Getting Started with Apple Pay In-App Provisioning

Version 1.23

March 2017

Version Information

Date	Version	Changes
08/31/2015	Draft	Initial
01/06/2016	1.1	Added language regarding Adam ID submission
01/27/2016	1.11	Added FPAN/DPAN suffix, passesOfType, remotePaymentPasses as methods to determine whether to present the Add to Apple Wallet button
07/14/2016	1.2	Updated In-App Provisioning Flow to include cryptography details and added to/formatted the issuer host development step Updated the testing process section for clarity, payment data configuration table, references to PNO to include service provider Added description for PKAddPaymentPassRequestConfiguration, details to cryptography section, Appendix A for the provisioning flow sample, pass metadata to prerequisites, description of test vector usage, and additional details on entitlements
10/03/2016	1.21	Added advice on replacing Add to Apple Wallet Button after a payment pass is provisioned Also added “name” to payment data configuration 3 - encrypted FPAN Added language to clarify Enterprise Team IDs aren’t supported, and clarified numeric nature of Adam ID
02/21/2017	1.22	Added clarification on Crypto OTP formats, sample JSON dictionary, alternative to Add to Apple Wallet button Updated font, formatting
03/02/2017	1.23	Added link to Apple Pay acceptance mark and cleaned up version information

I. Overview

Apple Pay In-App Provisioning provides a credit or debit card issuer the ability to initiate the card provisioning process for Apple Pay directly from the issuer's iOS app.

Cardholders will find the In-App Provisioning feature an extremely convenient method to provision their payment details into their iOS devices by avoiding the need to input card details manually.

Issuers will also find In-App Provisioning an effective component of a seamless mobile banking experience. By driving the provisioning of cards via their iOS mobile apps, issuers can create a unified interface for card provisioning and their other banking services.

Finally, for certain card products issued globally, account details may not be embossed on the actual plastic carried by cardmembers. In-App Provisioning would serve as the sole channel for initiating a card provisioning request for those portfolios as the account details would be provided directly by the issuer to the iOS device.

II. Prerequisites

To implement In-App Provisioning within your iOS app, you must:

1. Support Apple Pay for your card portfolio
2. Develop the app for iOS 9 or later
3. Build the capability on the issuer host system to:

    A. Receive Apple public certificates (ECC) from your iOS app
    B. Generate an ephemeral key pair (ECC)
    C. Utilize the public certificates and ephemeral key pair to generate a shared secret and derive a shared key to encrypt a payment data payload
    D. Transmit encrypted payment data and ephemeral public key back to your iOS app

4. Adhere to the Apple Pay In-App Provisioning security guidelines including support for Multiple Factor Authentication (MFA). For details, please refer to the *Apple Pay In-App Provisioning Security Entitlement Guidelines* document.
5. Ensure your PNO or service provider populates the following keys within the `DPANCardDescriptor` array of the `LinkAndProvisionResponse` API as follows:

A. `associatedApplicationIdentifiers`

- Allows the respective app to see, access and activate your payment passes
- This key needs to match your developer account Team ID and app Bundle ID; look here for more information
- If you use an explicit App ID, it may not match your developer account Team ID

B. `associatedStoreIdentifiers`

- Links to your iOS app or redirects users to download your iOS mobile app from the App Store, in case it is not yet installed on the user's device
- This key needs to match the Adam ID of the iOS mobile app

III. Best Practices

Authenticate the user via One Time Password (OTP) when the user first installs your mobile banking app onto his or her device.
By having the user authenticate when he or she first installs the app, you satisfy Apple Pay MFA requirements and avoid the need to introduce additional authentication steps within the In-App Provisioning flow after the user selects "Add to Apple Wallet." This will save your cardholders time and effort and will result in a more positive user experience.

Use the "Add to Apple Wallet" button in all relevant locations.
Be sure to include the "Add to Apple Wallet" button within your app wherever card or account management features are presented. This will serve to integrate Apple Pay as a central feature of your product offering. See **Section VI** for more details.

