by Arnold Nefkens

Code Sign Hell

How one solves code sign errors when having multiple Distribution Certificates with the same name.

Introduction

Code signing is the process all developers encounter while making releases for the Apple App Store or making builds for an in-house distribution. If the setup of your development environment is done right, it's no problem. Sometimes, though, there are issues which result in a code sign error. This article relates to the problem I encountered while making several releases of Apps for either the App Store or for the company's in-house account. Not only do I write about the problem, I have also found a solution / workaround

What is code signing?

According to tech note #2250 of the Apple Documentation for iOS development: "Code signing is the process by which your compiled iOS Application is sealed and identified as yours." In order to code sign we need signing certificates and provisioning profiles.

An app is signed when you want to:

- Run an app on a development device;
- Testing of the app using an Ad Hoc Distribution;
- Submitting your App to the App Store;
- Creating and installing an Enterprise build for In-House Distribution.

Problem

You have two Apple Developer Accounts, one for the App Store and one for In-House Apps. Both have the same company name in the portal and thus also in the distribution certificates. These names are set according to the name of your company as listed in the Chamber of Commerce. If you have both accounts setup on your development Mac, and you want to make a release of an app for either account, the following error occurs:

```
Code Sign error: Certificate identity 'iPhone Distribution:
<Name>.' appears more than once in the keychain. The
codesign tool requires there only be one
```

At least, that was the error we encountered when we had setup our development Macs and tried to make a release of an app for the App Store. First we thought that one of our Macs was not setup properly, so we tried on another Mac, one of our colleagues. And she had made a successful release just a week before. But now encountered the same error as I did.

Why was the build not successful? We do have the right certificate and keys, don't we? So the first step we took was removing the certificates and keys and starting over. We just deleted everything we had that was certificate related and started over. For now, we limited ourselves in setting up the App Store account. As you can probably guess, this was happening on a Friday afternoon, when we had to do a release in the App Store, and the deadline was that Friday afternoon. So we had setup our development Macs and made the release, no problem.

After a couple of days we needed to release an In-House app, and since we deleted the necessary certificates for our In-House account, we added it, made double sure that we had a different key-pair and downloaded the new certificate. Double-clicked on the "ios_distribution.cer" file and double-checked in the Keychain if the private key was attached to the certificate. It was. So we are good, right? Wrong, the moment we did the Archive of the in-house build, the darn error was back. "What is happening?", we asked ourselves. Next step we even resorted in deleting ourselves as team members to start fully from scratch. A step to be taken as a last resort, but did not work at all.

All these experiences started my investigation in the why and what is happening.

Use of the Terminal

When I talk to iOS developers, I have learned that a lot of developers do not really like the Terminal. Thanks to my experience as a Server Administrator the Terminal is an environment I prefer to do my maintenance of servers and clients. And in solving this problem that experience helped a lot. Without the terminal knowledge the problem would almost impossible to solve.

Analysis of the problem

What is happening, exactly? What does the error mean? How can we solve this? First we had some discussion about what the best approach was, and one of us mentioned: "Why not ask Apple to change the name of the portal?"

Short-term solution

Changing the name would be the easiest solution, if the names were different, then there would be only one iPhone Distribution Certificate with that specific name. So I contacted Apple, to ask them if it was possible to change the name of our Enterprise account. The answer Apple gave me was that I had to use different keychains.

So changing the name was not going to happen. That was by the way the only advice Apple gave us. But ok, lets create a workaround.

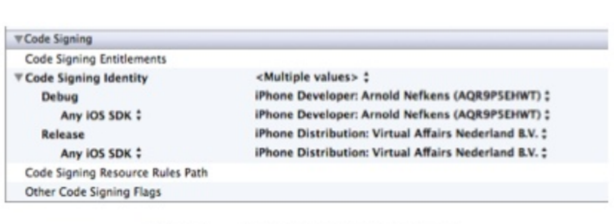
Investigating a better solution

Using different keychains is what Apple advised. So I created a new keychain, using the Keychain Access application, found in **/Applications/Utilities**. I placed in this keychain the needed distribution certificate and the key-pair. And we should be on our merry way, right? No, not really, the error was still there. What is happening? I have placed the certificate in a different keychain, why is there still a code sign error?

