HST

MAKING ELECTRONIC TRANSACTIONS **SECURE**

# Push Provisioning
for Apple Pay

Version 1.9 (23.12)

Summary

1. Introduction

This document provides technical information about the card push provisioning functionality in iOS applications to enable Issuer application to perform card provisioning to ApplePay using HST's SDK which will abstract the complexity of technical & security requirements required by Apple Pay, Visa and Mastercard.

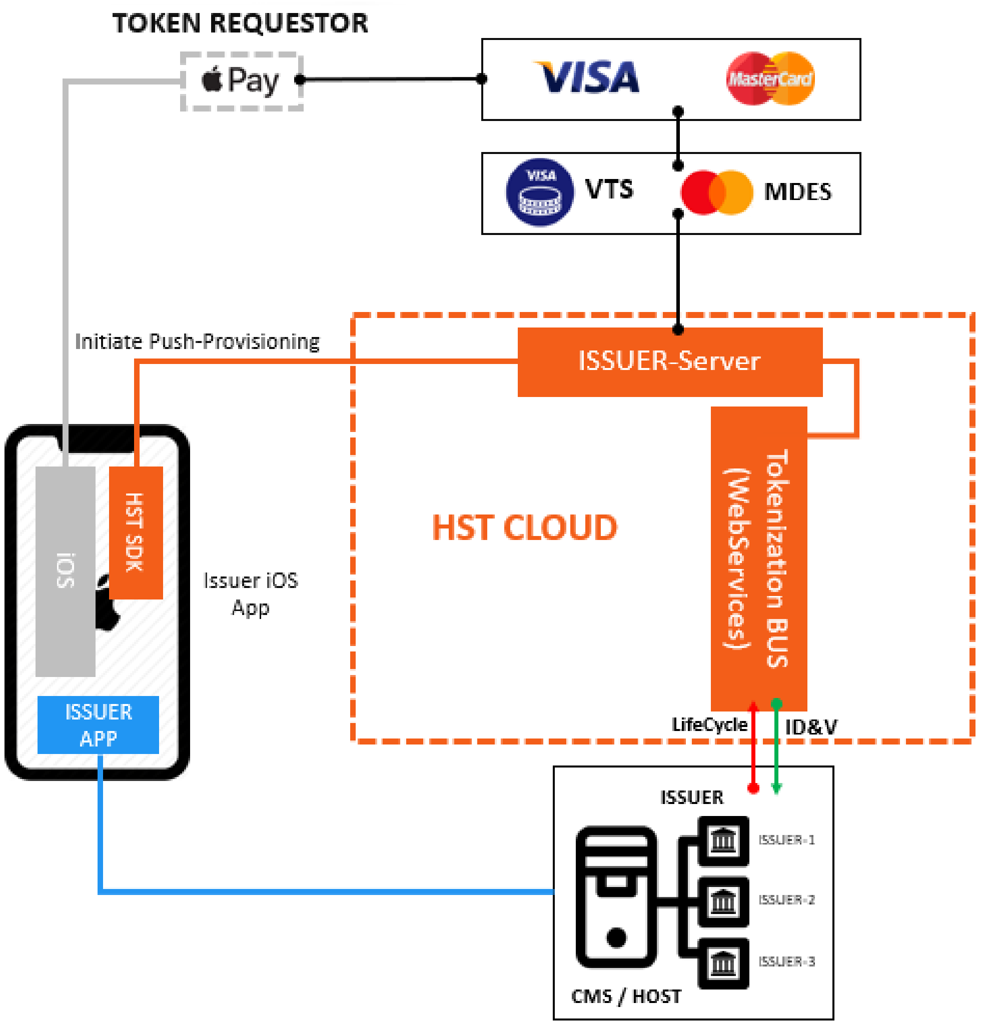
It also describes the steps of the entire card digitalization flow starting from Issuer iOS Application to the Apple Pay application, the HST Issuer Server APIs and HST Push Provisioning SDK methods and requirements for implementation, also providing illustrative examples of the flow.

Besides the In App Provisioning, there are other requirements the issuer must follows to comply with all Apple Pay requirements. The Issuer must read the following documentation to be aware of all these requirements:

- [AMR] Card Issuer Functional Requirements v2.5
- Getting Started with Apple Pay In-App Provisioning, Verification & Security (3.0)
- Apple Pay E2E Certification Guide V2.0

2. General Architecture

The following diagram presents the general architecture of the platform, its components, and sites of execution:



3. In App Provisioning

This implementation allows the cardholder to start card provisioning to ApplePay from the Issuer iOS application by just clicking on a button called "Add to Apple Pay", avoiding manual insertion of card information by the cardholder.

3.1. Push Provisioning Flow

The Push Provisioning process starts on the bank App installed on the phone and can be divided in 3 steps: Card Selection, Data Preparation and Provisioning. During this process, the bank App interacts with HST SDK (H2P) and optionally with HST's backend APIs to execute these steps.

3.1.1. Card Selection Step

On Card Selection step the banking App must determine if the device is capable of card digitization and, for each card belonging to the cardholder, if it is going to present the button "Add to Apple Wallet" or the information "Added to Apple Wallet" (alternately "Available in Apple Wallet").

The device capability can be checked using the SDK method isAvailable().

To determine which cards have already been digitized on the device, the application should use the method getCards(). This method lists all cards digitized on the Apple Wallet and attached devices (Apple Watch) which the Issuer App has access (please refer to section 5.4. PNO Pass Metadata Configuration in documentation Getting Started with Apple Pay In-App Provisioning Verification & Security 3.0).

Note that if a card has been digitized on both devices (phone and Apple Watch), it will be presented in two different entries of the list returned by the method. The option "Add to Apple Wallet" can be changed to "Add to Apple Watch" if the card has been digitized on the phone but not on the watch.

3.1.2. Data Preparation Step

After the cardholder selects the card to be digitized and clicks "Add to Apple Wallet", the Issuer App must prepare card information for the digitization process. Considering that card information is confidential, the issuer has the following options:

Option C: The Issuer encrypts card information and passes it directly to the SDK on the phone on the provisioning step.

Option C is more suitable for Issuers that want to minimize the interaction with HST's backend. Regardless of the option used, the Issuer shall not manipulate or encrypt card information on the banking App. All the cryptography should be done on the issuer backend to avoid security issues.

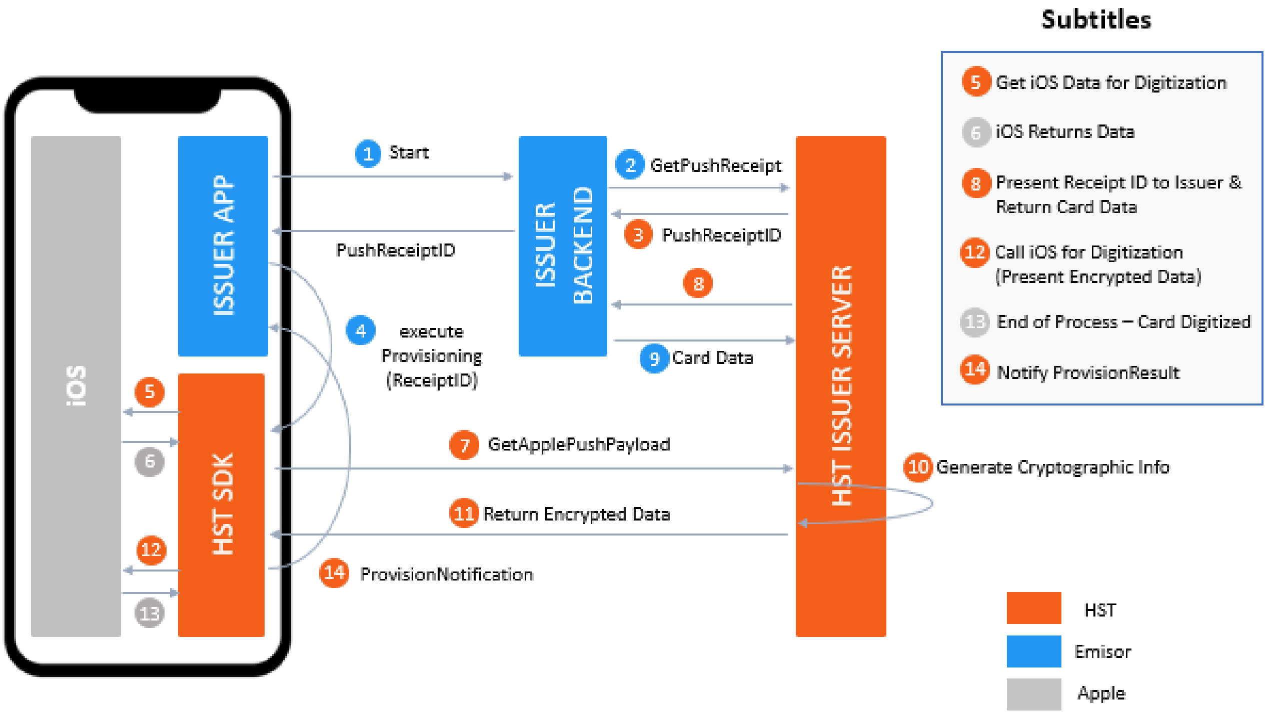
Option A (deprecated): *~~The Issuer obtains a receipt ID of the process on HST's backend, that will be used to inquiry the issuer about the card information on the provisioning step.~~*

Option B (deprecated): *~~The Issuer informs HST's backend of the card data and obtains a receipt ID that will be used later to link the card information to the digitization process.~~*

~~Options A and B are more suitable to Issuers already connected to HST's backend. They have the advantage of not dealing with card information on the device. On the other hand, the Issuer has to implement APIs on the backend.~~

3.1.2.1. Option A (deprecated)

~~The following diagram presents the sequence of calls for Option A.~~


Subtitles
5 Get iOS Data for Digitization
6 iOS Returns Data
8 Present Receipt ID to Issuer & Return Card Data
12 Call iOS for Digitization (Present Encrypted Data)
13 End of Process – Card Digitized
14 Notify ProvisionResult
iOS
ISSUER APP
HST SDK
ISSUER BACKEND
HST ISSUER SERVER
1 Start
PushReceiptID
2 GetPushReceipt
3 PushReceiptID
8
9 Card Data
4 execute Provisioning (ReceiptID)
5
6
7 GetApplePushPayload
10 Generate Cryptographic Info
11 Return Encrypted Data
12
13
14 ProvisionNotification
HST
Emisor
Apple

~~The process starts after the customer clicks on "Add to Apple Wallet" button associated to a specific card. The Issuer Backend calls the GetPushReceipt API on HST's backend (refer to inbound APIs on chapter 5). The API will return a pushReceiptID for the process which the issuer must link to the card to be pushed. The pushReceiptID and the card data must be temporarily stored on the backend for further processing. Note that the encryptedCardInfo field should not be used on this call.~~