Once the pass is provisioned, you can replace the button with text such as "Added to Apple Wallet" or "Available in Apple Wallet."

Display the "Add to Apple Wallet" button for a particular card until it has been added to all associated devices.
You can use the `canAddPaymentPassWithPrimaryAccountIdentifier` method to ensure the button is presented until the pass in question can no longer be provisioned on the device or the Apple Watch, if paired. In the event the issuer iOS app does not have access to the FPAN ID, you can make use of the `passesOfType` and `remotePaymentPasses` methods to obtain the DPAN/FPAN ID and DPAN/FPAN suffix to determine whether to present the "Add to Apple Wallet" button.

Provide key elements within `PKAddPaymentPassRequestConfiguration` which drive the user experience for the In-App Provisioning view controller.
The issuer can provide the following keys to support a positive user experience:

1. CH_NAME - The cardholder name to be displayed is defined within the `cardholderName` key [Figure 1].

2. FPAN_SUFFIX - The funding PAN suffix to be displayed is defined within the `primaryAccountNumberSuffix` key [Figure 1]. This value should be 4 digits and will have dots prepended to indicate that it is a suffix.

3. Localized_Description - The card product description to be displayed is defined within the `localizedDescription` key [Figure 2].

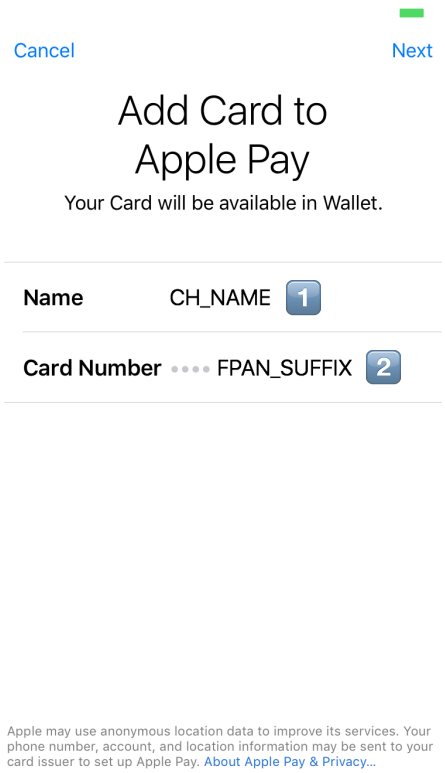


Figure 1

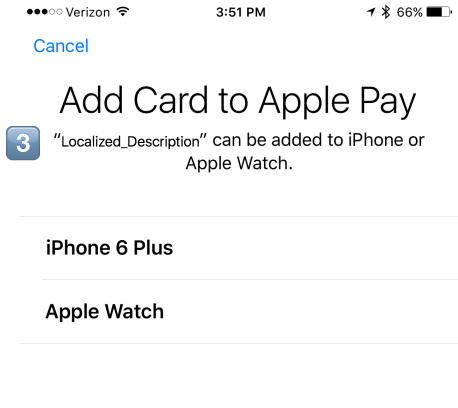


Figure 2

4. Funding PAN Identifier (optional) - If the FPAN ID is passed to Wallet within the `primaryAccountIdentifier` key, Wallet will present only the devices on which the payment pass can still be provisioned [Figure 2]. This screen appears only on an iPhone with a paired Apple Watch.

5. Payment Network (optional) - If a value is provided within `paymentNetwork` key, Wallet will show only the artwork for this specific payment network on the introductory page of the In-App Provisioning Flow. This screen only appears if no card has previously been provisioned within Apple Wallet.

Announce the availability of Apple Pay In-App Provisioning to users.
Encourage new users to get the most from their card through the use of splash screens/ interstitials or clear calls to action.

Link to the Apple Pay setup flow in case you're not able to develop full issuer application based provisioning support within your iOS mobile app and issuer host system.
Developers can leverage the `openPaymentSetup` API within the PassKit programming framework to direct users from the app to the wallet provisioning flow to provision a payment card. For more details, see the Add to Apple Wallet section of this document.