Then it hit me, I knew that the actual building of a project was handled using the **xcodebuild** UNIX executable. What if the code signing is also being handled using a UNIX application? Mac OS X is a UNIX environment. It turned out that the code signing is handled using the UNIX executable **codesign**. Well, if there is a UNIX application, most likely there is also a man page of the UNIX executable. I opened the Terminal and tried **man codesign**. It worked. There is a man page. Looking through the man page, one part caught my eye:

```
--keychain filename
              During signing, only search for the signing
identity in the keychain file specified. This can be used
to break any matching ties if you have multiple similarly-
named identities in several keychains on the user's search
list. Note that the standard keychain search path is still
consulted while constructing the certificate chain being
embedded in the signature.
```

There is thus an option to force the use of a keychain.



▾Code Signing	
Code Signing Entitlements	
▾Code Signing Identity	<Multiple values> ⁝
Debug	iPhone Developer: Arnold Nefkens (AQR9PSEHWT) ⁝
Any iOS SDK ⁝	iPhone Developer: Arnold Nefkens (AQR9PSEHWT) ⁝
Release	iPhone Distribution: Virtual Affairs Nederland B.V. ⁝
Any iOS SDK ⁝	iPhone Distribution: Virtual Affairs Nederland B.V. ⁝
Code Signing Resource Rules Path	
Other Code Signing Flags	

Figure 1 – Codesign options in Xcode.

In the Build Settings of an target you can set the "Other Code Signing Flags", these flags are the options and commands found in the man page of the **codesign** executable.

In here I set the correct flag of the needed keychain. I used the absolute path to the keychain. Keychains are by default located in your **/Users/<username>/Library/Keychains/**.

Using a relative path did not work, as a side note, in my experience it is better to use absolute paths when working in the terminal.

Still the code sign error. What now? I took a closer look in the man page, there is a note in there: "Note that the standard keychain search path is still consulted while constructing the certificate chain being embedded in the signature."

Search path? What search path? What if... What if... The keychains listed in the Keychain Access application are not the actual keychain files, but a list of keychains that are combined as the search path that the **codesign** consults? That would be interesting.

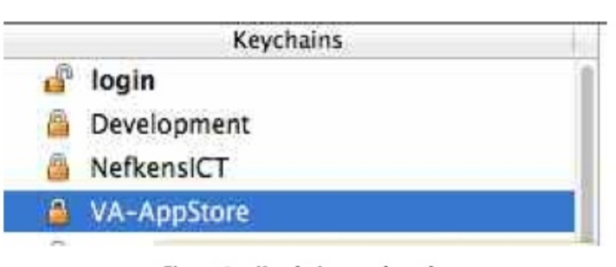
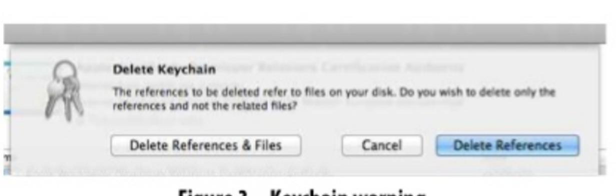


Figure 2 – Keychain search path.

How can we alter the search path in Keychain Access? It is a Mac application, so what comes naturally to me, is just delete a keychain in the list.

If you want to delete a keychain, just select the keychain and press backspace. The following message appears:



Delete Keychain

The references to be deleted refer to files on your disk. Do you wish to delete only the references and not the related files?

Figure 3 – Keychain warning.

As you can see the Keychains list is indeed just a list of references to the actual files on your disk.

Therefore is the list, as displayed in the Keychain Access application, the actual search path of keychains that are consulted when constructing the certificate chain which is being embedded in the signature.