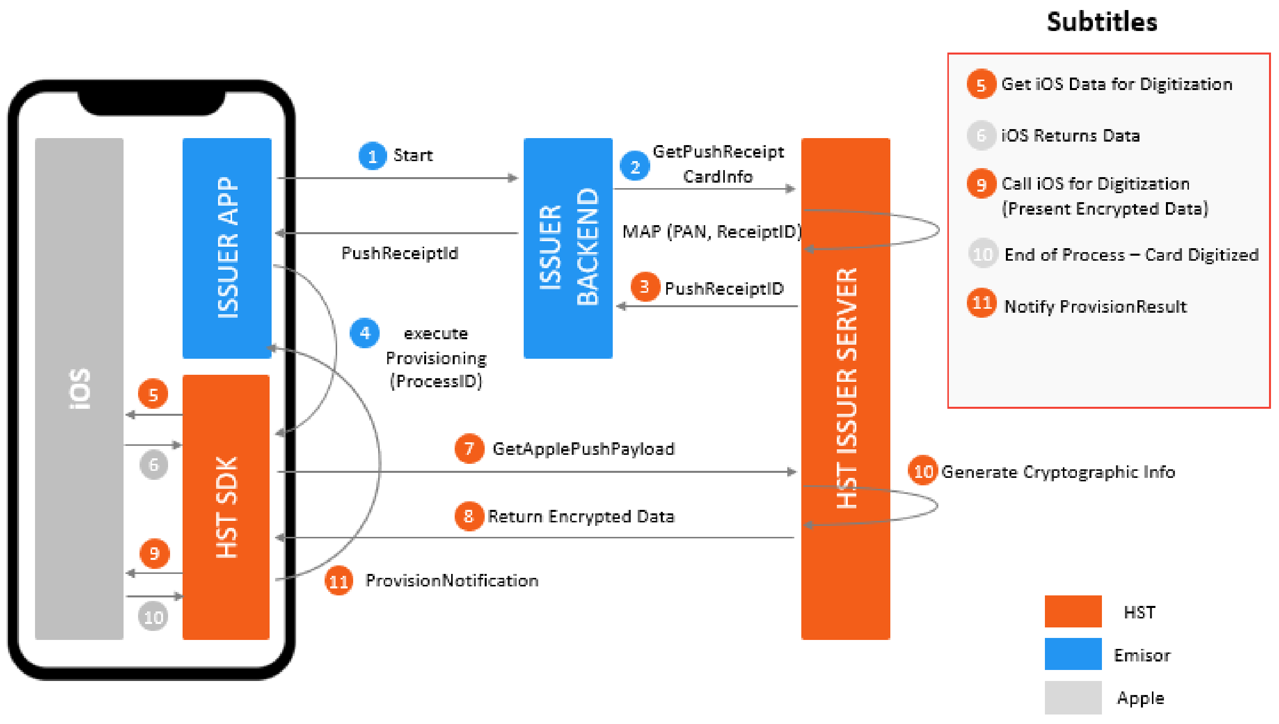
~~On the provisioning phase of the flow (step 7 of the diagram) a call to the SDK will send information to HST backend, that will trigger a call to issuer backend (step 8 of the diagram). Issuer must implement the API GetPushCard (refer to outbound APIs on chapter 5.6.1). HST backend sends the~~

~~*pushReceiptID of the process and the Issuer backend must return the card data associated to the pushReceiptID.*~~

~~***IMPORTANT:** In this use case, the sensitive data referred on the 9th step is not held in HST database. The card data is only used during the transaction and purged immediately from de system's volatile memory.*~~

3.1.2.2. Option B (deprecated)

~~*The following diagram presents the sequence of calls for Option B.*~~


Subtitles
5 Get iOS Data for Digitization
6 iOS Returns Data
9 Call iOS for Digitization (Present Encrypted Data)
10 End of Process – Card Digitized
11 Notify ProvisionResult
iOS
ISSUER APP
HST SDK
ISSUER BACKEND
HST ISSUER SERVER
1 Start
PushReceiptId
2 GetPushReceipt CardInfo
MAP (PAN, ReceiptID)
3 PushReceiptID
4 execute Provisioning (ProcessID)
5
6
7 GetApplePushPayload
8 Return Encrypted Data
9
10
10 Generate Cryptographic Info
11 ProvisionNotification
HST
Emisor
Apple

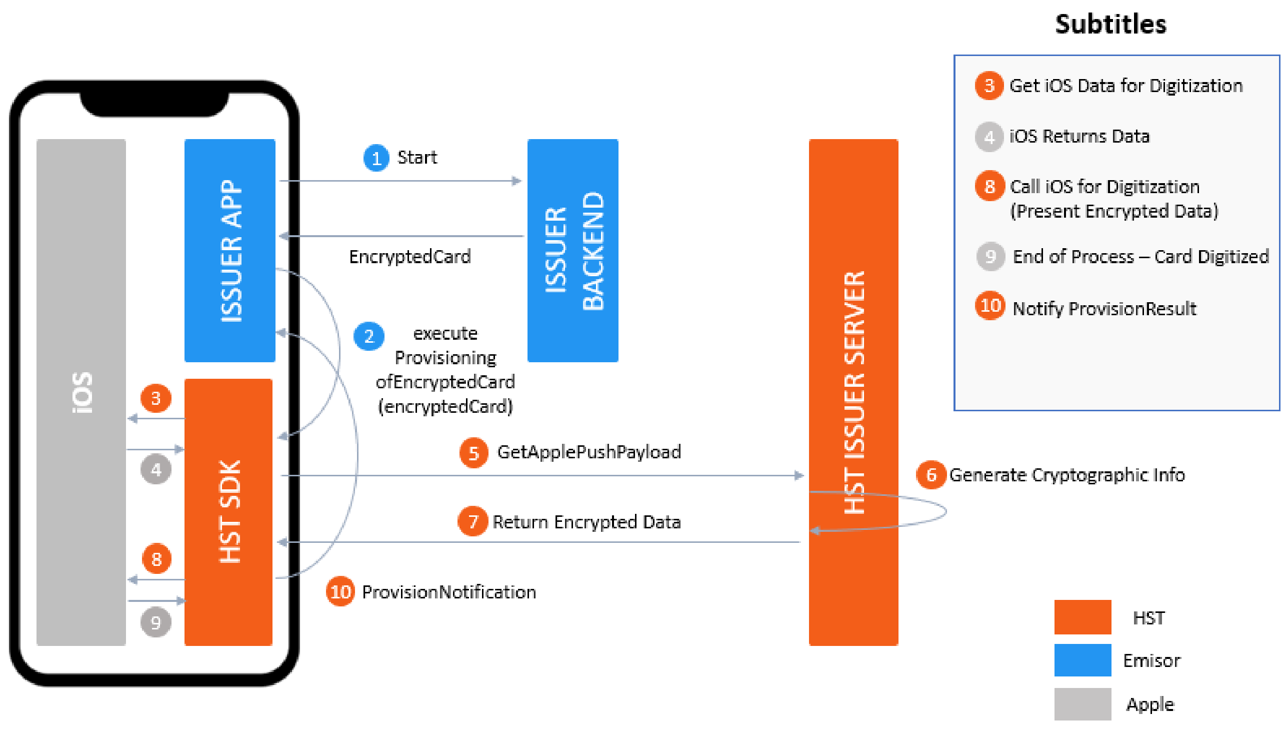
~~*The process starts after the customer clicks on "Add to Apple Wallet" button associated to a specific card. The Issuer Backend calls the GetPushReceipt API on HST's backend (refer to inbound APIs on chapter 5). In this case the Issuer must send the encryptedCardInfo field. HST's backend will temporarily store card data and will return a pushReceiptID for the process, that will be associated to the card data received.*~~

~~*On the provisioning phase of the flow (step 7 of the diagram) a call to the SDK will send information to HST backend, that will match the pushReceiptID to the card data temporarily stored. HST will not make any extra call to the Issuer backend.*~~

~~***Note**: In this use case, the sensitive data referred on the third step is held in HST database for 15 minutes before automatic exclusion.*~~

3.1.2.3. Option C

The following diagram presents the sequence of calls for Option C.


Subtitles
3 Get iOS Data for Digitization
4 iOS Returns Data
8 Call iOS for Digitization (Present Encrypted Data)
9 End of Process – Card Digitized
10 Notify ProvisionResult
iOS
ISSUER APP
HST SDK
ISSUER BACKEND
HST ISSUER SERVER
1 Start
EncryptedCard
2 execute Provisioning ofEncryptedCard (encryptedCard)
3
4
5 GetApplePushPayload
6 Generate Cryptographic Info
7 Return Encrypted Data
8
9
10 ProvisionNotification
HST
Emisor
Apple

For option C there is no communication between Issuer and HST's backend. Issuer should encrypt card data using JWE standard according to the *executeProvisioningOfEncryptedCard*() method of the SDK. More information about this cryptographic process is explained at section 7.

3.1.3. Provisioning Step

After the cardholder selects the card to be digitized and the Issuer system prepares the data to be exchanged, the Issuer App must call the executeProvisioning() or executeProvisioningOfEncryptedCard() methods of the SDK. When working with options A or B, the executeProvisioning() method has to be used, and the Issuer App will provide the pushReceiptID as a parameter. During this call the SDK will communicate with HST's backend and will retrieve the card information necessary to prepare the push provisioning. If option C is used, the App should call executeProvisioningOfEncryptedCard() and provide the encrypted card info as parameter. In either case, HST's backend will use the card info to prepare the data to be pushed to Apple Wallet.

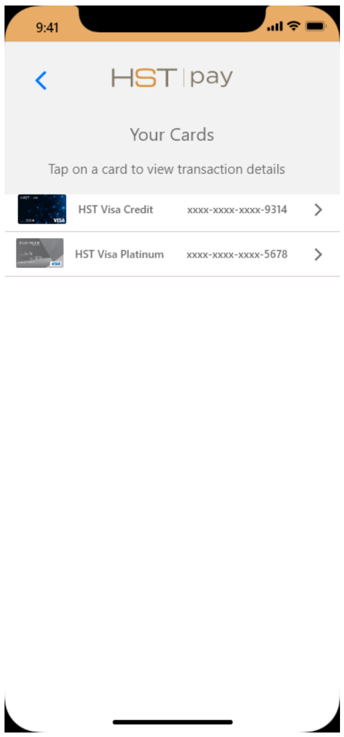
After the push information is prepared on HST's Backend, it is returned to the SDK. The SDK continues the process with Apple Wallet. The information of "*cardholderName*", "*panLastFour*" and "*cardDescr*", provided as parameters to the *executeProvisioning* methods, are used by Apple Wallet on the confirmation screen. The parameter *panID* is used by Apple Wallet for the device selection screen. If the card has never been digitized on the device, this parameter should be nil and in case there is an Apple Watch connected to the phone, Apple Wallet will present a screen so the customer can choose to which device the card is going to be tokenized. If the card has already been tokenized in the phone or watch, the *getCards*() method returns it in the list of digitized cards. The related *cardID* should be presented on the second provisioning so Apple Wallet will not present this device as an option to the cardholder. This parameter should be used to enhance the user experience.

The *executeProvisiong*() and *executeProvisioningOfEncryptedCard*() methods are asynchronous. The App should implement the class CommEvents and provide an instance of this class as parameter to these methods (parameter events). Before the SDK starts processing the provisioning, it will execute the callback function *onPreExecute*(). Considering that the SDK will communicate with HST's backend during this process, and this communication can take some time, it is highly recommended that the App implements this method and presents a message to the customer informing that the provisioning is been processed.

At the end of the provisioning, the SDK will execute the callback function *onPostExecute*(). The result of the provisioning will be returned on object *commEventsResult*.