IV. Provisioning Profiles for your App

You will need to submit a request to enable your developer Team ID for the appropriate Apple Pay In-App Provisioning entitlements. Enterprise Team IDs are not supported. Provide your app name, Team ID, and Adam ID via e-mail to apple-pay-provisioning@apple.com.

Once the entitlements have been granted, you'll need to include the distribution entitlement into a provisioning profile and ensure you are leveraging the same profile to develop the app within Xcode. Please follow these steps:

1. Head to the Apple Developer Website and proceed to login
2. Select Certificates, Identifiers & Profiles
3. Select "Distribution" underneath the "Provisioning Profiles" heading on the sidebar
4. On the right, select the distribution iOS provisioning profile that you'll use to deploy your App to the App Store
5. Click "edit" and, from the ensuing entitlements drop down, select "ApplePay In-App Provisioning Distribution" to add the entitlement to the profile. See Figure 3 for details.

Once you've generated a profile which has been assigned the entitlement for In-App Provisioning, within Xcode, head to the *Preferences > Accounts > (Your Account) > View Details* pane where you can then find and download the profile you generated.

Lastly, in build settings, you can then adjust the Provisioning Profile to the newly generated profile. See Figure 4 for details.

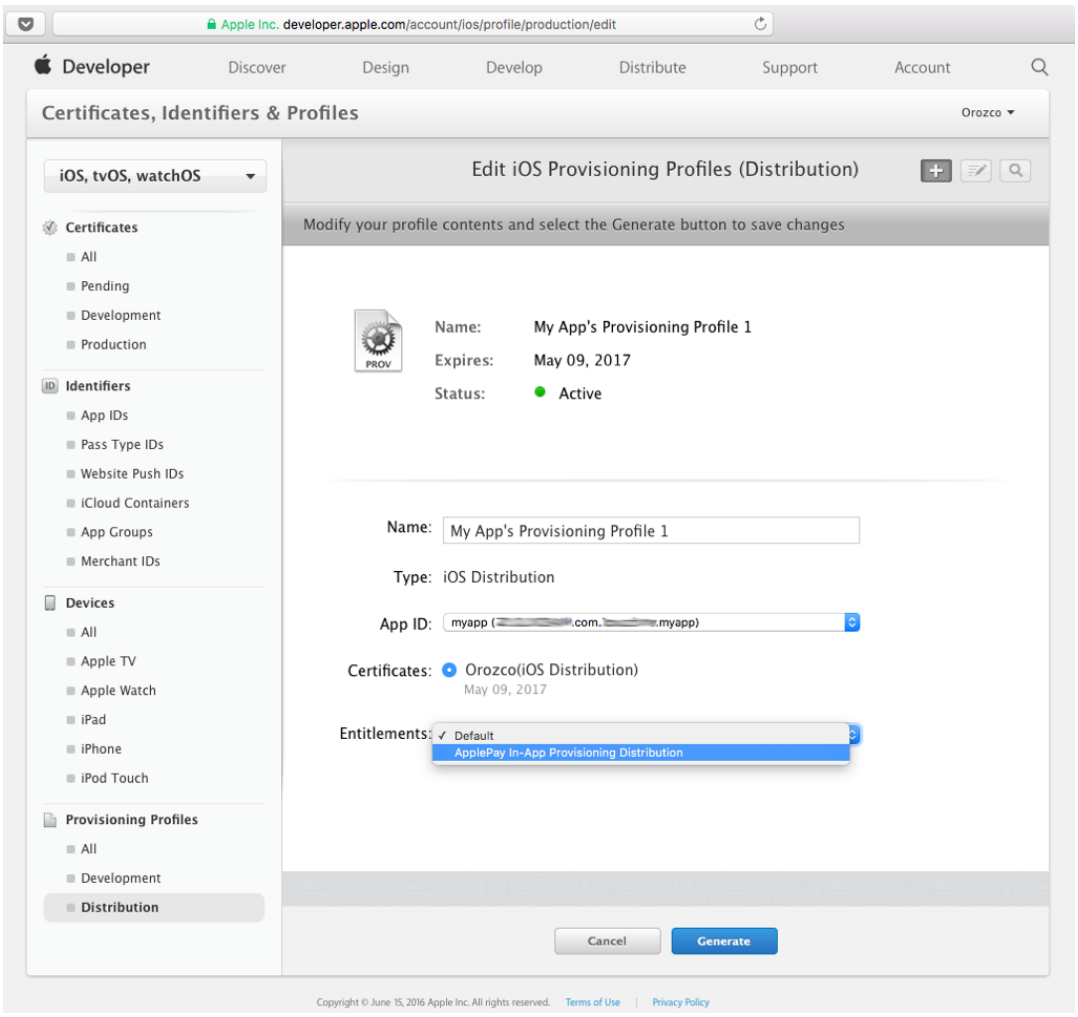


Figure 3

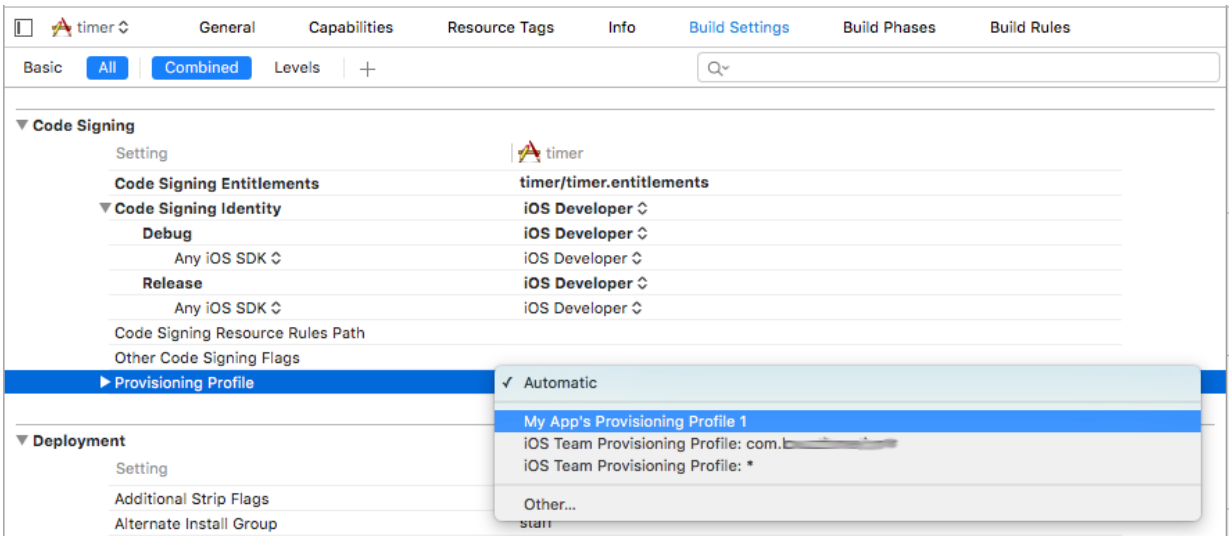


Figure 4

V. In-App Provisioning Flow

Please find below a description and diagram [Figure 5] of the In-App Provisioning process (ECC):

Figure 5

1. User initiates the In-App Provisioning process by selecting the "Add to Apple Wallet" button
2. Apple Wallet requests the public certificates under which the issuer host should encrypt the payment data payload
3. The public certificates and `nonce` are provided to Apple Wallet. Apple Wallet passes the *nonce* to the secure element for signing. `nonceSignature` is returned to Wallet
4. Wallet then passes the public certificates, `nonce`, and the `nonceSignature` to the iOS app
5. The app passes the public certificates, `nonce`, and `nonceSignature` to the issuer host
6. The issuer host will then:

