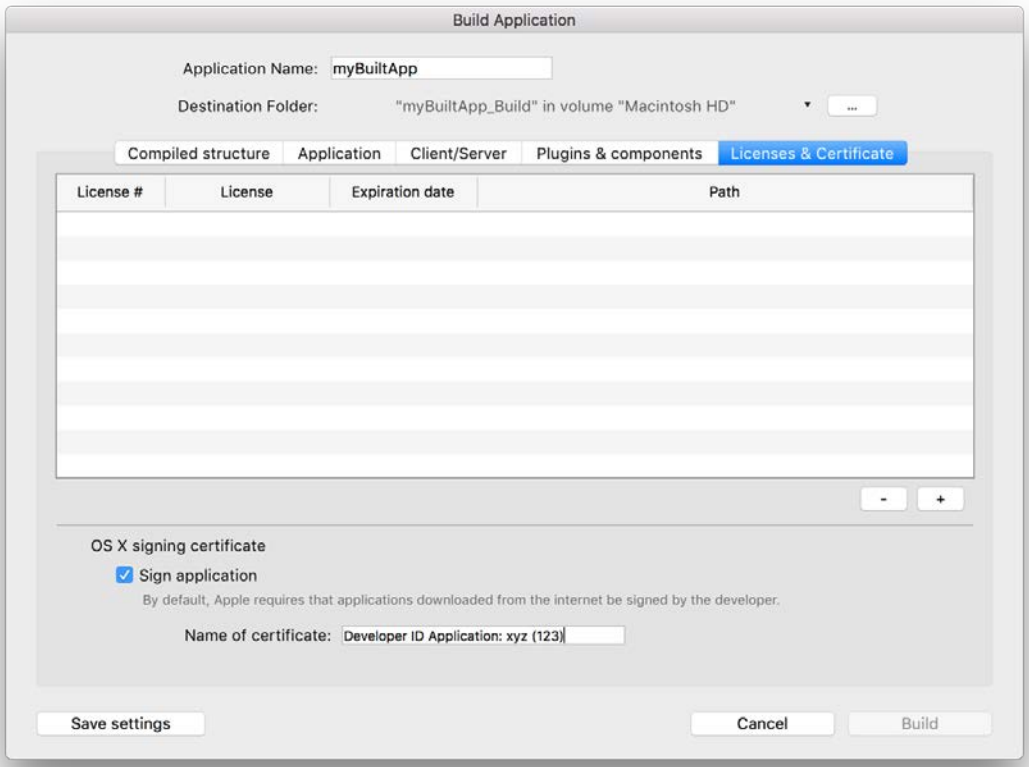
The name listed here is the name that should be used when identifying the certificate. For example, the name to be used for the certificate depicted in this screenshot is “Developer ID Application: 4D Inc (Q2R63U84YG)”

How to sign an app on MacOS

On MacOS a merged 4D application can be signed from within the BUILD APPLICATION process, or afterwards from the Terminal application. It is usually preferred to sign the application from within the BUILD APPLICATION process, however, if an error occurs during the signing process it is suggested to switch the process and perform the signing from Terminal in order to obtain more information about the failure.

Sign the application from 4D during BUILD APPLICATION

The Build Application dialog in 4D provides the developer the option of Code Signing their application during the build process. This feature is only available on the MacOS operating system. This option is exposed at the bottom of the License tab of the Build Application dialog:



The name of the certificate should match the name of the certificate provided by Apple. This name can also be observed from within the Keychain Access application as demonstrated in the ‘locating the certificate Keychain Access” section of this document.

What to do when an error occurs

When a problem occurs, an error will be presented in the build app log and the application will not be present in the destination folder. The error message may

look like this:

The data in the logs may look like this:

```
<Log>
  <MessageType>Error</MessageType>
  <Target>Application</Target>
  <CodeDesc>SIGNING_FAILED</CodeDesc>
  <CodeId>38</CodeId>
  <Message>Code signature failed</Message>
</Log>
```

To investigate this, the developer should rebuild the application with the code signing option disabled, and then attempt to code sign the application from the Terminal. Using the Terminal allows the developer to see the underlying error messages to better understand and correct the situation.

The following section outlines how to sign the application from the Terminal.