3.2. Illustrative Mobile Application Screen Flow

The following screens illustrate the process of card push provisioning from the Issuer iOS Application to Apple Pay:



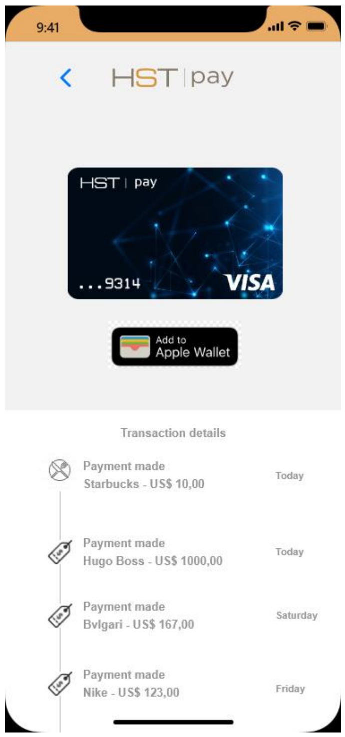
9:41

HST | pay

Your Cards

Tap on a card to view transaction details

HST Visa Credit	xxxx-xxxx-xxxx-9314	>
HST Visa Platinum	xxxx-xxxx-xxxx-5678	>

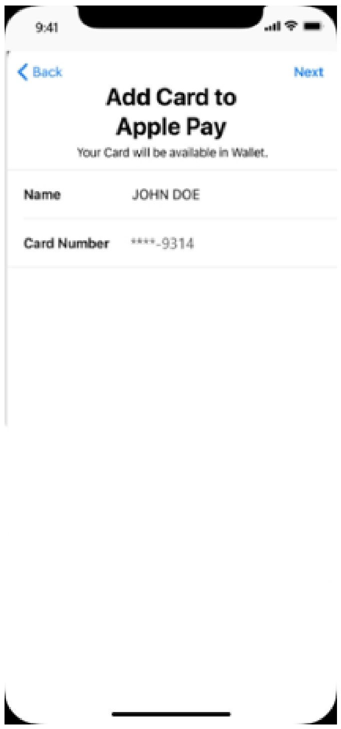


9:41

HST | pay

Transaction details

Payment made Starbucks - US$ 10,00	Today
Payment made Hugo Boss - US$ 1000,00	Today
Payment made Bvlgari - US$ 167,00	Saturday
Payment made Nike - US$ 123,00	Friday



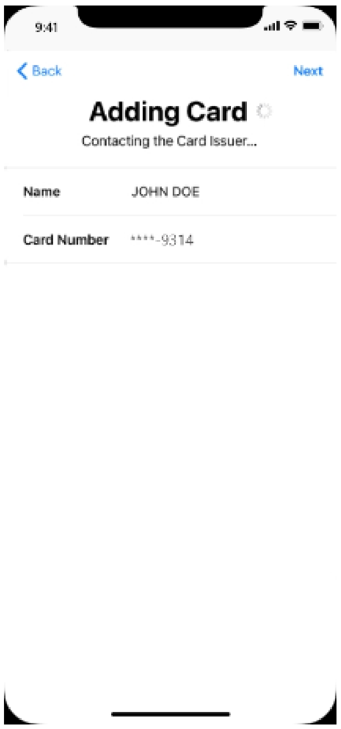
9:41

< Back Next

Add Card to Apple Pay

Your Card will be available in Wallet.

Name	JOHN DOE
Card Number	****-9314



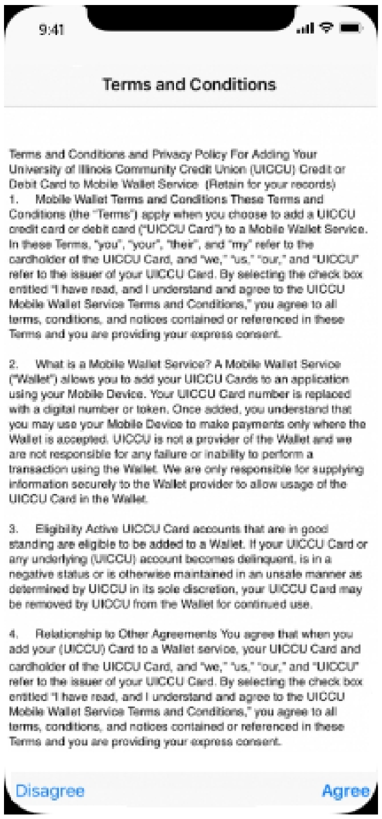
9:41

< Back Next

Adding Card

Contacting the Card Issuer...

Name	JOHN DOE
Card Number	****-9314



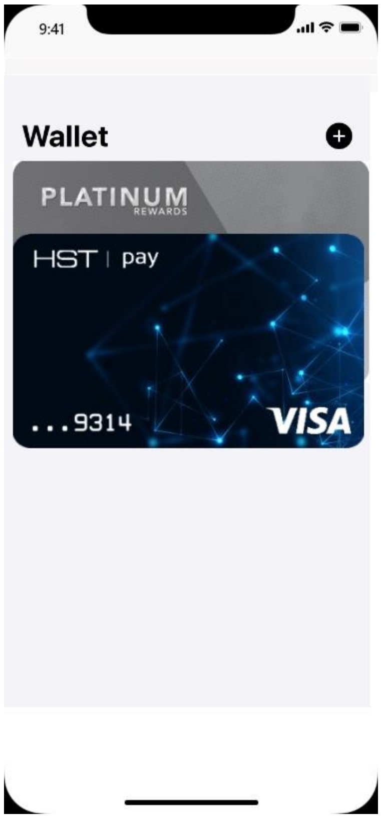
Terms and Conditions and Privacy Policy For Adding Your University of Illinois Community Credit Union (UICCU) Credit or Debit Card to Mobile Wallet Service (Retain for your records)

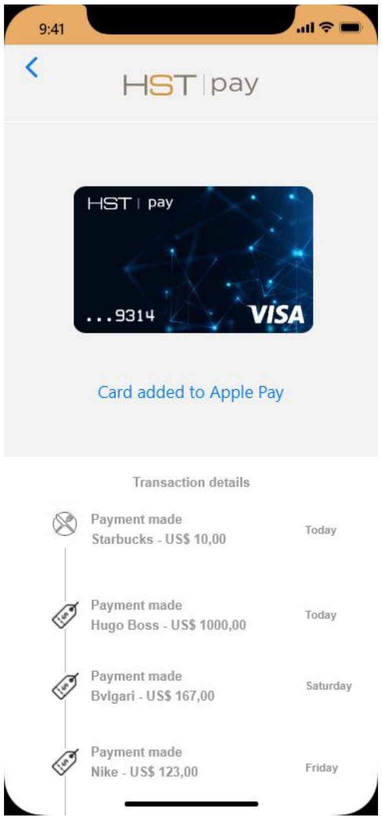
1. Mobile Wallet Terms and Conditions These Terms and Conditions (the "Terms") apply when you choose to add a UICCU credit card or debit card ("UICCU Card") to a Mobile Wallet Service. In these Terms, "you", "your", "their", and "my" refer to the cardholder of the UICCU Card, and "we," "us," "our," and "UICCU" refer to the issuer of your UICCU Card. By selecting the check box entitled "I have read, and I understand and agree to the UICCU Mobile Wallet Service Terms and Conditions," you agree to all terms, conditions, and notices contained or referenced in these Terms and you are providing your express consent.

2. What is a Mobile Wallet Service? A Mobile Wallet Service ("Wallet") allows you to add your UICCU Cards to an application using your Mobile Device. Your UICCU Card number is replaced with a digital number or token. Once added, you understand that you may use your Mobile Device to make payments only where the Wallet is accepted. UICCU is not a provider of the Wallet and we are not responsible for any failure or inability to perform a transaction using the Wallet. We are only responsible for supplying information securely to the Wallet provider to allow usage of the UICCU Card in the Wallet.

3. Eligibility Active UICCU Card accounts that are in good standing are eligible to be added to a Wallet. If your UICCU Card or any underlying (UICCU) account becomes delinquent, is in a negative status or is otherwise maintained in an unsafe manner as determined by UICCU in its sole discretion, your UICCU Card may be removed by UICCU from the Wallet for continued use.

4. Relationship to Other Agreements You agree that when you add your (UICCU) Card to a Wallet service, your UICCU Card and cardholder of the UICCU Card, and "we," "us," "our," and "UICCU" refer to the issuer of your UICCU Card. By selecting the check box entitled "I have read, and I understand and agree to the UICCU Mobile Wallet Service Terms and Conditions," you agree to all terms, conditions, and notices contained or referenced in these Terms and you are providing your express consent.





Transaction details

Payment made	Starbucks - US$ 10,00	Today
Payment made	Hugo Boss - US$ 1000,00	Today
Payment made	Bvlgari - US$ 167,00	Saturday
Payment made	Nike - US$ 123,00	Friday

4. HP2 SDK

The HST Push Provisioning (HP2) SDK is the library that has to be incorporated to the bank's App in order to execute the push provisioning.