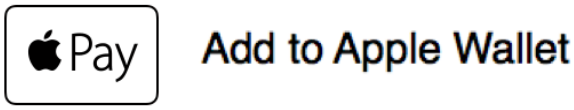
    A. Prepare the payment data payload for the user
    B. Generate an ephemeral key pair,
    C. Encrypt the payload with a shared key derived from the Apple public certificates and generated private ephemeral key
    D. Deliver the encrypted payload and ephemeral public key back to the app. The issuer host will also generate a cryptographic OTP per the Payment Network Operator (PNO) or service provider specifications and pass that to the iOS app as well

7. The app passes the encrypted payment data payload [`encryptedPassData`], the ephemeral public key assigned to the ephemeral private key used to encrypt the payment data payload [`ephemeralPublicKey`], and the cryptographic OTP value [`activationData`] to Wallet through the `PKAddPaymentPassRequest` class
8. Apple Wallet passes the details to the Apple Server where validation checks are performed
9. Payment data is passed to the PNO or service provider and the regular provisioning flow commences

VI. The Add to Apple Wallet Button

To offer In-App provisioning, it is recommended that developers invoke the the `PKAddPassButton` which is available in iOS 9. The use of this button within your app must comply with the Apple guidelines.

Alternatively, you can leverage the Apple Pay logo within a row selector. See directly below for an example. Note that the font for the text "Add to Apple Wallet" can be selected to match your iOS app. See here for the Apple Pay acceptance mark.



In case you are unable to develop the full In-App Provisioning capability, you can also leverage the `OpenPaymentSetup` method as well as the `PKPaymentButton` class with `PKPaymentButtonTypeSetUp` to expose the "Setup up Apple Pay" button which can link to the Wallet provisioning flow.

VII. Provisioning Path Recommendations

Just as with the regular provisioning process, Apple will provide a path recommendation along with any applicable reason codes to issuers. Through In-App Provisioning, an issuer has the option to promote a yellow path recommendation from Apple to the green path with the exception of *Reason Code 0G - Orange Path*.

In case Apple recommends Orange Path for a provisioning request, an issuer can utilize the existing process for Orange Path validation including OTP to a tenured channel with one notable exception: In-App Verification **can not** be presented as an authentication option for provisioning requests initiated within the issuer iOS app.

For more information, please see the *Apple Pay In-App Provisioning Security Entitlement Guidelines*.

VIII. Cryptography

No cryptographic operations to support the transmission of payment data to Apple Wallet should be performed on the iOS device. Rather, the cryptographic functions should be performed by the issuer host.

Via `PKAddPaymentPassViewController`, ECC public certificates will be provided to the iOS app which should then be passed to the issuer host. The issuer host must first verify that the certificate chain is rooted in the Apple Certificate Authority. Subsequently, the issuer host can extract the static public key and generate an ECC ephemeral key pair. The issuer host will then utilize the static public key from Apple and the generated ephemeral private key to derive a shared secret. The shared secret will be inputted into a KDF to calculate the shared key. Details for generating the shared key can be found in Appendix B of the the *Issuer Application Based Provisioning* specification.

The issuer host uses the derived shared key to encrypt the payment details (`encryptedPassData`), and provide it to the iOS app along with the `ephemeralPublicKey` and cryptographic OTP (`activationData`).

IX. Payment Data Configurations

In-App Provisioning supports three configurations for the payment data payload generated by your issuer host (Table 9-1). As the supported configuration(s) vary by PNO or service provider, please reach out to your PNO or service provider relationship manager to confirm which of the following configurations your issuer host should support.

For more information on payment data format, please see the "*`PKAddPaymentPassRequest`*" section of the Issuer Application Based Provisioning specification.