Now a solution is forming. As long as we click on the default option "Delete References" we will be fine. And when we need to do a build, for either account, we just add the keychain reference we need and delete the other keychain reference.

Solution / Workaround

Essentially there is not really a solution, but it is more of a workaround. For me the following setup of certificates, private keys and such works best. Most of the harder code sign errors are gone.

Setup

Create a Keychain file called: "Development" or any other name that makes sense to you.

Right-Click in the Keychain list and select "New Keychain", store this keychain file in the same location as the other

keychain files, by default the location is **/Users/<username>/Library/Keychains/**.

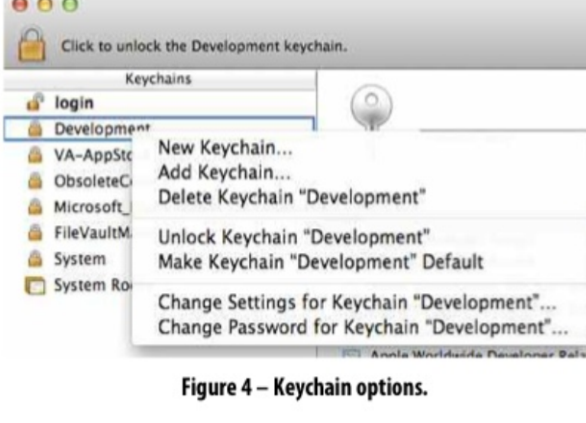
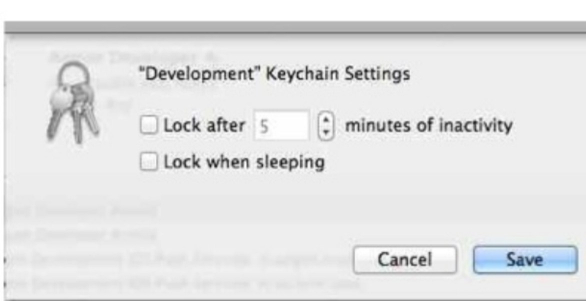

Click to unlock the Development keychain.
Keychains
login
Development
VA-AppSto
ObsoleteC
Microsoft_
FileVaultM
System
System Ro
New Keychain...
Add Keychain...
Delete Keychain "Development"
Unlock Keychain "Development"
Make Keychain "Development" Default
Change Settings for Keychain "Development"...
Change Password for Keychain "Development"...

Figure 4 – Keychain options.

In this keychain you store your development certificates and keys. You can just drag and drop the certificate and key-pairs from the Login Keychain, most likely the keychain in which these items are already stored. Keep in mind that you have to enter your password for the originating keychain when moving the certificate and the private key. If you want the keychain to have the same settings as in Figure 5, right-click on the "Development" keychain and select change "Change Settings for Keychain "Development".



"Development" Keychain Settings

☐ Lock after 5 minutes of inactivity

☐ Lock when sleeping

Figure 5 – Keychain settings.

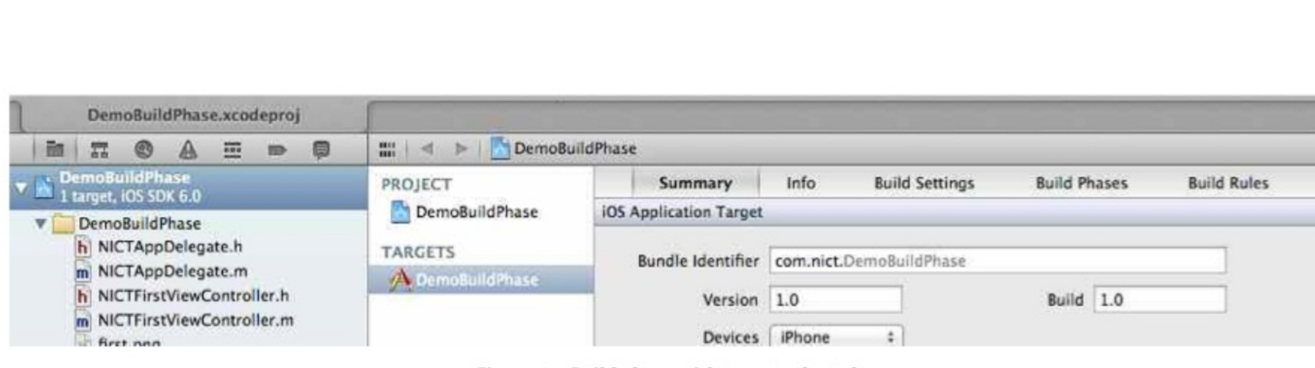
In this settings pane you can mimic the settings of the Login keychain.