4.1. Use cases and sequence flows

4.1.1. Initialization

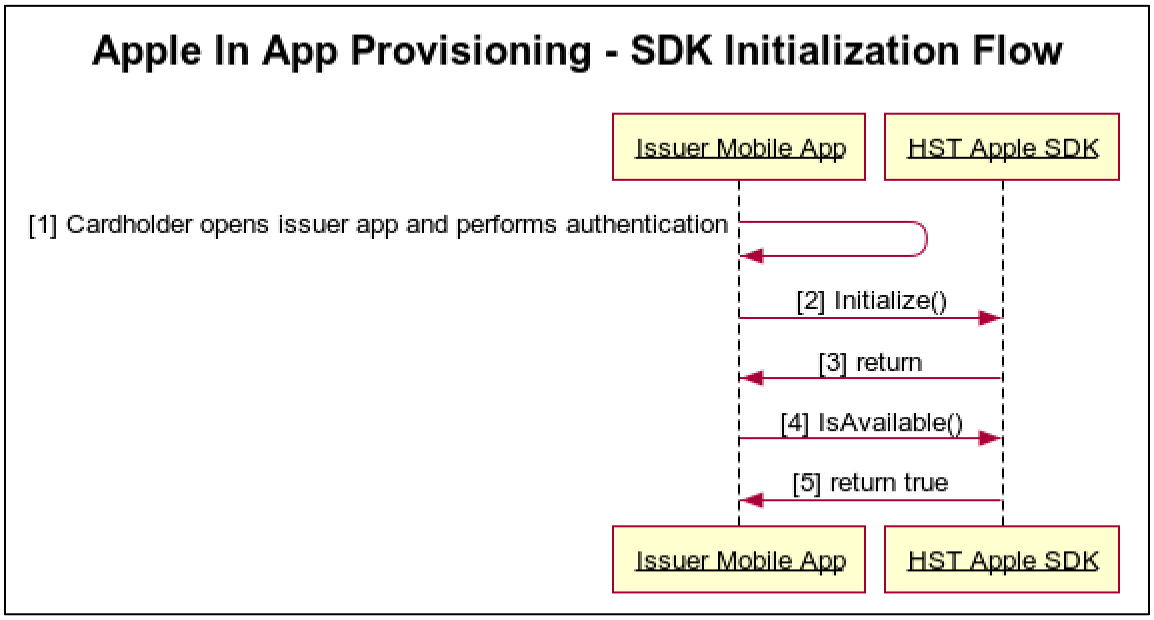


Figure 1- Initialization Flow

4.1.2. Adding a new card in Apple Pay

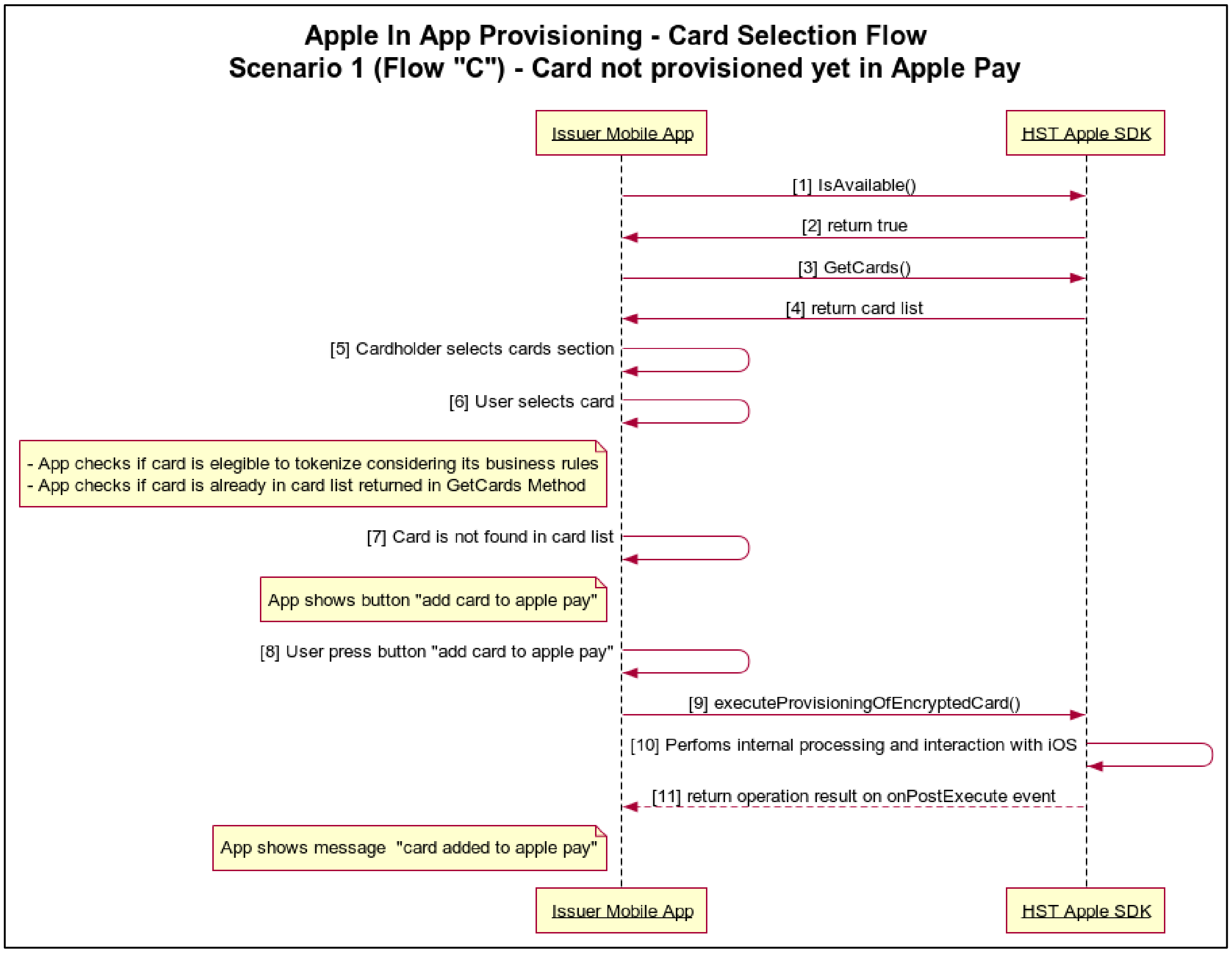

Apple In App Provisioning - Card Selection Flow
Scenario 1 (Flow "C") - Card not provisioned yet in Apple Pay
Issuer Mobile App
HST Apple SDK
[1] IsAvailable()
[2] return true
[3] GetCards()
[4] return card list
[5] Cardholder selects cards section
[6] User selects card
- App checks if card is elegible to tokenize considering its business rules
- App checks if card is already in card list returned in GetCards Method
[7] Card is not found in card list
App shows button "add card to apple pay"
[8] User press button "add card to apple pay"
[9] executeProvisioningOfEncryptedCard()
[10] Perfoms internal processing and interaction with iOS
[11] return operation result on onPostExecute event
App shows message "card added to apple pay"
Issuer Mobile App
HST Apple SDK

Figure 2 - Card selection flow - Card not provisioned in Apple Pay yet

4.1.3. Adding a card in another device

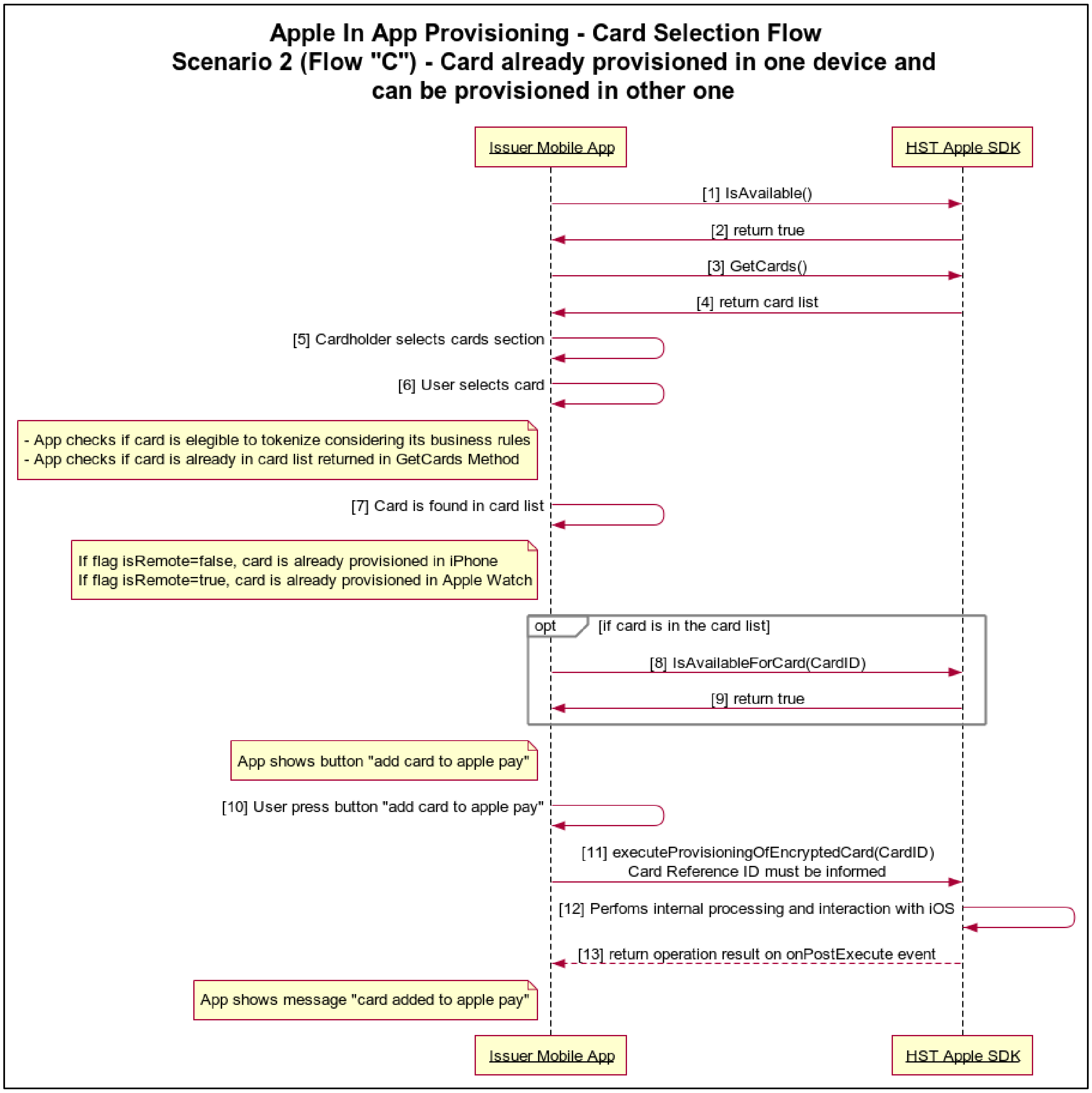

Apple In App Provisioning - Card Selection Flow
Scenario 2 (Flow "C") - Card already provisioned in one device and can be provisioned in other one
Issuer Mobile App
HST Apple SDK
[1] IsAvailable()
[2] return true
[3] GetCards()
[4] return card list
[5] Cardholder selects cards section
[6] User selects card
- App checks if card is elegible to tokenize considering its business rules
- App checks if card is already in card list returned in GetCards Method
[7] Card is found in card list
If flag isRemote=false, card is already provisioned in iPhone
If flag isRemote=true, card is already provisioned in Apple Watch
opt
[if card is in the card list]
[8] IsAvailableForCard(CardID)
[9] return true
App shows button "add card to apple pay"
[10] User press button "add card to apple pay"
[11] executeProvisioningOfEncryptedCard(CardID)
Card Reference ID must be informed
[12] Perfoms internal processing and interaction with iOS
[13] return operation result on onPostExecute event
App shows message "card added to apple pay"
Issuer Mobile App
HST Apple SDK

Figure 3 - Card already provisioned in one device and can be provisioned in another one

4.1.4. Card already provisioned in all available devices

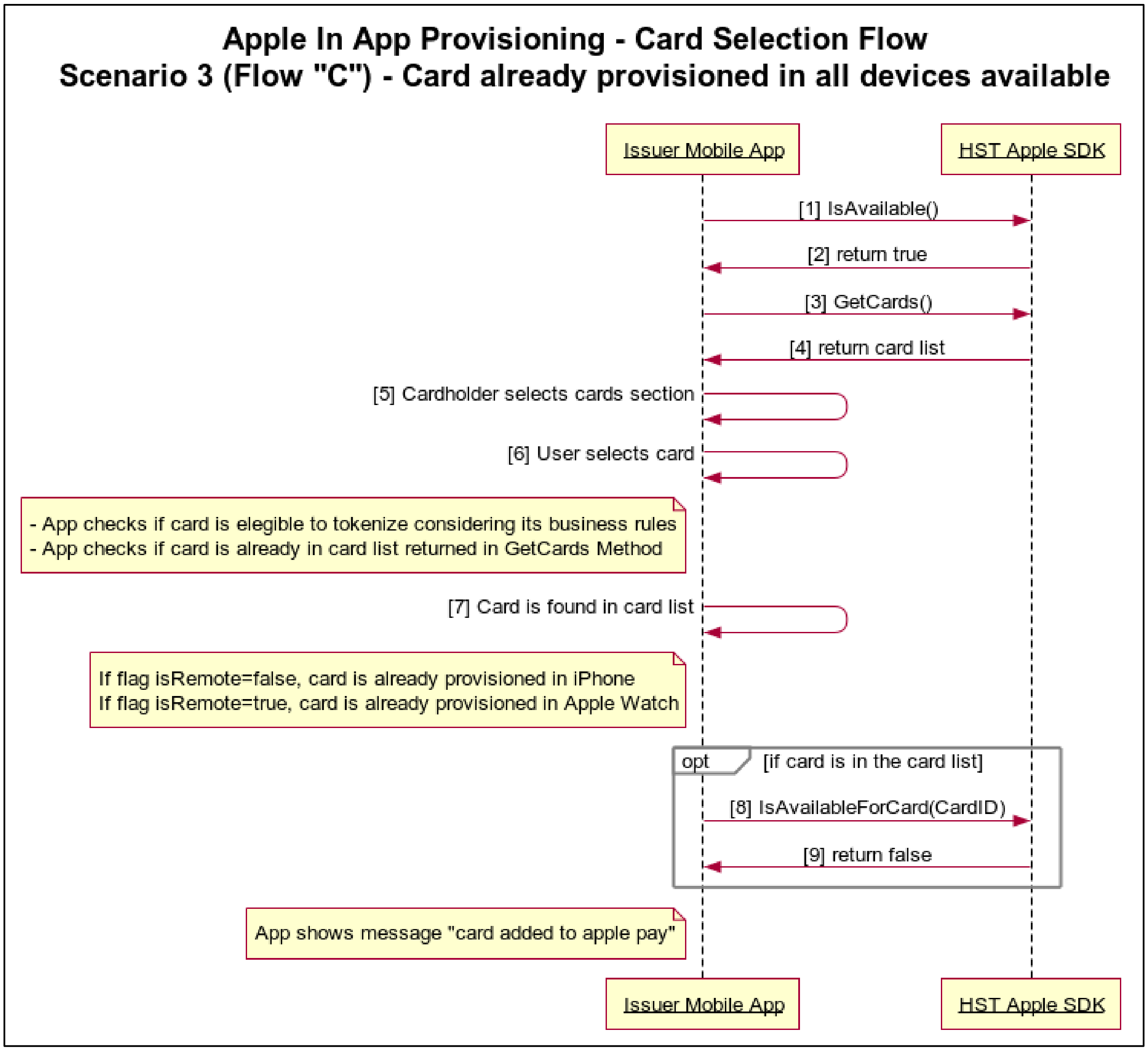

Apple In App Provisioning - Card Selection Flow
Scenario 3 (Flow "C") - Card already provisioned in all devices available
Issuer Mobile App
HST Apple SDK
[1] IsAvailable()
[2] return true
[3] GetCards()
[4] return card list
[5] Cardholder selects cards section
[6] User selects card
- App checks if card is elegible to tokenize considering its business rules
- App checks if card is already in card list returned in GetCards Method
[7] Card is found in card list
If flag isRemote=false, card is already provisioned in iPhone
If flag isRemote=true, card is already provisioned in Apple Watch
opt
[if card is in the card list]
[8] IsAvailableForCard(CardID)
[9] return false
App shows message "card added to apple pay"
Issuer Mobile App
HST Apple SDK

Figure 4 - Card already provisioned in all devices available

4.2. HP2 SDK Methods

This section describes the HP2 API methods that can be called by the Issuer Mobile application. All these methods are defined in **HP2** framework.