For informational purposes, please see directly below for a sample JSON dictionary which contains all the keys enumerated in the chart above. **Please note** that in the specific payload you will generate, only the keys applicable for the particular configuration you are using should be present.

```
{“nonce":"9c023092",
“nonceSignature":"4082f883ae62d0700c283e225ee9d286713ef74456ba1f07376c
f17d71bf0be013f926d48661939406O ced56030f41f84df916eaab5504e456a8530dc9
c821f6ed3e3af62b5d8f3e4a22ca2018670fee4e",
"name":"Tester Bob”,
“expiration":"12\/15",
“primaryAccountNumber”:"9876543210",
“encryptedPrimaryAccountNumber”:”XXXXXXXXXXXXX”,
“networkName”:”Visa”,
```

```
“productType”:”XXXXXXXXX”,
"primaryAccountNumberPrefix": “483692046”,
“primaryAccountIdentifier”:”XXXXXXXXXXX”}
```

Table 9-1

Configurations		
1	2	3
FPAN	FPAN ID	eFPAN
Name	Nonce	Name*
Expiration Date	Nonce Signature	FPAN Prefix
Nonce	Product Type	Network Name
Nonce Signature		Nonce
Product Type		Nonce Signature
		Product Type

* Name is *required* in payment data configuration 3 unless otherwise specified by Apple, your PNO, or your service provider. Also note, cells in blue are *optional*. Product Type will be provided by Apple, if required.

Note that, within the JSON dictionary prior to encryption, nonce and nonce signature should be included after hex encoding. If you are provisioning Visa cards via VTS, the encrypted FPAN should be Base 64 encoded within the JSON dictionary.

X. Cryptographic OTP

In addition to the encrypted payment data payload, the issuer host must also provide a cryptographic OTP value within the `activationData` property of the `PKAddPaymentPassRequest`. This value is defined by the PNO or service provider to facilitate the verification of the provisioning request.

For more information on the format of the `activationData` field, please see the “`PKAddPaymentPassRequest`” section of the *Issuer Application Based Provisioning* specification. For any questions relating to the generation or validation of cryptographic OTP, please refer to the specifications from your respective PNO or service provider.

As a final note, please be advised that Apple will Base64 encode the cryptographic OTP prior to passing it to the PNO or service provider for validation. Please take this into

consideration during development as well as when testing this feature with the PNO or service provider.

The data used to define the `activationData` object, which is later passed to iOS via the APIs, should appear as follows:

For Visa:

```
MBPAD-1-FK-123456.1--TDEA-7AF291C91F3ED4EF92C1D45EFF127C1F9ABC12347E
```

For MasterCard:

```
{“tokenUniqueReferenceIncluded":"XXXXX",
“signatureAlgorithm":"XXXXXXXX",
“signature":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
“expirationDateIncluded":"XXXX",
"version":"X"}
```

XI. Sample Payment Data

To help you test your issuer host encryption as well as your iOS app’s interface to Apple Wallet, Apple has provided test vectors including a sample set of public certificates, plaintext, ephemeral key pair, shared secret, shared key, and cipher text. To obtain these vectors, please contact apple-pay-provisioning@apple.com.

To use the test vectors:

1. Validate the certificate chain and extract the static public key from the certificates and compare to the static public key listed within the `encryptionlog.txt` file
2. Instead of generating an ephemeral key pair, you can use the ephemeral key pair provided within the `encryptionlog.txt` file
3. Use the static public key and ephemeral private key to derive the shared secret using the ECDH protocol; compare the derived shared secret to the value listed in the `encryptionlog.txt` file
4. Generate the other info as specified in NIST SP 800-56A, Section 5.8.1 as well as Appendix B of the Issuer Application Based Provisioning Specification; compare with the other info value listed within the `encryptionlog.txt` file
5. Input the shared secret and other info into the key derivation function specified in NIST SP 800-56A, Section 5.8.1 to generate the shared key; compare the shared key to the shared key listed within the `encryptionlog.txt` file
6. Encrypt the JSON payload within the cleartext file with the shared key derived in Step 5; compare the resulting cipher text with the cipher text listed within the `encryptionlog.txt` file

Once you have validated your ability to generate each data point, you have successfully validated your cryptography with the test vectors.