Create another Keychain file named: "App Store" or any other name that makes sense to you.

In this keychain you place all the distribution certificates and necessary key-pairs for use in the App Store. Besides the distribution certificate I also place a copy of the Apple WWDR intermediate certificate, just to be sure.

Create the last keychain file, and name it: "In-House" or any other name that makes sense to you.

In this last keychain you place all of the needed certificates and key-pairs for using in the In-House account. Just to be sure I also place here another copy of the Apple WWDR intermediate certificate.



iOS Application Target

Bundle Identifier: com.nict.DemoBuildPhase

Version: 1.0 Build: 1.0

Devices: iPhone

Figure 6 – Build phase with target selected.

Workflow after setup

So before you do a build for the App Store, for example, you delete the reference of the In-House keychain file, using the Keychain Access application. And make sure you do have the App Store keychain reference listed in the Keychain Access application. Now the building should not be a problem anymore. Or at least the code sign error is gone. When you want to make a release for the In-House account you do the same, but now removing the App Store keychain and adding the In-House keychain references.

Bonus

Having to change the search path of the keychain manually is somewhat of a pain, and can be easily forgotten. For this reason I investigated the possibility of changing the search path using the terminal. This is also possible. There is a UNIX executable called: **security**. With this executable you have command-line access to keychains and the Security Framework of your Mac. I checked the man page of the executable: **man security**.

In the man page of **security** we find the following option, amongst a ton of other options:

Excerpt from man page of security

```
     list-keychains [-h] [-d user|system|common|dynamic] [-s
[keychain...]]
              Display or manipulate the keychain search list.

              -d user|system|common|dynamic
                          Use the specified preference domain.
              -s          Set the search list to the specified
keychains.
```

As a result of this command we cannot only list the search path in the Keychain Access via the command line, but we can also set it using the extra flag -s.

If we can change the search path using the command line, can we use that script line in Xcode during the build process? Yes, we can.

Part of Xcode is the capability of executing shell scripts before and/or after the building. We could use this feature for setting the search path for the project and setting them back after the build is done.

How to change to change the search path using Xcode, before the actual building?

When you select a target in Xcode, you can see the "Build Phases" and make changes to them.

Select the "Build Phase" and then click on the "Add Build Phase", then from the selection menu, select "Add Run Script". This will add the run script step to the build phases of your target.

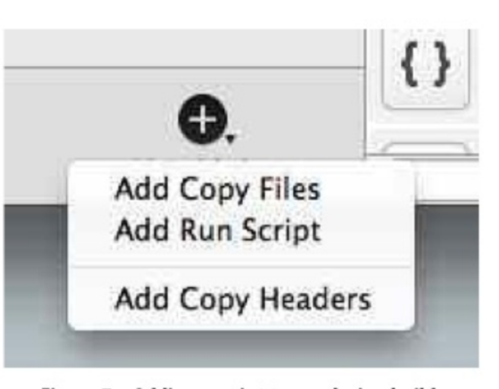


Figure 7 – Adding a script to run during build.

But beware the run script is added as the last step in the phases. So we have to move the step up, so the script step is executed before the compiling is being executed. We just have to drag the step up to the desired moment in the build phases.