4.2.1. Initialize

Description:	This class constructor is responsible for initializing the SDK. It must be called before any other method. Due to security reasons, it is extremely recommended to call this class constructor during application startup.
Prototype:	**public** init (institutionCode: String)
Input Param:	*String institutionCode* ↳The financial institution code provided during onboarding on HST Tokenization Cloud. Contact your technical support to get it.
Return:	None.
Requirements:	None.
Comments:	None.
Exception:	None.

Table 1 – Initialize API

4.2.2. Get Cards

Description:	Returns the cards digitalized on the phone or connected devices.
Prototype:	**public func** getCards ()-> [Card]?
Input Param:	None.
Return:	The method returns a list of available cards. Check Card object to get more details.
Requirements:	None.
Comments:	Only cards entitled to the specific App are returned.
Exception:	None.

Table 2 – Get Cards API

4.2.3. Is Available

Description:	Checks if the device has the capability to digitize cards.
Prototype:	**public func** isAvailable ()-> Bool
Input Param:	None.
Return:	True if the device is available for card digitization, False otherwise.
Requirements:	None.
Comments:	None.
Exception:	None.

Table 3 – Is Available API

4.2.4. Is Available for Card

Description:	Checks if the specific card is available or not for digitization.
Prototype:	`public func isAvailableForCard (panRefId: String) -> Bool`
Input Param:	*String panRefId* ↳ The PAN Reference ID returned on the first digitization of the card on the device.
Return:	True if the card is available for digitization, False otherwise.
Requirements:	None.
Comments:	None.
Exception:	None.

Table 4 – Is Available for Card API

4.2.5. Execute Provisioning

Description:	Starts the provisioning process of the card associated to the pushReceiptIdentifier.
Prototype:	`public func executeProvisioning (parentViewController: ViewController, institutionCode: String, cardholderName: String, panLastFour: String, cardDescr: String, panId: String, pnp: String, pushRecId: String, events: CommEvents?) throws`
Input Params:	*ViewController parentViewController* ↳ An instance of a controller to handle the process of card provisioning in a secure way. *String institutionCode* ↳The financial institution code provided during onboarding on HST Tokenization Cloud. Contact your technical support to get it. *String cardholderName* ↳ Cardholder name, to be presented on the confirmation screen (see screen example on comments). *String panLastFour* ↳ Last 4 Digits of the PAN to be presented on the confirmation screen. *String cardDescr* ↳ Card description to be presented on the confirmation screen. *String panID* ↳ PAN Reference ID. Can be obtained from object Card returned on function GetCards(). Nil if the panID doesn't exist. *String pnp* ↳ Payment Network Provider ("visa", "mastercard" or "amex"). *String pushRecId* ↳ Push receipt Identifier. *CommEvents events*
Return:	
Requirements:	

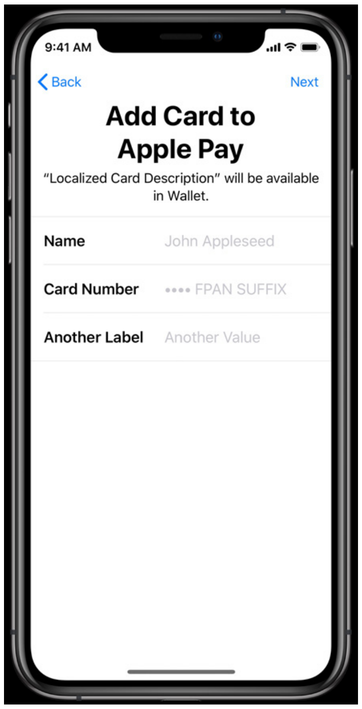
Comments:	↳ Instance of the class CommEvents used to callback the App. The operation result will be returned on onPostExecute method(). None. None. While processing the request, iOS will present the following screen for customer confirmation.  9:41 AM Back Next **Add Card to Apple Pay** "Localized Card Description" will be available in Wallet. **Name** John Appleseed **Card Number** •••• FPAN SUFFIX **Another Label** Another Value
Exception:	None.

Table 5 – Execute Provisioning API

4.2.6. Execute Provisioning of Encrypted Card

Description: Starts the provisioning process of the encrypted card.

Prototype:

```
public func executeProvisioningOfEncryptedCard (parentViewController: ViewController, institutionCode: String, cardholderName: String, panSufix: String, cardDescr: String, panId: String, pnp: String, encCard: String, events: CommEvents?) throws
```

Input Params:

ViewController parentViewController

↳ The instance of a controller to handle the process of card provisioning in a secure way.

String institutionCode

↳The financial institution code provided during onboarding on HST Tokenization Cloud. Contact your technical support to get it.

String cardholderName

↳ Cardholder name, to be presented on the confirmation screen (see screen example on comments).

String panLastFour

↳ Last 4 Digits of the PAN to be presented on the confirmation screen.

String cardDescr

↳ Card description to be presented on the confirmation screen.

String panID

↳ PAN Reference ID. Can be obtained from object Card returned on function GetCards().

String pnp

↳ Payment Network Provider (“visa”, “mastercard” or “amex”).

String encCard

↳ Encrypted card information. See comments.

CommEvents events

↳ Instance of the class CommEvents used to callback the App. The operation result will be returned on onPostExecute method().

Return: None.

Requirements: None.

Comments: encCard contains a String JWE encrypted CardInfo object (case sensitive fields). More details about cryptographic operation are described in section 7.

```
CardInfo
{
        "PAN": "1234123412341234",
        "cardholderName": "Jose",
        "expirationDate": {
                "month": "05",
```

Exception:	"year": "2024" } }

Table 6 – Execute Provisioning API

4.2.7. Update DataBase

Description:	Updates the local database with non-sensitive cards information. Data will be used in In-App Provisioning Extension flow, described in section 9.
Prototype:	**public func** updateDataBase (`cardDataList:` [CardDataModel])
Input Param:	*CardDataModel[] cardDataList* ↳ List of cards to be updated in local database.
Return:	None.
Requirements:	None.
Comments:	None.
Exception:	None.

Table 7 –Update DataBase API

4.3. Classes and Enumerations

4.3.1. Card Class

Indicates the card information.

String CardId	Card identifier.
Enum CardType BARCODE PAYMENT CARD ANY	Card type.
String serialNumber	A value that uniquely identifies the card.
Boolean isRemote	True if card is hosted on a connected device. False if the card is hosted on the phone.
String deviceName	The name of the device hosting the card.
UIImage Icon	Icon that identifies the card.
String organizationName	The name of the organization that created the card.

Date relevantDate	The date when the card is most likely to be needed or useful.
String tokenId	Token ID (this is not the real Token)
String tokenLastFour	Token last 4 digits
String panId	PAN Reference ID
String panLastFour	PAN last 4 digits
CardStatus enum activationState	The activation status of the card.

Table 8 – Card Class

4.3.2. CardStatus Enum

Indicates the card payment status which could be any of the below codes.

UInt CardStatus (enum)	The current activation state of the card payment.
ACTIVATED	
REQUIRES_ACTIVATION	
ACTIVATING	
SUSPENDED	
DEACTIVATED	

Table 9 – CardStatus Enum

4.3.3. CommEvents Class

This class contains the callback functions for the provisioning process.

```
open class CommEvents
{
        public init() {}
        open func onPreExecute() {}
        open func onPostExecute(result: CommEventResult) {}
}
```

4.3.4. CommEventResult Class

Retrieves provisioning events information.

Int Result	Response code of the operation. It is defined in class HP2Errors.

Int Detail	Detailed response code of the operation. It is defined in class HP2ErrorDetails.
String Message	Friendly string representing response code.
AnyObject? ResultObject	Object containing additional information about the execution of the operation. If no additional information is needed, this object is null.

Table 10 – CommEeventResult Class

4.3.5. CardType Enum

Indicates the card status

CardType (enum)	The type of the card
BARCODE	
PAYMENT_CARD	
SECURE_ELEMENT	
ANY	

Table 11 – CardType Enum

4.3.6. HP2Errors Class

Static Int	**Description**
0	SUCCESS
-1	AUTH_ERROR
-99	GENERIC ERROR
1	INVALID_DATA
2	UNSUPPORTED_VERSION
3	INVALID_SIGNATURE
4	NOT_ENTITLED
5	UNSUPPORTED
6	USER_CANCELLED
7	SYSTEM_CANCELLED

Table 12 – Error Codes

4.3.7. HP2ErrorDetails Class

Static Int	Description
0	SUCCESS
-99	UNKNOWN
-1	AUTH_ERROR
1	INVALID_DATA
2	UNSUPPORTED_VERSION
3	INVALID_SIGNATURE
4	NOT_ENTITLED
5	UNSUPPORTED
6	USER_CANCELLED
7	SYSTEM_CANCELLED

Table 13 – Detailed Error Codes

4.3.8. CardDataModel Class

Indicates the card data to be used in In-app provisioning Extension flow.

String cardHolderName	The cardholder's name required to build Apple Pay add card UI.
String cardID	An ID used to identify the card in the Issuer's systems.
String cardImageBase64	The card image encoded in base 64 required to build Apple Pay add card UI.
String lastFourDigits	The card last four numbers required to build Apple Pay add card UI.
String localizedDescription	The card description required to build Apple Pay add card UI.
String paymentNetwork	The payment network associated with the card (example: "Mastercard", "Visa").

Table 14 – Card Class

5. Backend APIs (Deprecated - Option A and Option B)

~~5.1. Connectivity~~

~~The inbound and outbound APIs are designed as RPC style stateless webservices where each API endpoint represents an operation service published that only can be performed using **JSON** payload format. All strings in request and response are UTF-8 encoded and may have a version number API, which allows multiple versions of concurrent APIs to be deployed simultaneously.~~

~~Table 01 defines the supported HTTP response codes.~~

~~**Error Code**~~	~~**Description**~~
~~200~~	~~Success~~
~~400~~	~~Invalid request~~
~~401~~	~~Request Denied~~
~~403~~	~~Not allowed~~
~~404~~	~~Not found~~
~~500~~	~~Internal server error~~
~~501~~	~~Not implemented~~
~~503~~	~~Service not available~~

~~Table 15 – HTTP Response Codes~~

~~5.2. URL Scheme~~

~~The URL API follows the scheme bellow:~~

~~scheme://host[:port]/version/apiName~~

~~**URL ELEMENT**~~	~~**Definition**~~
~~Scheme~~	~~HTTPS~~
~~host[:port]~~	~~Described in the sections below~~
~~Version~~	~~v3~~
~~API~~	

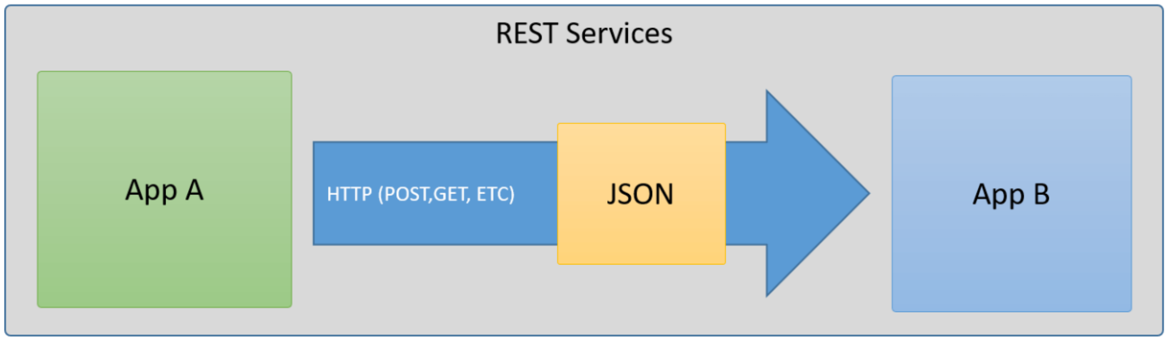
~~5.3. Key Management~~

~~The process of exchanging client/server certificates for the establishment of mutual authentication in TLS 1.2 will be performed by HST (Compliance) and Issuers during the project initial steps.~~