XII. Testing Prior to Release on the App Store

It is the responsibility of the issuer to ensure the In-App Provisioning functionality is thoroughly tested in the app across all supported iOS versions and devices prior to releasing to the general public for download.

Testing will occur through the use of the production environments. The iOS app will be distributed for testing purposes via the production App Store after the necessary approvals. A few points to note:

- The issuer must also provide the Adam ID, the numeric Apple ID of the application, to apple-pay-provisioning@apple.com prior to testing.

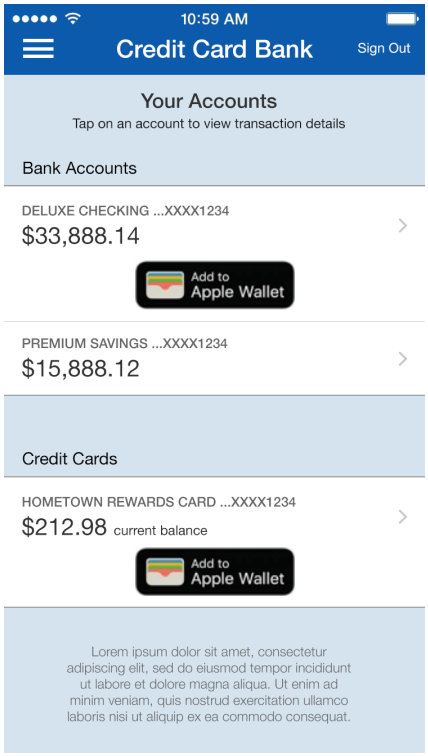
- The distribution of the app for testing purposes must be through the use of Promo Codes. Please look here for more information on the use of Promo Codes for limiting the distribution of an app via the App Store.

- Be sure to select "Manual Release" when submitting your app for App Review, otherwise you may inadvertently release the test app to the general public.

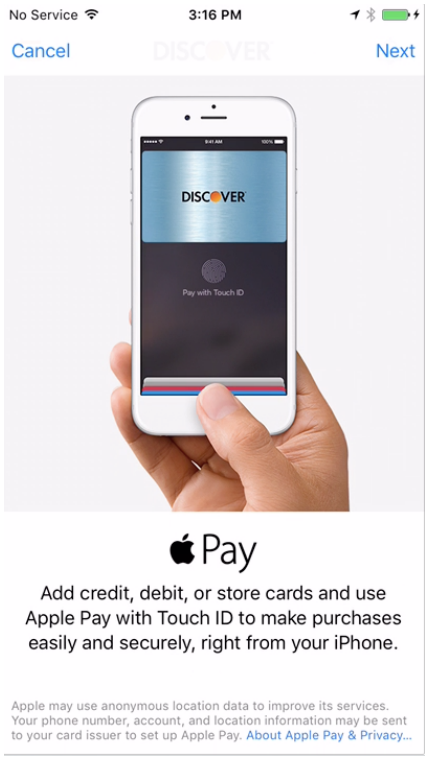
- Once testing is complete, the app can be made available for public download by selecting "Release This Version" within iTunes Connect. In case changes have been made to the app after inclusion on the App Store for testing, you will need to "Cancel This Release" within iTunes Connect. You can then re-submit your corrected app to the App Store for approval. Please click here for additional information on this part of the process.

Please note that Test Flight can not currently be used to distribute apps for In-App Provisioning testing.

