Calling Security Framework from Swift

This thread has been locked by a moderator.

346

I spend *way* too much time interacting with the Security framework. Most Security framework APIs are kinda clunky to call from Swift, largely because they use Core Foundation conventions. However, I see a lot of folks working much harder than they should to call these APIs. This post contains two tips to make your life easier.

Many Security framework APIs work in terms of `CFDictionary`. I regularly see folks create these dictionaries like so:

```
let query: [String: Any] = [
    kSecClass as String: kSecClassKey,
    kSecMatchLimit as String: kSecMatchLimitAll,
    kSecReturnRef as String: true,
]
var copyResult: CFTypeRef?
let status = SecItemCopyMatching(query as CFDictionary, &copyResult)
```

That's much too hard. Try this instead:

```
let query = [
    kSecClass: kSecClassKey,
    kSecMatchLimit: kSecMatchLimitAll,
    kSecReturnRef: true,
] as NSDictionary
var copyResult: CFTypeRef?
let status = SecItemCopyMatching(query, &copyResult)
```

Much nicer.

Security framework APIs have a wide variety of ways to indicate an error:

- Some routines return an `OSStatus` and that's it.
- Some routines return an `OSStatus` and an 'out' value.
- Some routines return a pointer, where `nil` indicates an error.
- Some routines return a pointer, where `nil` indicates an error, with a `CFError` 'out' value.
- Some routines return a Boolean, where false indicates an error, with a `CFError` 'out' value.

In Swift you really just want to call the API and have it throw. The code pasted in at the end of this post helps with that. It declares a bunch of overloaded `secCall(…)` functions, one for each of the cases outlined above. It takes code like this:

```
let query = [
    kSecClass: kSecClassKey,
    kSecMatchLimit: kSecMatchLimitAll,
    kSecReturnRef: true,
] as NSDictionary
var copyResult: CFTypeRef? = nil
let err = SecItemCopyMatching(query, &copyResult)
guard err == errSecSuccess else {
    throw NSError(domain: NSOSStatusErrorDomain, code: Int(err))
}
let keys = copyResult! as! [SecKey]
```

and turns it into this:

```
let query = [
    kSecClass: kSecClassKey,
    kSecMatchLimit: kSecMatchLimitAll,
    kSecReturnRef: true,
] as NSDictionary
let keys = try secCall { SecItemCopyMatching(query, $0) } as! [SecKey]
```

Still not exactly pretty, but definitely an improvement.

Share and Enjoy

—

Quinn "The Eskimo!" @ Developer Technical Support @ Apple

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

```
/// Calls a Security framework function, throwing if it returns an error.
///
/// For example, the `SecACLRemove` function has a signature like this:
///
/// ```
/// func SecACLRemove(…) -> OSStatus
/// ```
///
/// and so you call it like this:
///
/// ```
/// try secCall { SecACLRemove(acl) }
/// ```
///
/// - Parameter body: A function that returns an `OSStatus` value.
/// - Throws: If `body` returns anything other than `errSecSuccess`.

func secCall(_ body: () -> OSStatus) throws {
    let err = body()
    guard err == errSecSuccess else {
        throw NSError(domain: NSOSStatusErrorDomain, code: Int(err), userInfo: nil)
    }
}

/// Calls a Security framework function that returns an error and a value indirectly.
///
/// For example, the `SecItemCopyMatching` function has a signature like this:
///
/// ```
/// func SecItemCopyMatching(…, _ result: UnsafeMutablePointer<CFTypeRef?>?) -> OSStatus
/// ```
///
/// and so you call it like this:
///
/// ```
/// let keys = try secCall { SecItemCopyMatching([
///     kSecClass: kSecClassKey,
///     kSecMatchLimit: kSecMatchLimitAll,
///     kSecReturnRef: true,
/// ] as NSDictionary, $0) }
/// ```
///
/// - Parameter body: A function that returns an `OSStatus` value and takes a
/// 'out' pointer to return the result indirectly.
/// - Throws: If `body` returns anything other than `errSecSuccess`.
/// - Returns: The value returned indirectly by the function.

func secCall<Result>(_ body: (_ resultPtr: UnsafeMutablePointer<Result?>) -> OSStatus) throws -> Result {
    var result: Result? = nil
    let err = body(&result)
    guard err == errSecSuccess else {
        throw NSError(domain: NSOSStatusErrorDomain, code: Int(err), userInfo: nil)
    }
    return result!
}

/// Calls a Security framework function that returns `nil` on error.
///
/// For example, the `SecKeyCopyPublicKey` function has a signature like this:
///
/// ```
/// func SecKeyCopyPublicKey(…) -> SecKey?
/// ```
///
/// and so you call it like this:
///
/// ```
/// let publicKey = try secCall { SecKeyCopyPublicKey(privateKey) }
/// ```
///
/// - Parameters:
///   - code: An `OSStatus` value to throw if there's an error; defaults to `errSecParam`.
///   - body: A function that returns a value, or `nil` if there's an error.
/// - Throws: If `body` returns `nil`.
/// - Returns: On success, the non-`nil` value returned by `body`.

func secCall<Result>(_ code: Int = Int(errSecParam), _ body: () -> Result?) throws -> Result {
    guard let result = body() else {
        throw NSError(domain: NSOSStatusErrorDomain, code: code, userInfo: nil)
    }
    return result
}

/// Calls a Security framework function that returns `nil` on error along with a `CFError` indirectly.
///
/// For example, the `SecKeyCreateDecryptedData` function has a signature like this:
///
/// ```
/// func SecKeyCreateDecryptedData(…, _ error: UnsafeMutablePointer<Unmanaged<CFError>?>?) -> CFData?
/// ```
///
/// and so you call it like this:
///
/// ```
/// let plainText = try secCall { SecKeyCreateDecryptedData(privateKey, .rsaEncryptionPKCS1, cypherText, $0) }
/// ```
///
/// - Parameter body: A function that returns a value, which returns `nil` if
/// there's an error and, in that case, places a `CFError` value in the 'out' parameter.
/// - Throws: If `body` returns `nil`.
/// - Returns: On success, the non-`nil` value returned by `body`.

func secCall<Result>(_ body: (_ resultPtr: UnsafeMutablePointer<Unmanaged<CFError>?>) -> Result?) throws -> Result {
    var errorQ: Unmanaged<CFError>? = nil
    guard let result = body(&errorQ) else {
        throw errorQ!.takeRetainedValue() as Error
    }
    return result
}

/// Calls a Security framework function that returns false on error along with a `CFError` indirectly.
///
/// For example, the `SecKeyVerifySignature` function has a signature like this:
///
/// ```
/// func SecKeyVerifySignature(…, _ error: UnsafeMutablePointer<Unmanaged<CFError>?>?) -> Bool
/// ```
///
/// and so you call it like this:
///
/// ```
/// try secCall { SecKeyVerifySignature(publicKey, .ecdsaSignatureMessageX962SHA1, signedData, signature, $0) }
/// ```
///
/// - Parameter body: A function that returns a false if there's an error and,
/// in that case, places a `CFError` value in the 'out' parameter.
/// - Throws: If `body` returns false.

func secCall(_ body: (_ resultPtr: UnsafeMutablePointer<Unmanaged<CFError>?>) -> Bool) throws {
    var errorQ: Unmanaged<CFError>? = nil
    guard !body(&errorQ) else {
        throw errorQ!.takeRetainedValue() as Error
    }
}
```

Swift Security

Reply

Posted 2 months ago by eskimo

Add a Comment