Wallet Extension

Official Apple reference: https://applepaydemo.apple.com/in-app-provisioning-extensions

Wallet Extensions allow users to start card provisioning directly from Apple Wallet, without opening the issuer application first. Apple Wallet displays cards that are available for provisioning from the installed issuer app.

This feature requires two extensions:

- **Non-UI Extension** — provides status, available pass entries, and generates provisioning requests.
- **UI Extension** — handles user authentication when required.

Apple Wallet invokes the extension methods directly. The application should provide only cards that are eligible for provisioning and are not already added to Apple Wallet.

Process Overview

1. User opens Apple Wallet and starts adding a card.
2. Apple Wallet checks the issuer app extension status.
3. The extension reports whether pass entries are available.
4. If required, the UI extension authenticates the user.
5. The non-UI extension provides available cards using `passEntries`.
6. User selects a card.
7. The extension generates `PKAddPaymentPassRequest` using TMP API data.

In `passEntries`, provide data analogous to the in-app provisioning flow described earlier, including card metadata, suffix, cardholder name, and identifier required to start provisioning.

Non-UI Extension Example

```
import PassKit

final class IssuerProvisioningExtensionHandler: PKIssuerProvisioningExtensionHandler {

    override func status(
        completion: @escaping (PKIssuerProvisioningExtensionStatus) -> Void
    ) {
        let status = PKIssuerProvisioningExtensionStatus()
        status.passEntriesAvailable = true
        status.remotePassEntriesAvailable = true
```

```
        status.requiresAuthentication = true

        completion(status)
    }

    override func passEntries(
        completion: @escaping ([PKIssuerProvisioningExtensionPaymentPassEntry]) -> Void
    ) {
        let configuration = PKAddPaymentPassRequestConfiguration(encryptionScheme: .ECC_V2)!
        configuration.cardholderName = "John Doe"
        configuration.primaryAccountSuffix = "1234"

        let entry = PKIssuerProvisioningExtensionPaymentPassEntry(
            identifier: "card-id-123",
            title: "Example Card",
            art: UIImage(named: "card-art")!,
            addRequestConfiguration: configuration
        )

        completion([entry])
    }

    override func remotePassEntries(
        completion: @escaping ([PKIssuerProvisioningExtensionPaymentPassEntry]) -> Void
    ) {
        let configuration = PKAddPaymentPassRequestConfiguration(encryptionScheme: .ECC_V2)!
        configuration.cardholderName = "John Doe"
        configuration.primaryAccountSuffix = "1234"

        let entry = PKIssuerProvisioningExtensionPaymentPassEntry(
            identifier: "card-id-123",
            title: "Example Card",
            art: UIImage(named: "card-art")!,
            addRequestConfiguration: configuration
        )

        completion([entry])
    }

    override func generateAddPaymentPassRequestForPassEntryWithIdentifier(
```

```
        _ identifier: String,
        configuration: PKAddPaymentPassRequestConfiguration,
        certificateChain certificates: [Data],
        nonce: Data,
        nonceSignature: Data,
        completionHandler handler: @escaping (PKAddPaymentPassRequest?) -> Void
    ) {
        Task {
            do {
                let signedCard = try await signCard(
                    cardId: identifier,
                    certificates: certificates,
                    nonce: nonce,
                    nonceSignature: nonceSignature
                )

                let request = PKAddPaymentPassRequest()
                request.activationData = Data(base64Encoded: signedCard.activationData)
                request.ephemeralPublicKey = Data(base64Encoded:
signedCard.ephemeralPublicKey)
                request.encryptedPassData = Data(base64Encoded: signedCard.encryptedData)

                handler(request)
            } catch {
                handler(nil)
            }
        }
    }
}
```

Signing Card with TMP API

Inside `generateAddPaymentPassRequestForPassEntryWithIdentifier`, call:

```
POST /issuer/push-provisioning/signed-cards
```

Use the certificate chain, nonce, and nonce signature provided by Apple Wallet. TMP returns `activationData`, `ephemeralPublicKey`, and `encryptedData`, which are used to create `PKAddPaymentPassRequest`.

```
struct SignCardResponse: Decodable {
    let activationData: String
    let ephemeralPublicKey: String
    let encryptedData: String
}

func signCard(
    cardId: String,
    certificates: [Data],
    nonce: Data,
    nonceSignature: Data
) async throws -> SignCardResponse {
    // Use your existing TMP API integration layer here.
    // The payload is analogous to the in-app provisioning flow.

    fatalError("Example only")
}
```

Sharing Data Between the App and Extensions

Wallet Extensions run in a separate process from the main application. The main app may not be running when Apple Wallet invokes the extension. Because of that, shared data should be stored in a location available to both the app and its extensions.

App Groups

Configure an App Group for:

- the main application target,
- the non-UI Wallet Extension,
- the UI Wallet Extension.

Use the same App Group identifier for all targets, for example:

```
group.com.example.issuerapp
```

Shared non-sensitive data can be stored using:

```
let sharedDefaults = UserDefaults(
    suiteName: "group.com.example.issuerapp"
```

```
)


sharedDefaults?.set(cardIds, forKey: "cardIds")
let cardIds = sharedDefaults?.stringArray(forKey: "cardIds")
```

Shared Keychain

For sensitive data, such as authentication tokens or user session data, configure **Keychain Sharing** for the main app and both extensions.

All targets must use the same Keychain Access Group, for example:

```
$(AppIdentifierPrefix)com.example.issuerapp.shared
```

Use the shared keychain to store data required by the extensions to authenticate requests and communicate with the backend/TMP integration layer.

Notes

- Wallet Extensions require the appropriate Apple entitlement and allow listing.
- The user must open and log in to the issuer app at least once before Apple Wallet can detect the extensions.
- `passEntries` should return only cards that are eligible for provisioning.
- Existing cards should be filtered out using the same PassKit and TMP token status logic described in previous sections.

Revision #1
Created 19 May 2026 07:36:11 by Bartłomiej Jończy
Updated 19 May 2026 07:36:50 by Bartłomiej Jończy