# Troubleshooting -34018 Keychain Errors

This thread has been locked. Questions are automatically locked after two months of inactivity, or sooner if deemed necessary by a moderator.

1.9k

Recently I've had a couple of folks ping me about debugging reproducible -34018 errors when using the keychain. Pasted in below is my advice on that topic. If you have any feedback about this, or you are having this problem and can't fix it using these instructions, please open a new thread in Core OS > Security and post the details there.

Share and Enjoy — Quinn "The Eskimo!" Apple Developer Relations, Developer Technical Support, Core OS/Hardware

```
let myEmail = "eskimo" + "1" + "@apple.com"
```

Change history:

- 11 Mar 2019 — First posted.
- 13 Mar 2019 — Added a clarification about query dictionaries.

## Troubleshooting -34018 Keychain Errors

*Learn how to resolve -34018 errors from the keychain.*

Error -34018 translates to

```
errSecMissingEntitlement
```

. This error means that your app is trying to use a keychain access group for which it does not have entitlements. See Set Your App's Access Groups for information on how the system determines the list of keychain access groups that you have access to.
**Note** On macOS, the advice in this article only applies if you're using iCloud Keychain. If you're using the traditional file-based keychain, you should never see error -34018.

There are two common scenarios for this error:

- Reproducible. The problem happens every time you run the code, typically during development but possibly in other contexts, like after submitting your app to TestFlight, or during enterprise deployment.
- Intermittent. The problems shows up very occasionally on user devices in the field but is otherwise hard to reproduce.

If you're seeing this problem intermittently, read the suggestions in Error -34018 errSecMissingEntitlement. In contrast, if the problem is reproducible, read the rest of this article for advice on how to debug it.

## Check Your Entitlements

The first step in troubleshooting this problem is to check your app's entitlements. To start, use the

```
codesign
```

tool to dump the entitlements:

```
$ codesign -d --entitlements :- /path/to/your.app
```

**IMPORTANT** Dump the entitlements of your built app, not the

```
.entitlements
```

file you see in your Xcode project. The

```
.entitlements
```

file is an important input to Xcode's code signing machinery, but it is not what the system uses to determine your app's entitlements.
You should see something like this:

```
$ codesign -d --entitlements :- TestKeychain.app
…
<plist version="1.0">
<dict>
    <key>com.apple.developer.team-identifier</key>
    <string>SKMME9E2Y8</string>
    <key>application-identifier</key>
    <string>SKMME9E2Y8.com.example.apple-samplecode.testkeychain.app</string>
    <key>keychain-access-groups</key>
    <array>
        <string>SKMME9E2Y8.example.apple-samplecode.testkeychain.app</string>
        <string>SKMME9E2Y8.example.apple-samplecode.testkeychain.shared</string>
        …
    </array>
    <key>com.apple.security.application-groups</key>
    <array>
        <string>group.com.example.apple-samplecode.testkeychain</string>
        …
    </array>
    …
</dict>
</plist>
```

In this output you'll see the following:

- The

  ```
  com.apple.developer.team-identifier
  ```

  property is your Team ID.
- The

  ```
  application-identifier
  ```

  (

  ```
  com.apple.application-identifier
  ```

  on macOS) is your App ID, that is, your App ID prefix (in most cases this is your Team ID) followed by your bundle ID.
- ```
  keychain-access-groups
  ```

  , if present, starts with your App ID and then lists any other keychain access groups you use.
- ```
  com.apple.security.application-groups
  ```

  , if present, lists the shared app groups you use.

As discussed in Set Your App's Access Groups, the system uses the last three entitlements to form of list of keychain access groups that you're app is entitled to use. Your keychain access group must appear in one of these entitlements. If it's not there, read Technote 2415 Entitlements Troubleshooting for advice on how to fix that.

## Check Your Keychain Calls

Once you've confirmed that your app has the entitlements to access the expected keychain access group, the next step is to confirm that you're passing the correct access group to the keychain API. To do this, set a breakpoint on your keychain calls. For example, in the following code snippet you would set a breakpoint on the last line:

```
let query: NSDictionary = [
    kSecClass: kSecClassGenericPassword,
    kSecAttrService: "myService",
    kSecAttrAccount: username,
    kSecAttrAccessGroup: "SKMME9E2Y8.example.apple-samplecode.testkeychain.shared",
    kSecMatchLimit: kSecMatchLimitAll,
    kSecReturnData: true,
]
var copyResult: CFTypeRef? = nil
let err = SecItemCopyMatching(query, &copyResult)
```

**Note** While I'm using

```
SecItemCopyMatching
```

in this example, you're unlikely to get an

```
errSecMissingEntitlement
```

back from that routine because the parameter you pass in is a query dictionary, and a bogus

```
kSecAttrAccessGroup
```

in a query dictionary will simply cause the query to not match. However, the general approach described here — setting a breakpoint on a keychain call and inspecting the

```
kSecAttrAccessGroup
```

value in the parameter dictionary or dictionaries — is relevant to all keychain calls.
When you hit the breakpoint, use the debugger to print the

```
query
```

dictionary:

```
(lldb) p query
(NSDictionary) $R4 = 0x0000600000fedb00 6 key/value pairs {
  …
  [5] = {
    key = 0x0000000111838958 "agrp"
    value = "SKMME9E2Y8.example.apple-samplecode.testkeychain.shared"
  }
}
```

Here the

```
agrp
```

attribute holds the keychain access group being searched (

```
agrp
```

is the value of

```
kSecAttrAccessGroup
```

). It must either be not present, in which case you get the default behaviour discussed below, or included in the list of entitlements as determined by the previous section. If it's some other value, trace the origin of that bad value and correct it.
**Note** If the

```
kSecAttrAccessGroup
```

attribute is missing, you will see one of three behaviours:

- For query dictionaries, like the one passed to

  ```
  SecItemCopyMatching
  ```

  , the system interprets a missing value as a wildcard, that is, the query will match an item in any access group.
- For

  ```
  SecItemAdd
  ```

  , the system will use your app's default keychain access group, that is, the first entry in the list of entitlements as determined by the previous section.
- For the second parameter of

  ```
  SecItemUpdate
  ```

  , a missing value indicates that it should not change the keychain access group attribute.

Security

Asked 1 year ago by eskimo

Reply to this question