~~There is a specific procedure to follow to initiate the certificates exchange process that will be shared with the responsible contact of the Issuer. All the communication will be performed using the kms@hst.com.br e-mail.~~

~~5.4. Software Architecture and Technology~~

~~The inbound and outbound APIs must be implemented/invoked using **REST API JSON** style.~~



~~Implementation using SOAP (XML schemas) **MUST NOT** be used.~~

~~5.5. Onboarding HST Cloud~~

~~**Definition of Parameters**~~

- ~~**Financial Institution Code:** Unique Code defined by HST during Issuer Onboarding that identifies the Issuer at HST Pay Token Services.~~
- ~~**Sensitive Issuer Key (SIK):** It is an AES key generated by HST in its HSM during onboarding and shared between Issuer and HST by kms@hst.com.br e-mail, explaining the process.~~

~~**Notes:**~~

1. ~~Information about the SIK used in testing environment and other dynamic parameters for each issuer will be provided in a specific document.~~
2. ~~The EncryptedData used in the JSON examples provided in this document were ciphered using the following testing SIKs:~~
3. ~~**If the Issuer already has a SIK being used by HST Issuer Server, this process will not be required, and the same key will be used.**~~

- ~~**AES-128 key type:** "404142434445464748494A4B4C4D4E4F".~~
- ~~**AES-256 key type:** "404142434445464748494A4B4C4D4E4F4F4E4D4C4B4A494847464544434241 40".~~

~~5.6. Application Program Interfaces (APIs) - Outbound~~

~~The OutBound interface functions are called during card digitization, when an Issuer has to be notified about a token status change or to authenticate a user and retrieve available cards associated to the user.~~

~~5.6.1. GetPushCard~~

~~This API is invoked after HST Issuer Server receives the Receipt ID from the Issuer. It returns card data related to the card selected by the cardholder to be added in Apple Pay.~~

~~**API endpoint**~~	~~**Method**~~
~~**Sandbox**: https://{sandbox-issuer-host:port}/api/v3/getpushcard~~	~~POST~~
~~**Production**: https://{issuer-host:port}/api/v3/getpushcard~~	~~POST~~

~~GetPushCardRequest~~

~~Element:~~	~~**requestID**~~
~~Description:~~	~~Request identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**institutionCode**~~
~~Description:~~	~~Institution identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**pushReceiptID**~~
~~Description:~~	~~Push Receipt Identifier. This information is returned from the Inbound API *GetPushReceipt*.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~

~~GetPushCardResponse~~

~~Element:~~	~~**requestID**~~
~~Description:~~	~~Request identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**pushReceiptID**~~
~~Description:~~	~~Push Receipt Identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**returnCode**~~
~~Description:~~	~~Return Code: "00" for OK.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**errorDescription**~~
~~Description:~~	~~Error description returned only in error conditions.~~
~~Type:~~	~~String~~
~~Required:~~	~~Conditional~~
~~Element:~~	~~**encryptedCardInfo**~~
~~Description:~~	~~Encrypted CardInfo object related to the selected cards The credential list must be serialized and set in the EncryptedPayload object (string element encryptedData).~~
~~Type:~~	~~EncryptedPayload~~
~~Required:~~	~~Conditional~~
~~Element:~~	~~**cardProduct**~~
~~Description:~~	~~Unique identifier of the card product as registered on the platform. These values are defined during Issuer Onboarding and is out of the scope of this document.~~

~~Type:~~	~~String~~
~~Required:~~	~~Conditional~~

~~JSON Examples~~

~~GetPushCardRequest~~

```
~~{~~
~~    "requestID":"4",~~
~~    "institutionCode":"HST",~~
~~    "pushReceiptID":"1643ef957-622d-4137-abdf-fa605e81e72c"~~
~~}~~
```

~~GetPushCardResponse~~

```
~~{~~
~~    "requestID":"4",~~
~~    "pushReceiptID":"1643ef957-622d-4137-abdf-fa605e81e72c",~~
~~    "returnCode":"00",~~
~~    "encryptedCardInfo":{~~
~~        "algorithm":"aes-ccm128",~~
~~        "nonce":"71a7605140a38651d0ac42c5",~~
~~        "encryptedData":"ldf4DE4xtTxNPoQZilN9sEKJbef8Hu9cMfIu3kx9m0ot3sLtbE4UvFMJg~~
~~TXko1EBA+teLXHTQOhn/xR9aWfvnJuJ0o7jcUQvE7Rr1JVSCrdF8NQyb1E6EMdeU2j3T5NZ/VksIaQLB~~
~~saafXyc7S26y2HPs7DIkUSlt6RG+qOdw7kgDSufGniIudyxlomKiMXrrhoV/Y1SLhlPHhIn2rq5q5rUa~~
~~KSnIYOYpsvrzs32JwCMCuxaVC6+fSbx",~~
~~        "MACLength":16,~~
~~   "cardProduct":"153576579"~~
~~    }~~
~~}~~
```

```
~~Where:~~

~~//Plain CardInfo Object Data:~~

~~{~~
~~    "PAN":"1111110000000003",~~
~~    "expirationDate":{~~
~~        "month":"11",~~
~~        "year":"2024"~~
~~    },~~
~~    "cardholderName":"FRANCISCO PEREIRA"~~
~~}~~
```

~~5.7. Tokenization BUS (Inbound)~~

~~The HST Tokenization BUS webservice is designed to allow issuers to directly with the Issuer server.~~

~~5.7.1. GetPushReceipt~~

~~This API is used to create a Receipt ID, which is necessary to identify in HST Issuer Server a card push provisioning request.~~

~~**API endpoint**~~	~~**Method**~~
~~**Sandbox:** https://sandbox-issuer-bus-tokenization.com:9205/api/v3/getpushreceipt~~	~~POST~~
~~**Production:** https://issuer-bus-tokenization.com:9205/api/v3/getpushreceipt~~	~~POST~~

~~GetPushReceiptRequest~~

~~Element:~~	~~**requestID**~~
~~Description:~~	~~Request identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**institutionCode**~~
~~Description:~~	~~Institution identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**encryptedCardInfo**~~
~~Description:~~	~~Encrypted CardInfo related to the card being digitized.~~
~~Type:~~	~~EncryptedPayload.~~
~~Required:~~	~~Conditional~~

~~GetPushReceiptResponse~~

~~Element:~~	~~**requestID**~~
~~Description:~~	~~Request identifier.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~
~~Element:~~	~~**returnCode**~~
~~Description:~~	~~Return Code: “00” for OK.~~
~~Type:~~	~~String~~
~~Required:~~	~~Yes~~

~~Element:~~	~~**errorDescription**~~
~~Description:~~	~~Error description returned only in error conditions.~~
~~Type:~~	~~String~~
~~Required:~~	~~Conditional~~
~~Element:~~	~~**pushReceiptID**~~
~~Description:~~	~~Push Receipt Identifier created by Issuer Server.~~
~~Type:~~	~~String~~
~~Required:~~	~~Conditional~~
~~Element:~~	~~**receiptExpirationTime**~~
~~Description:~~	~~Date and time when the receipt will expire. The receipt generated will be valid up to 15 minutes after its creation.~~ ~~Format: yyyy-MM-ddTHH:mm:ss.SSSZ~~ ~~The value will be in GMT.~~
~~Type:~~	~~String~~
~~Required:~~	~~Conditional~~

~~GetPushReceiptRequest~~

```
~~{~~
~~    "requestID":"5",~~
~~    "institutionCode":"HST",~~
~~    "encryptedCardInfo":{~~
~~        "algorithm":"aes-ccm128",~~
~~        "nonce":"04f19e5de0661a0d88d5ba25",~~
~~        "encryptedData":"g95bUZ/F8YOadqLL9Vv/mfAql7Z1Znmv6Wt1qZnoyzz4ythhvKpVKx3ZW~~
~~dYn0meCqRE8BQGrLP5oaBwBxmuBwIjEWz2UKWXICKyjV6ari3XFXCB1ois1Dl2NxqvGqapR9jpjWNFN/~~
~~sw0zorh1I4TT7Zv1a/EKs5Mkw8K2QWBFCfppCIG8zphhoRk4oIn0ayE9ipB1WacoQuA/1lIaoYTdQKvp~~
~~F1Dh5wGTxWmL/xKTsb/wFTtquNNvvVIhEd0sG3O9sR6WxoFyzo7fBeySkhStmKxmhLyJaXLbr4LNutIM~~
~~3WOfms0MJyBca2qITtb2DjdhVoVBoNxVA+LCHC7/YarEEsVb1vC",~~
~~        "MACLength":16,~~
~~   "cardProduct":"153576579"~~
~~    }~~

~~}~~
```

~~Where:~~

~~**//Plain CardInfo Object Data:**~~

```
~~{~~
~~    "PAN":"1111110000000003",~~
~~    "expirationDate":{~~
~~        "month":"11",~~
~~        "year":"2024"~~
~~    },~~
~~    "cardholderName":"FRANCISCO PEREIRA"~~
~~}~~
```

~~GetPushReceiptResponse~~

```
~~{~~
~~    "requestID":"5",~~
~~    "returnCode":"00",~~
~~    "pushReceiptID":"1643ef957-622d-4137-abdf-fa605e81e72c",~~

    ~~"receiptExpirationTime":"2020-03-04 17:34:06.123"~~
~~}~~
```

~~5.8. General Objects~~

~~5.8.1. ExpirationDate~~

~~Element:~~	**~~Month~~**
~~Description:~~	~~Month of expiry date.~~
~~Type:~~	~~String~~
~~Size:~~	~~2~~
~~Required:~~	~~Y~~
~~Element:~~	**~~Year~~**
~~Description:~~	~~Year of expiry date (i.e. **XXXX**).~~
~~Type:~~	~~String~~
~~Size:~~	~~4~~
~~Required:~~	~~Y~~

~~5.8.2. CardInfo~~

~~Element:~~	**~~PAN~~**
~~Description:~~	~~Primary Account Value.~~
~~Type:~~	~~String~~
~~Size:~~	~~16-19~~
~~Required:~~	~~Y~~
~~Element:~~	**~~expirationDate~~**
~~Description:~~	~~Card expiration date.~~
~~Type:~~	~~ExpirationDate~~
~~Required:~~	~~C~~
~~Element:~~	**~~CVV2~~**
~~Description:~~	~~Card Verification Value presented on the back of the physical card.~~
~~Type:~~	~~String~~
~~Size:~~	~~3~~
~~Required:~~	~~C~~

~~Element:~~	~~**cardholderName**~~
~~Description:~~	~~Cardholder Name as it appears on card. Special characters or numbers are not valid.~~
~~Type:~~	~~String~~
~~Size:~~	~~Max 32~~
~~Required:~~	~~C~~

~~5.8.3. EncryptedPayload~~

~~Element:~~	~~**algorithm**~~
~~Description:~~	~~Encryption Algorithm used to protect data. Supported types are: "**none**", "**aes-gcm128**", "**aes-ccm128**", "**aes-gcm256**", "**aes-ccm256**". Refer to https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf and https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf~~ ~~**The issuers must implement at least two encryption algorithms: "none" and any other one. In case Issuer has chosen "none" as algorithm, Nonce and IV elements cannot be sent.**~~
~~Type:~~	~~String~~
~~Required:~~	~~Y~~
~~Element:~~	~~**nonce**~~
~~Description:~~	~~Nonce for AES_CCM.~~
~~Type:~~	~~String~~
~~Size:~~	~~7, 8, 9, 10, 11, 12 or 13 (if not sure, 11 should be used)~~
~~Required:~~	~~C~~
~~Element:~~	~~**iv**~~
~~Description:~~	~~Initial Vector for AES_GCM.~~
~~Type:~~	~~String~~
~~Size:~~	~~16~~
~~Required:~~	~~C~~
~~Element:~~	~~**encryptedData**~~
~~Description:~~	~~Encrypt Data value using SIK. **All the ciphered data must be transmitted in base64.**~~
~~Type:~~	~~String~~
~~Size:~~	~~0-256k~~
~~Required:~~	~~Y~~
~~Element:~~	~~**associatedData**~~
~~Description:~~	~~Data that is not encrypted but used for MAC calculation.~~
~~Type:~~	~~String~~
~~Size:~~	~~0-256K~~
~~Required:~~	~~C~~

~~Element:~~	~~**MACLength**~~
~~Description:~~	~~Specifies the MAC length that will be generated. The MAC contents are located at the end of the encryptedData element.~~ ~~Valid values for **CCM algorithm**: 4, 6, 8, 10, 12, 14 or 16 bytes (reasonable minimum is 12).~~ ~~Valid values for **GCM algorithm**: 4, 6, 8, 10, 12, 14 or 16 bytes (reasonable minimum is 12).~~
~~Type:~~	~~Numeric~~
~~Required:~~	~~Y~~

6. Apple Integration Requirements

This section describes the requirements to access Apple SDK and the steps for the integration during implementation.

