Resolving errSecInternalComponent errors during code signing

This thread has been locked by a moderator.

216

One code signing issue I commonly see, both here on DevForums and in my Day Job™ with DTS, is that the `codesign` command fails with `errSecInternalComponent`. This issue crops up in a wide variety of circumstances and the correct fix depends on the specific problem. This post is my attempt to clarify the potential causes of this error and help folks resolve it.

If you have any questions or comments about this, please start a new thread, tagging it with *Code Signing* so that I see it.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Resolving errSecInternalComponent errors during code signing

In some circumstances the `codesign` command might fail with the error `errSecInternalComponent`. For example:

```
% codesign -s "Apple Development" "MyTrue"
MyTrue: errSecInternalComponent
```

This typically affects folks who are signing code in a nonstandard environment, for example, when logged into a Mac via SSH or when signing code on a continuous integration (CI) server. This post explains how to resolve such issues, starting in the simplest case, signing from Terminal app, and then going on to discuss SSH and other contexts.

IMPORTANT Before going further, make sure you understand the difference between a digital identity and a certificate. See Certificate Signing Requests Explained for the details.

Test from Terminal

Code signing makes extensive use of the keychain and that's sensitive to the *execution context* in which it's running. So, the first step in resolving this problem is to test your code signing from Terminal. To start, log in to the Mac using the GUI.

Note If you don't have access to the GUI, see *Working without the GUI*, below.

Check that Keychain Access shows that your code signing identity's certificate is trusted. Select the certificate and look for a green checkmark with the text "This certificate is valid". If you see a red cross with an explanatory text like "… certificate is not trusted', follow the instructions in Fixing an untrusted code signing certificate.

In Terminal, run the `security` tool to check that your code signing identity is available:

```
% security find-identity -p codesigning

Policy: Code Signing
  Matching identities
  1) 4E587951B7052D8CBBB8B06325CD134D4CDA04977 "Apple Development: …"
     1 identities found

  Valid identities only
  1) 4E587951B7052D8CBBB8B06325CD134D4CDA04977 "Apple Development: …"
     1 valid identities found
```

If the identity is missing from the `Matching identities` list, you don't have a code signing identity to sign with. If you see your code signing identity's certificate in the keychain, it's possible that you're missing its private key. See Certificate Signing Requests Explained for more about that issue.

If the identity is shown in the `Matching identities` list but not in the `Valid identities only` list, see Fixing an untrusted code signing certificate.

This example assumes that you're testing with an Apple Development signing identity. If you're using something else, you'll see a different identity name in this list. Use that identity name in the `codesign` command below.

Still in Terminal, make copy of the `true` tool to use for this test:

```
% cp "/usr/bin/true" "MyTrue"
```

Try to sign it:

```
% codesign -s "Apple Development" -f "MyTrue"
MyTrue: replacing existing signature
```

The `-f` flag tells `codesign` to replace the existing signature.

This command may display one or more keychain dialogs but, once you respond to those, it should correctly sign `MyTrue`. If it doesn't, something else is afoot and I recommend that you seek dedicated help per the start of this post.

Eliminate keychain alerts

When you signed your code in the previous section, you may have seen one of two different types of keychain alerts:

- Keychain unlock dialog
- Access control list (ACL) dialog

The keychain unlock dialog looks like this:

```
codesign wants to use the … keychain.

Please enter the keychain password.

Password: [                          ]

                        [Cancel] [(OK)]
```

The keychain containing your code signing identity is locked, and you must enter the keychain password to unlock it. You rarely see this dialog when logged in via the GUI because the system automatically unlocks the login keychain when you log in. However, the underlying cause of this alert will become relevant in the next section, when you log in via SSH.

The ACL dialog looks like this:

```
codesign wants to sign using key … in your keychain.

To allow this, enter the … keychain password.

Password: [                          ]

[Always Allow]          [Deny] [(Allow)]
```

The ACL for the your code signing identity's private key prevents `codesign` from using the private key without your explicit approval. If you enter your password and click Allow, `codesign` can use the private key once. If you click Always Allow, the system adds `codesign` to the private key's ACL so that it doesn't have to ask again.

To avoid this alert in the future, enter your keychain password and click Always Allow. Now repeat the `codesign` command from the previous section. It will sign the code without presenting any dialogs.

Test over SSH

Once you can sign your code in Terminal without seeing any dialogs, it's time to repeat that process over SSH. To start, log out of the GUI and then log in via SSH.

If you're testing on a CI system, log in to that system by running `ssh` from Terminal on your Mac. If you want to test on your local Mac, choose one of these options

- If you have a second Mac, log in to that second Mac using the GUI, launch Terminal, and then run `ssh` to log in to your main Mac from there.
- If you have an iPad, use a third-party iPad SSH app to log in to your main Mac over SSH.
- Use a virtualisation app to run a macOS guest that you can treat like your CI system.