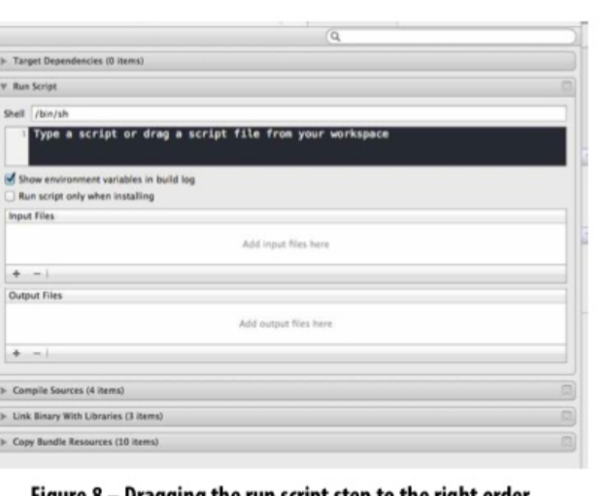

Target Dependencies (0 items)
Run Script
Shell /bin/sh
Type a script or drag a script file from your workspace
Show environment variables in build log
Run script only when installing
Input Files
Add input files here
Output Files
Add output files here
Compile Sources (4 items)
Link Binary With Libraries (3 items)
Copy Bundle Resources (10 items)

Figure 8 – Dragging the run script step to the right order.

As you can see in the screenshot in Figure 8, we now have to option to create a custom shell script that will be executed after the Target Dependencies, and before the actual Compiling. Now what do we put in the script?

We use the following line of shell script when I need to do In-House build:

```
/usr/bin/security -v list-keychains -s
/Users/<username>/Library/Keychains/VA-Enterprise.keychain
/Users/<username>/Library/Keychains/login.keychain
/Users/<username>/Library/Keychains/Development.keychain
```

I will explain the script line piece by piece; make sure that you write the script line to be in one line:

-v this will make the command verbose, it will generate more output in the logging always nice to see.

list-keychains is the command to tell the security CLI tool you want to manipulate the search path.

-s The flag to set the search path with the keychains you want.

Here you type the actual absolute paths to keychains you want to use. In the above example we have three keychains we want to include: VA-Enterprise.keychain, login.keychain and the Development.keychain.

We could add another build phase "Run Script" if we like at the end to reset the search path back to the keychains you would like to have. However if you add to all of your iOS projects the script step in setting the correct search path, there should be no need to have this extra step at the end.

Conclusion

If you have multiple distribution certificates with the same name, you can get rid of the code sign error, if you move the distribution certificates and key-pairs in separate keychains.

Keychains listed in the Keychain Access application are references to the actual files.

The list of keychains is in effect the search path codesign consults in creating the certificate chain, which is being embedded in the signature.

Using the security UNIX executable you can change the search path as a script step in Xcode.

As an alternative you can change the search path manually in the Keychain Access application, via deleting the reference of the keychain you do not need, and adding the reference of the keychain you do need.

An extra bonus can be, that if you are a freelancer and working for a number of clients, you can off-course also create keychain files for each client and store in here the client's distribution certificates and key-pairs.

For us, most of the code sign errors we now encounter are easy to fix. We have created the extra keychains and added the "Run Script" step to the "Build Phase" to change the search path in the security framework, so the **codesign** process has only access to the keychains needed for creating the certificate chain that will be embedded in the signature.

Bibliography and References

Apple Documentation Tech Note: 2250

http://developer.apple.com/library/ios/#technotes/tn2250/_index.html

Man page of the **codesign** UNIX executable

Man page of the **security** UNIX executable

MT

About The Author

Arnold Nefkens is an independent Apple Consultant, part of the Dutch ACN. In 2008, while attending the WWDC, he started with iPhone development. He has over 20 years of experience in Macintosh and Server administration. At the moment he works as an iOS Mobile Developer for Virtual Affairs, a full-service Internet agency situated in Amsterdam, Rotterdam and Sofia. He also works in his own company, Nefkens ICT, as a member of the ACN network.

http://www.virtual-affairs.com

http://www.nefkens-ict.nl

https://twitter.com/nefkensict