6.1. iOS and Swift minimum versions

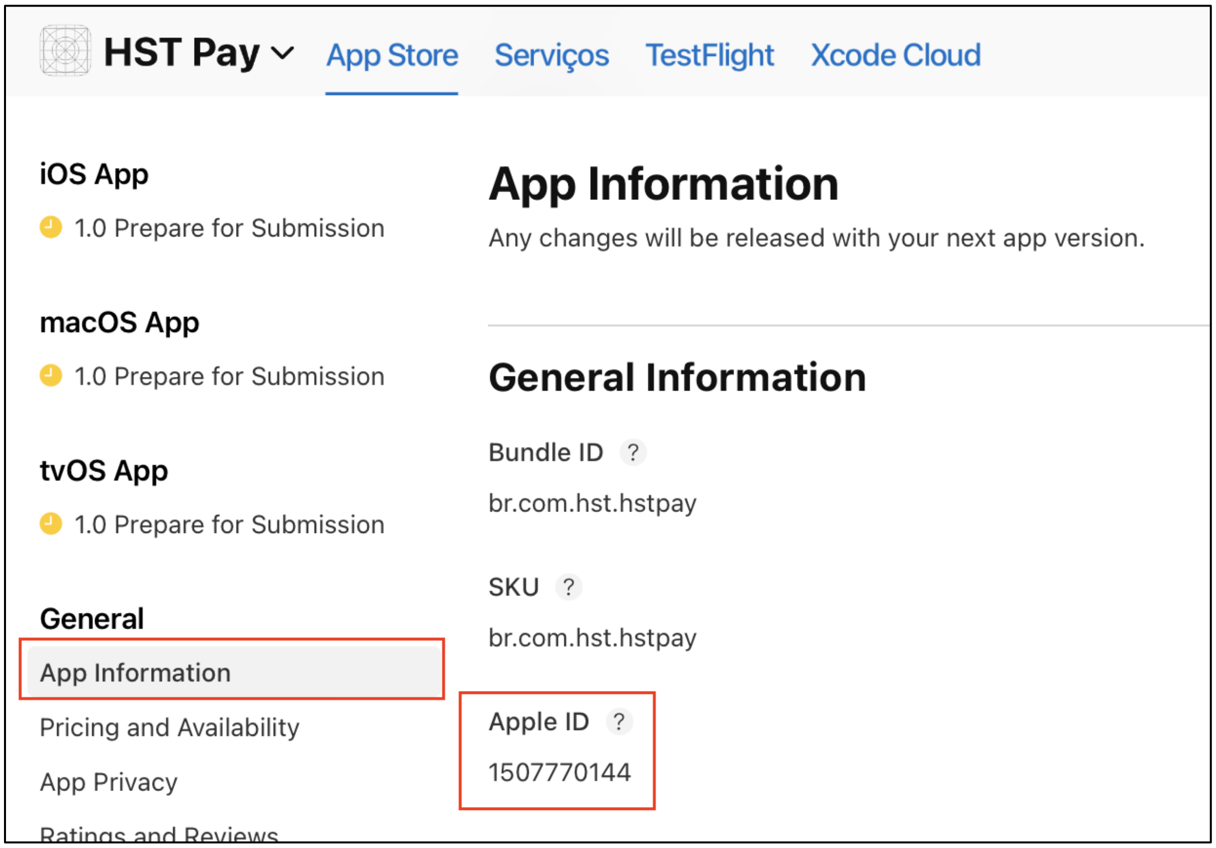
The minimum versions required for the integration of the Issuer App with Apple SDK are:

- **iOS: Version 9**
- **Swift: Version 4.2**

6.2. How to get Adam ID

Using App Store Connect, select App Information. Adam ID parameter corresponds to Apple ID, as shown in the following image:

https://appstoreconnect.apple.com/apps/1507770144/appstore/info:



HST Pay App Store Serviços TestFlight Xcode Cloud

iOS App
1.0 Prepare for Submission

macOS App
1.0 Prepare for Submission

tvOS App
1.0 Prepare for Submission

General
App Information
Pricing and Availability
App Privacy
Ratings and Reviews

App Information

Any changes will be released with your next app version.

General Information

Bundle ID ?
br.com.hst.hstpay

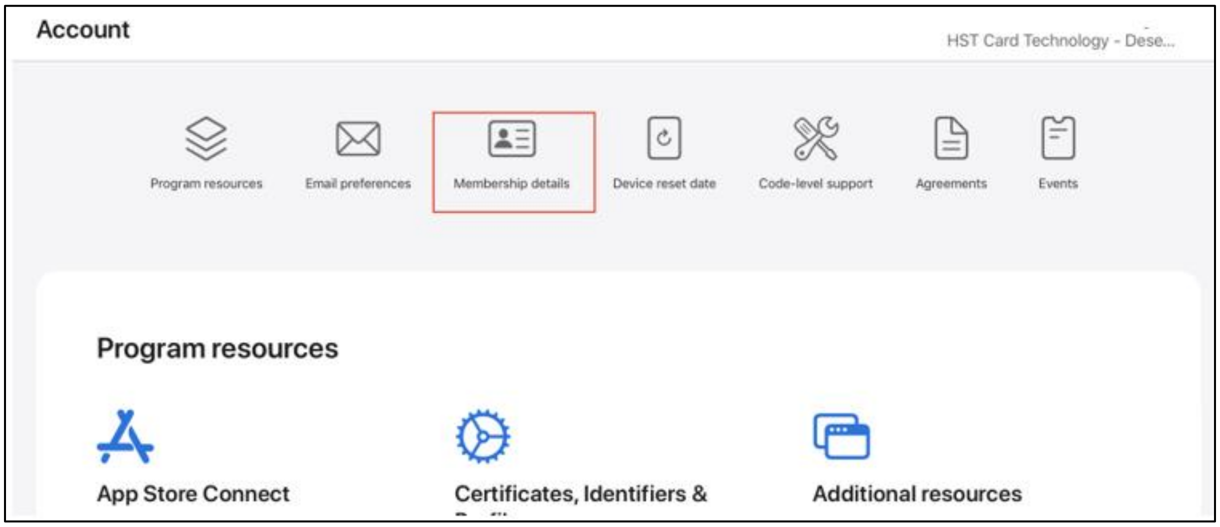
SKU ?
br.com.hst.hstpay

Apple ID ?
1507770144

6.3. How to get Team ID

In Developer Apple, click on *Membership details.* Team ID parameter can be found as shown in the following images:

https://developer.apple.com/account/



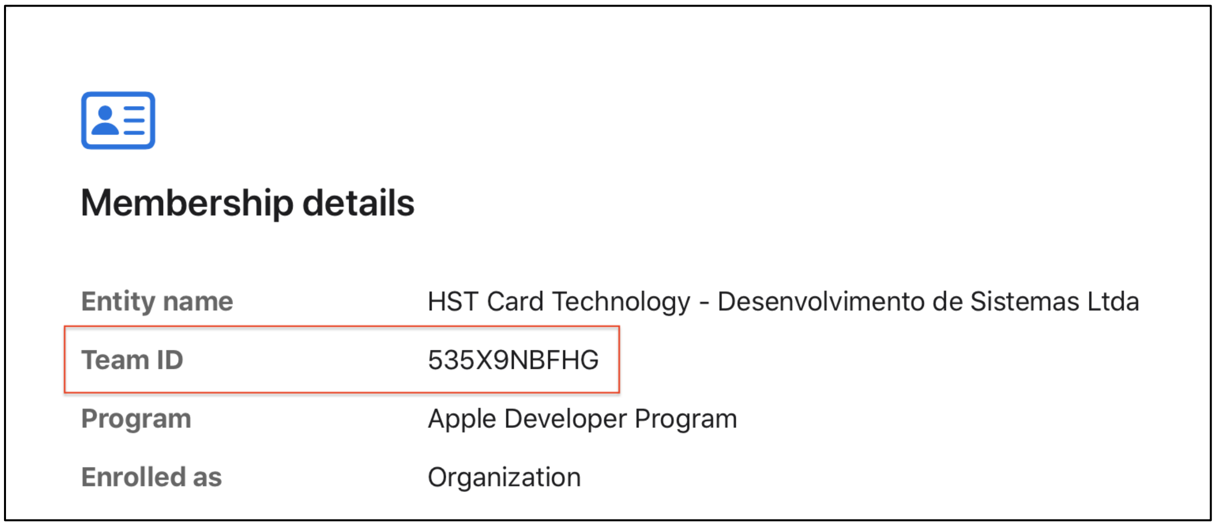
Account

HST Card Technology - Dese...

Program resources · Email preferences · Membership details · Device reset date · Code-level support · Agreements · Events

Program resources

App Store Connect · Certificates, Identifiers & · Additional resources


Membership details

Entity name	HST Card Technology - Desenvolvimento de Sistemas Ltda
Team ID	535X9NBFHG
Program	Apple Developer Program
Enrolled as	Organization

6.4. Requesting access for Apple

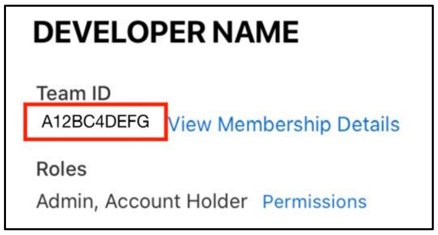
During the implementation process Apple requires that the Issuer follows a few requisites to request permission to use Apples SDK for the implementation process of the Push Provisioning flow.

- The first step required by the Issuer is to sign an agreement with Apple for the project.
- The issuer must receive a special entitlement distributed to the Team ID by Apple.
- App Apple ID (Adam ID) whitelisted by Apple.

Only production Team IDs and Adam IDs can receive the entitlement and can be whitelisted.
To request the entitlement and whitelist for Issuers App, the Issuer must send the information below by email to apple-pay-provisioning@apple.com:

- **Issuer Name and Country Code**
- **App Name**
- **Team ID** (*e.g. 1ABCD2FGHI*)
- **Adam ID** (*e.g. 123456789*)

The Team ID and Adam ID can be found in the iTunes Connect portal. An example of Team ID is shown below.


DEVELOPER NAME

Team ID
A12BC4DEFG

Roles
Admin, Account Holder

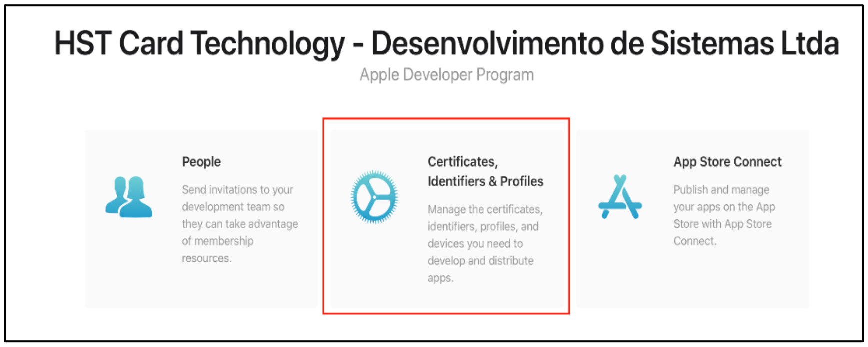
After that, the next section describes the steps of how to request access to Apple Pay and integrates the SDK for the push provisioning flow.

