Sample Code

Logging a User into Your App with Face ID or Touch ID

Supplement your own authentication scheme with biometric authentication, making it easy for users to access sensitive parts of your app.

Download

iOS 15.5+ | iPadOS 15.5+ | Xcode 13.1+

Overview

Users love Touch ID and Face ID because these authentication mechanisms let them access their devices securely, with minimal effort. When you adopt the LocalAuthentication framework, you streamline the user authentication experience in the typical case, while providing a fallback option for when biometrics aren't available.

Set the Face ID Usage Description

In any project that uses biometrics, include the NSFaceIDUsageDescription key in your app's `Info.plist` file. Without this key, the system won't allow your app to use Face ID. The value for this key is a string that the system presents to the user the first time your app attempts to use Face ID. The string should clearly explain why your app needs access to this authentication mechanism. The system doesn't require a comparable usage description for Touch ID.

Create and Configure a Context

You perform biometric authentication in your app using an LAContext instance, which brokers interaction between your app and the Secure Enclave. Begin by creating a context:

```
var context = LAContext()
```

You can customize the messaging used by the context to guide the user through the flow. For example, you can set a custom message for the Cancel button that appears in various alert views:

```
context.localizedCancelTitle = "Enter Username/Password"
```

This helps the user understand that when they tap the button, they'll be reverting to your normal authentication procedure.

Test Policy Availability

Before attempting to authenticate, test to make sure that you actually have the ability to do so by calling the canEvaluatePolicy(_:error:) method:

```
var error: NSError?
guard context.canEvaluatePolicy(.deviceOwnerAuthentication, error: &error) else {
    print(error?.localizedDescription ?? "Can't evaluate policy")

    // Fall back to a asking for username and password.
    // ...
    return
}
```

Choose a value from the LAPolicy enumeration for which to test. The policy controls how the authentication behaves. For example, the deviceOwnerAuthentication policy used in this sample indicates that reverting to a passcode is allowed when biometrics fails or is unavailable. Alternatively, you can indicate the deviceOwnerAuthenticationWithBiometrics policy, which doesn't allow reverting to the device passcode.

Evaluate a Policy

When you're ready to authenticate, call the evaluatePolicy(_:localizedReason:reply:) method, using the same policy you already tested:

```
Task {
    do {
        try await context.evaluatePolicy(.deviceOwnerAuthentication, localizedReasor
        state = .loggedin
    } catch let error {
        print(error.localizedDescription)

        // Fall back to a asking for username and password.
        // ...
    }
}
```

For Touch ID, or when the user enters a passcode, the system displays the reason for authenticating that you provided in the method call. It's important to provide a clear explanation, localized for any regions in which you operate, of why your app is asking the user to authenticate. The name of your app already appears before the reason you give, so you don't need to include that in your message.

Optionally, Adjust Your User Interface to Accommodate Face ID

Aside from the biometric scanning operation itself, Face ID exhibits an important difference from Touch ID. When your app tries to use Touch ID, a system message asks the user to present a finger to scan. The user has time to think about and possibly abort the operation by canceling the prompt. When your app invokes Face ID, the device begins scanning the user's face right away. Users don't get a final opportunity to cancel. To accommodate this behavioral difference, you might want to provide a different UI, depending on the kind of biometrics available on the device.

The sample app provides a different UI by including a text label with a message that warns Face ID users that tapping the button results in an immediate Face ID scan. The app hides the label by default, revealing it only for users with Face ID who are currently logged out. Users with Touch ID (or no biometry at all) don't need the message because they can cancel the operation before the scan actually takes place.

You test what kind of biometry the device supports by reading the context's biometryType parameter:

```
faceIDLabel.isHidden = (state == .loggedin) || (context.biometryType != .faceID)
```

This parameter only contains a meaningful value after you run the canEvaluatePolicy(_:error:) method on the context at least once.

Provide a Fallback Alternative to Biometrics

For various reasons, authentication sometimes fails or is unavailable:

- The user’s device doesn’t have Touch ID or Face ID.
- The user isn’t enrolled in biometrics, or doesn’t have a passcode set.
- The user cancels the operation.
- Touch ID or Face ID fails to recognize the user.
- You’ve previously invalidated the context with a call to the invalidate() method.

For a complete list of possible error conditions, see LAError.Code.

This sample app doesn’t implement alternative authentication. In a real app, if you encounter a local authentication error, fall back to your own authentication scheme, like asking for a username and password. Use biometrics as a supplement to something you’re already doing. Don’t depend on biometrics as your only authentication option.

See Also

Essentials

{} Accessing Keychain Items with Face ID or Touch ID

Protect a keychain item with biometric authentication.