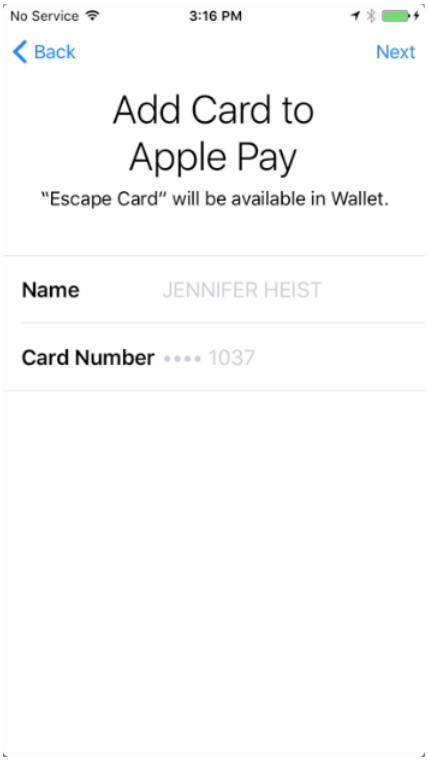
Appendix A - Sample Flows



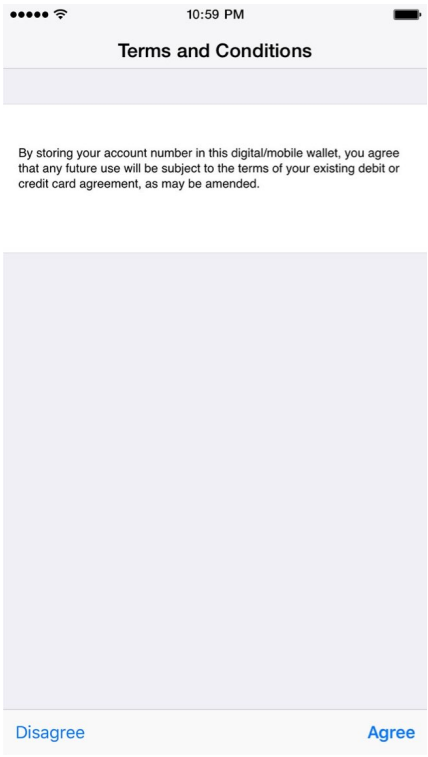
1. Present buttons within app



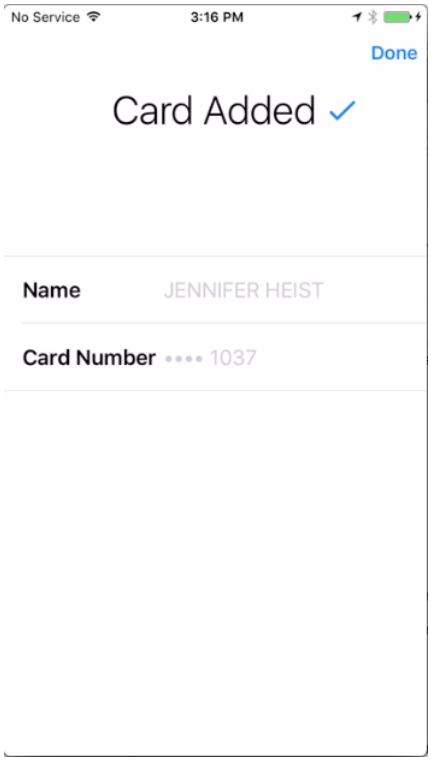
2. Wallet Flow Invoked



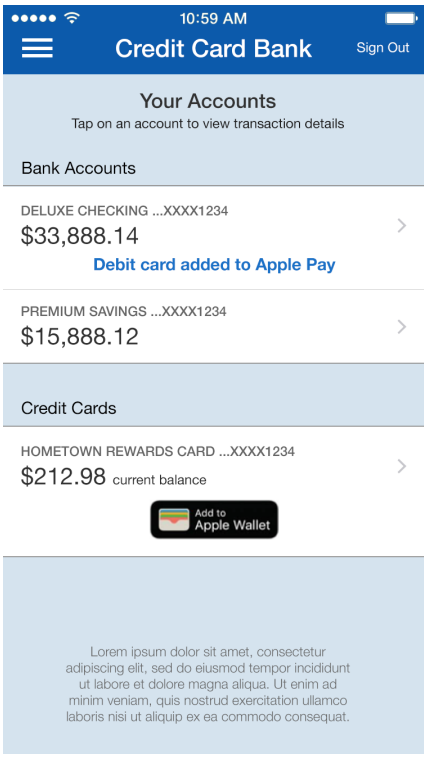
3



4



5



6. Upon completion, return user to the app