Once you're logged in over SSH, repeat the signing command from the earlier section:

```
% codesign -s "Apple Development" -f "MyTrue"
MyTrue: replacing existing signature
MyTrue: errSecInternalComponent
```

This fails because:

- The system locked the keychain when you logged out of the GUI.
- Logging in via SSH does not unlock the keychain.
- When `codesign` tries to use your code signing identity, the system attempts to present the keychain unlock dialog.
- That fails because you're logged in via SSH and thus don't have access to the GUI.
- The system returns the `errSecInternalComponent` error to `codesign`, which reports it to you.

To fix this, unlock your keychain using the `security` tool:

```
% security unlock-keychain
password to unlock default: KEYCHAIN_PASSWORD
% codesign -s "Apple Development" -f "MyTrue"
MyTrue: replacing existing signature
```

IMPORTANT This assumes that your code signing identity is in your login keychain. If it's in some other keychain, read the `security` man page to learn how to unlock a specific keychain.

Best practice is to store both parts of your code signing identity (the certificate and the private key) in the same keychain. If you split the identity across two keychains, unlock the keychain that contains the private key.

Test your CI job

Once you have everything working on your CI system over SSH, try running exactly the same commands in your CI job. If your CI system manages user contexts correctly, those commands should just work. If they don't, discuss this with your CI vendor.

Note macOS has a complex execution context model. For background on this, see the *Execution Contexts* section of Technote 2083 Daemons and Agents. Some CI systems do not correctly establish a user context when running jobs. For example, they might switch the traditional Unix execution context — the EUID, RUID, and so on — but not the security context. This mixed execution context causes problems for the keychain, which relies on the security context.

Avoid doing code signing work as root. Some folks run *everything* as root because they think it'll avoid problems. When working with the keychain the opposite is true: Running as root often causes more problems than it solves. These problems are most likely to show up when you use `sudo`, which creates a mixed execution context.

Working without the GUI

The instructions above assume you have access to the GUI so that you can test and resolve issues using GUI tools like Keychain Access. However, many CI systems don't give you access to the GUI, at best you might have interactive access using SSH.

Note If you CI system allows remote access using a screen sharing protocol, use that rather than messing around with the instructions here.

If you don't have access to the GUI of the machine on which you're signing code, there are three issues to deal with:

- Avoiding the keychain unlock dialog
- Avoiding the ACL dialog
- Investigating an untrusted code signing certificate issue

To unlock the keychain, use the `unlock-keychain` subcommand of the `security` tool, discussed in the *Test over SSH* section.

When logged in with the GUI, you can respond to ACL dialog by clicking Always Allow. This prevents that dialog showing up again. However, if you don't have GUI access there's no way to click that button. To get around this, import your signing identity and set its ACL to allow `codesign` to use it without extra authorisation. To do this, first unlock the keychain:

```
% security unlock-keychain
password to unlock default: KEYCHAIN_PASSWORD
```

Then use the `security` tool to import the PKCS#12 file:

```
% security import IDENTITY.p12 -T /usr/bin/codesign -P P12_PASSWORD
1 identity imported.
```

Note the `-T` option, which adds `codesign` to the private key's ACL.

Finally, modify set the partition list to allow access by Apple code:

```
% security set-key-partition-list -S "apple:" -l "Apple Development: …"
```

This example assumes you're using an Apple Development signing identity to test with. If you're using something else, replace `Apple Development: …` with that identity name.

Finally, investigating an untrusted code signing certificate issue remotely is quite challenging. Your best option here is to set up a local test environment, run your investigation in that environment, and then apply the results to your CI environment.

There are two good choices for your local test environment:

- Use a virtualisation app to create a 'clean' macOS guest, one that's never seen your code signing setup before.
- Create a new local account and do your testing there.

The first option is best because you can easily restore your VM to a clean state between tests.

When running through the process described in Fixing an untrusted code signing certificate, you might end up performing two different remedial actions:

- Importing an intermediate
- Reseting trust settings.

Once you understand these remediations, you need to apply them to your CI system. The first one is easy: To import an intermediate, run `security` with the `import` subcommand:

```
% security import INTERMEDIATE.cer
1 certificate imported.
```

Resetting trust settings is more of a challenge. It's probably possible to do this with the `security` tool but, honestly, if you think that your CI system has messed up trust settings it's easiest to throw it away and start again from scratch.

Revision History

- *2022-08-12* Extended the `unlock-keychain` explanation to cover the split identity issue.
- *2022-08-11* First posted.

Code Signing

Reply

Posted 2 months ago by eskimo

Add a Comment