XCI-Provider SDK for iOS

This guide is for developers integrating Project Verify into their iOS applications.

Background

Project Verify is a joint undertaking of the Mobile Authentication Taskforce. The taskforce provides customers with the ability to use their mobile phone number to sign into apps.

Installation

There are three ways to integrate ProjectVerify (and its dependency `AppAuth`) to your project: CocoaPods, Carthage and manually.

Dependencies

This SDK relies on *AppAuth* as an Open Id Connect client.
For more information about AppAuth, see the repository here.

Pre-release Git Access

While the SDK is under development, we recommend maintaining the Provider SDK source code as a git submodule. If that is not possible, download the source here and place it in your project directory.

```
git submodule add https://git.xcijv.net/sp-sdk/sp-sdk-ios
```

CocoaPods

During development, include the ProjectVerifyLogin SDK in your project as a development Cocoapod. After you place the source code in your repository, add the following to your Podfile.

```
pod 'CarriersSharedAPI', path: '{your-relative-path}/CarriersSharedAPI.podspec'
```

Then run `pod install`. This adds the local source as well as `AppAuth` to your application's workspace.

Carthage

More information coming soon - *Carthage* may be supported in the future.

Manual

The following steps show how to add ProjectVerifyLogin SDK to your project manually. For a project that links ProjectVerifyLogin manually, see the example SocialApp.

1 - Retrieve the source for `ProjectVerifyLogin`. We recommend adding it as a submodule.

2 - After you clone the repository, run `git submodule update --init --recursive` to recursively clone `AppAuth`. If you've been unable to use submodules, you must separately clone `AppAuth` and set your working copy to the release tag you would like to target. This SDK currently supports version 0.95.0.

3 - After you add the source via submodule or manually, then add `CarriersSharedAPI.xcodeproj` to your application's Xcode project.

4 - Having added the project, confirm that their deployment targets are less than or equal to your deployment target.

5 - Next, ensure that `AppAuth` is linked to `CarriersSharedAPI` and also included as a "Target Dependency" in the `CarrierSharedAPI` build phases.

6 - View your project's "Embedded Binaries" under your project's "General" panel. Add both `AppAuth` and `CarriersSharedAPI` frameworks here. Be sure to select the corresponding framework for the platform you're targeting (the iOS framework for an iOS target).

That's it! Build and run to ensure that everything is working correctly.

Integration

To integrate Project Verify into your iOS application, configure your Info.plist and instantiate Project Verify in your application delegate.

Configure your Info.plist

Retrieve your application's client id from the project verify dashboard. Add the following keys to your application's Info.plist:

```
<key>ProjectVerifyClientId</key>
<string>{your application's client id}</string>
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleTypeRole</key>
        <string>Editor</string>
        <key>CFBundleURLName</key>
        <string>{your bundle id}</string>
        <key>CFBundleURLSchemes</key>
        <array>
            <string>{your application's client id}</string>
        </array>
    </dict>
</array>
```

For examples of how to configure the property list, see the Social App, and Bank App.

Custom URL Schemes

If you would like to use universal links for your redirect scheme, it is possible to configure a custom URL scheme and a custom url host.

The following keys are made available for you to customize the structure of the redirect url:

```
<key>ProjectVerifyCustomScheme</key>
<string>{your application's custom scheme}</string>
<key>ProjectVerifyCustomHost</key>
<string>{your application's custom host}</string>
```

Note: for schemes other than `https`, you must add the scheme to your application's

`CFBundleURLTypes` list.

For an example of how the configure property list for these custom keys, see the Photo App.

Redirect urls will require the universal links to route the following paths to the application: `/authorize` and `/discoveryui`.
For more information about universal links, read Apple's documentation on the topic.

A Note on Custom URL Schemes

It is strongly recommended that you use either the default url scheme (your Project Verify client Id) or a universal linking scheme to support Project Verify Redirects. Other url schemes may be owned by other applications and can introduce unexpected behavior. For more information, view Apple's documentation on defining a custom url scheme.

Instantiate Project Verify in your Application Delegate

First you must configure your application delegate to support project verify:

```
import CarriersSharedAPI

class AppDelegate: UIResponder, UIApplicationDelegate {
    func application(_ application: UIApplication,
     didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: A

        ProjectVerifyAppDelegate.shared.application(
            application,
            didFinishLaunchingWithOptions: launchOptions
        )

        // Perform additional application setup.

        return true
    }

    func application(_ app: UIApplication,
                     open url: URL,
                     options: [UIApplication.OpenURLOptionsKey: Any] = [:]) -> Bool

        guard !ProjectVerifyAppDelegate.shared.application(app, open: url, options:
            return true
        }
        // Perform any other URL processing your app may need to perform.
        return true
    }
}
```

Using the ProjectVerifyAuthorizationButton

The SDK provides a branded button that automatically handles ProjectVerify authorization. Pass the code and associated identifiers to your secure server to complete the token request flow.

```
import CarriersSharedAPI

class LoginViewController {
    let projectVerifyButton = ProjectVerifyAuthorizationButton()

    override func viewDidLoad() {
        super.viewDidLoad()

        let scopes: [Scope] = [.profile, .email]
        projectVerifyButton.scopes = scopes
        projectVerifyButton.delegate = self
    }
}

extension LoginViewController: ProjectVerifyAuthorizeButtonDelegate {

    func buttonWillBeginAuthorizing(_ button: ProjectVerifyAuthorizeButton) {
        // perform any ui updates like showing an activity indicator.
    }

    func buttonDidFinish(
        _ button: ProjectVerifyAuthorizeButton,
        withResult result: AuthorizationResult) {

        // handle the outcome of the request:
        switch result {
        case .code(let authorizedResponse):
            let code = authorizedResponse.code
            let mcc = authorizedResponse.mcc
            let mnc = authorizedResponse.mnc
            // pass these identifiers to your secure server to perform a token requ
        case .error:
            // handle the error case appropriately
        case .cancelled:
            // perform any work required when the user cancels
        }
    }
}
```

Perform An Authorization Request Manually

If you require a more hands-on approach, you can use the `AuthorizationService` to request an authorization
code. Pass the code and associated identifiers to your secure server to complete the token request flow.

```
import CarriersSharedAPI

class LoginViewController {

    let authService = AuthorizationService()

    func loginWithProjectVerify() {
        // in response to some UI, perform an authorization using the Authorization
        let scopes: [Scope] = [.profile, .email]
        authService.connectWithProjectVerify(
            scopes: scopes,
            fromViewController: self) { result in

            switch result {
            case .code(let authorizedResponse):
                let code = authorizedResponse.code
                let mcc = authorizedResponse.mcc
                let mnc = authorizedResponse.mnc
                // pass these identifiers to your secure server to perform a token
            case .error:
                // handle the error case appropriately
            case .cancelled:
                // perform any work required when the user cancels
            }
        }
    }
}
```

Support

For technical questions, contact support.

Proprietary and Confidential

MUTUAL NONDISCLOSURE AGREEMENT.

Last Update: Document Version 0.9.2 - May 6, 2019