Sign the application from the Terminal

Sometimes it is useful to manually perform the signing process from Terminal. This is an important step when an error occurs with the built-in mechanism, but this can also be a useful step for developer who want a better understanding of how the signing process works. Whatever the case may be, this section covers the manual process of signing the macOS application from Terminal.

The general syntax used by 4D for signing is:

```
codesign -s "${nameCertificat}" -fvvvv "${PathSign}"
```

If the application is stored @ "/path/to/myBuiltApp.app/" and the developer certificate to use for signing is named "Developer ID Application: xyz (123)" then the command syntax would be:

```
codesign -s "Developer ID Application: xyz (123)" -fvvvv
/path/to/myBuiltApp.app/
```

Doing this should help reveal the underlying error that was encountered.

Some potential errors that could be found are:

- ./myBuiltApp.app: resource fork, Finder information, or similar detritus not allowed
- ./myBuiltApp.app: timestamps differ by 198 seconds - check the system clock
- Developer ID Application: xyz (123): ambiguous (matches "Developer ID Application: xyz (123)" and "Developer ID Installer: xyz (123)" in /Users/username/Library/Keychains/login.keychain-db)
- Agreeing to the Xcode/iOS license requires admin privileges, please run “sudo xcodebuild -license” and then retry this command.

Each of the errors above would require a different solution. Once the error received outside of 4D has been addressed it should be safe to attempt to build/sign from within 4D again.

How to sign an app on Windows

On Windows a merged 4D application must be signed outside of 4D because (unlike MacOS) there is no built-in mechanism to 4D that provides signing.

The following section outlines the process of signing the application on Windows. The examples in this section utilize the data points that were outlined earlier in this document when creating the signing certificate, those data points were:

Certificate location: C:\signing\cert.pfx
Certificate password: thePassword

However, if the developer used data points different from what is listed above then the correct information will need to be substituted in.

Installing signtool (Windows SDK)

In order to sign the application on Windows the developer must have already installed the signtool application. The signtool application is part of the Windows SDK which is available for free from Microsoft website at the following URL:

http://go.microsoft.com/fwlink/p/?linkid=84091

Once installed, signtool can be used from a Command Prompt window.

Sign the application from the Command Prompt

The following command is used to sign the application:

```
"C:\Program Files (x86)\Windows Kits\10\bin\x64\SignTool" sign /debug /f
c:\signing\cert.pfx /p thePassword /a C:\Path\to\MyBuiltServer.exe
```

If the signing is successful, the output may look like this:

```
The following certificates were considered:
    Issued to: 4D
    Issued by: 4D
    Expires:   Fri Aug 09 14:58:19 2019
    SHA1 hash: A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304

After EKU filter, 1 certs were left.
After expiry filter, 1 certs were left.
After Private Key filter, 1 certs were left.
The following certificate was selected:
    Issued to: 4D
    Issued by: 4D
    Expires:   Fri Aug 09 14:58:19 2019
    SHA1 hash: A6B2B0B3FECC6E4E1A3B562118F18BC82B63F304


The following additional certificates will be attached:
Done Adding Additional Store
Successfully signed: MyBuiltServer.exe

Number of files successfully Signed: 1
Number of warnings: 0
Number of errors: 0
```

The output above indicates that there was 1 successfully signed file.

Looking at the File Properties of the exe should reveal a Digital Signature tab that includes information about the signature and the company that signed the application.

Removing an existing signature

Application signing will fail if an existing signature exists on the application. The developer must remove the existing signature first, prior to signing.

In v16.4/v17.0 and higher, a tool called delcert is used internally during the build process to remove any existing signature. This produces a built application that is unsigned and ready for a signature to be added by the developer.

In v15 and prior, if the 4D Server is signed at the time of building the merged application the developer will need to remove the signature manually before signing it.

To remove the existing signature, an open source application named ‘delcert’ can be used. The application is available on the XDA forums.

Conclusion

This technical note described the general concepts of code signing an application on both Windows and MacOS. Two code signing techniques were presented and compared. Multiple advantages and disadvantages were laid out for the developer to ponder on. This information should allow the 4D developer the necessary knowledge needed to sign their applications on both MacOS and Windows.

Notes and external resources

The information regarding creating a self-signed certificate on Windows was up-to-date at the time writing according to the following document on Microsoft’s website:
https://docs.microsoft.com/en-us/windows/uwp/packaging/create-certificate-package-signing