6.5. Creating entitlement profile

Once the entitlements have been granted, it is necessary to include the distribution entitlement into a provisioning profile and ensure you are leveraging the same profile to develop the app within Xcode.
Go to Apple Developer and follow the steps bellow.

https://developer.apple.com/account/resources/profiles/add

a) Click "Certificates, Indentifiers & Profiles"

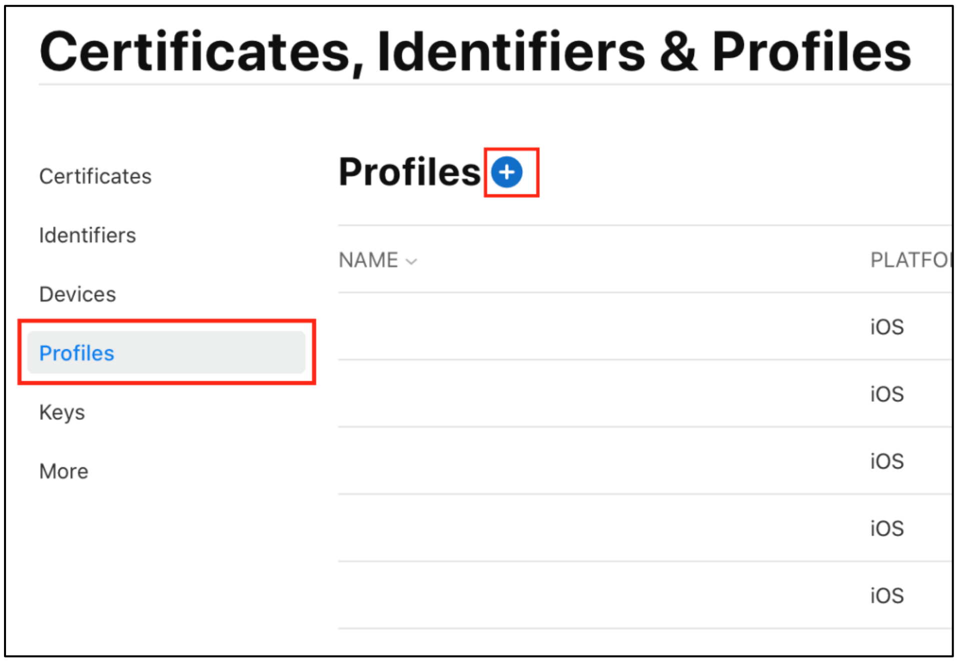

HST Card Technology - Desenvolvimento de Sistemas Ltda
Apple Developer Program

People
Send invitations to your development team so they can take advantage of membership resources.

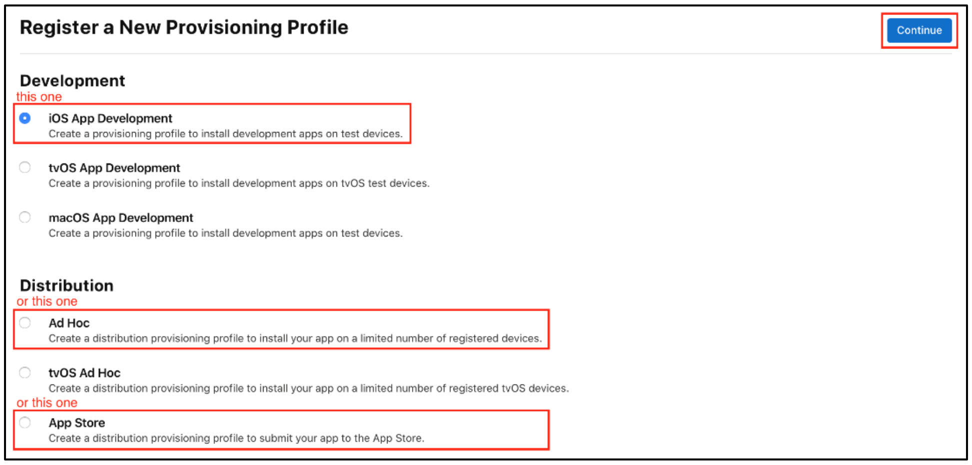
Certificates, Identifiers & Profiles
Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.

App Store Connect
Publish and manage your apps on the App Store with App Store Connect.

Select "Profiles" and then click at plus icon



b) Select one of de options "iOS App development", "Ad Hoc" or "App Store" and click "Continue" button.


Register a New Provisioning Profile

Development

this one

iOS App Development
Create a provisioning profile to install development apps on test devices.

tvOS App Development
Create a provisioning profile to install development apps on tvOS test devices.

macOS App Development
Create a provisioning profile to install development apps on test devices.

Distribution

or this one

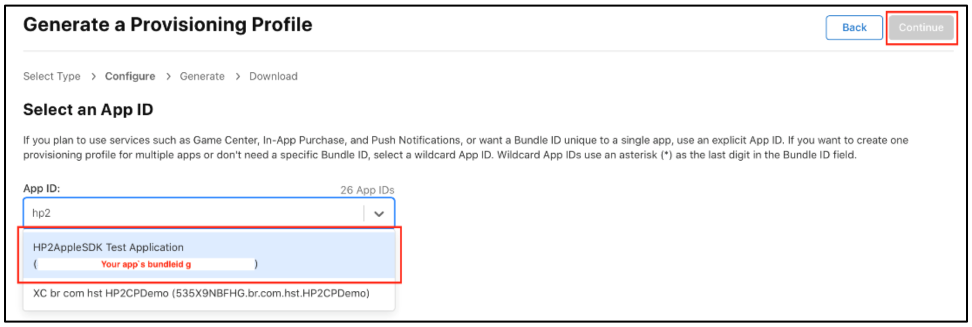
Ad Hoc
Create a distribution provisioning profile to install your app on a limited number of registered devices.

tvOS Ad Hoc
Create a distribution provisioning profile to install your app on a limited number of registered tvOS devices.

or this one

App Store
Create a distribution provisioning profile to submit your app to the App Store.

c) In the "App ID" field select your App with the right bundleid


Generate a Provisioning Profile

Select an App ID

If you plan to use services such as Game Center, In-App Purchase, and Push Notifications, or want a Bundle ID unique to a single app, use an explicit App ID. If you want to create one provisioning profile for multiple apps or don't need a specific Bundle ID, select a wildcard App ID. Wildcard App IDs use an asterisk (*) as the last digit in the Bundle ID field.

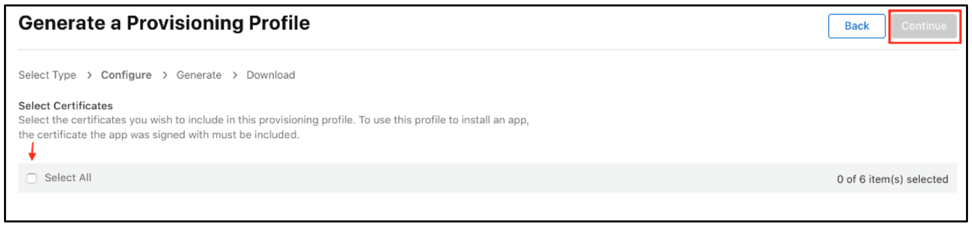
App ID: 26 App IDs

hp2

HP2AppleSDK Test Application
(Your app's bundleid g)

XC br com hst HP2CPDemo (535X9NBFHG.br.com.hst.HP2CPDemo)

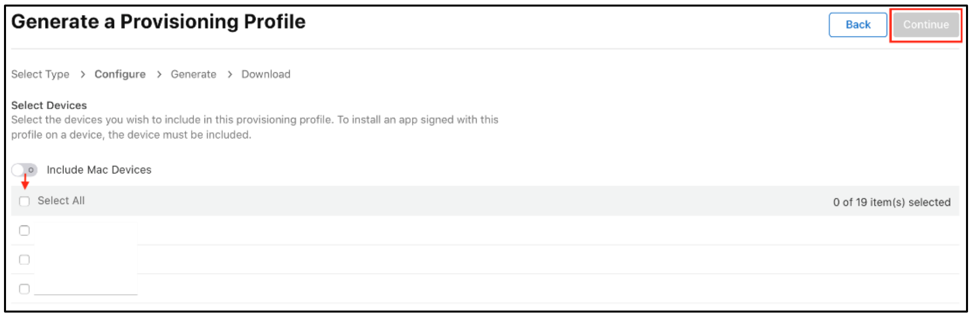
d) Select the certificates you wish to include in this provisioning profile and click "continue" button.


Generate a Provisioning Profile

Select Certificates
Select the certificates you wish to include in this provisioning profile. To use this profile to install an app, the certificate the app was signed with must be included.

Select All 0 of 6 item(s) selected

e) Select the devices you wish to include in this provisioning profile.

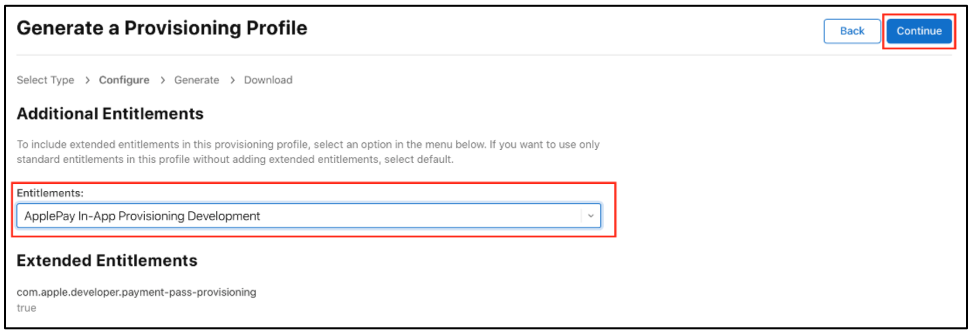

Generate a Provisioning Profile

Select Devices
Select the devices you wish to include in this provisioning profile. To install an app signed with this profile on a device, the device must be included.

Include Mac Devices

Select All 0 of 19 item(s) selected

f) Select the appropriated entitlement and click "Continue" button



Generate a Provisioning Profile

Select Type › Configure › Generate › Download

Additional Entitlements

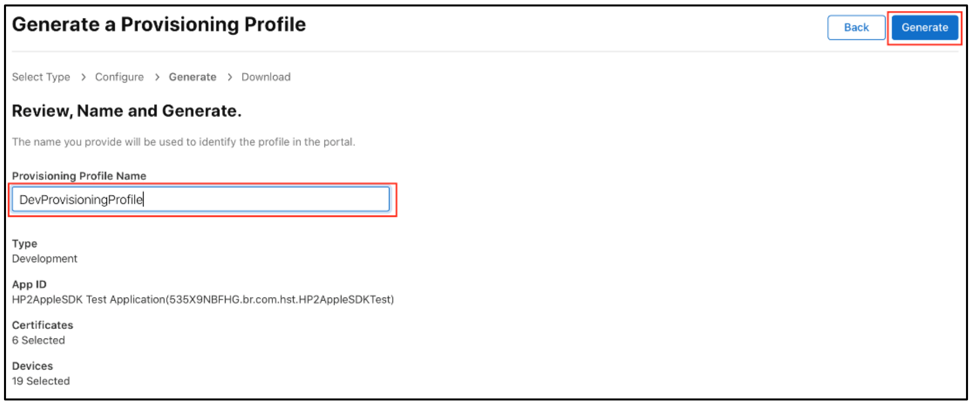
To include extended entitlements in this provisioning profile, select an option in the menu below. If you want to use only standard entitlements in this profile without adding extended entitlements, select default.

Entitlements: ApplePay In-App Provisioning Development

Extended Entitlements

com.apple.developer.payment-pass-provisioning
true

g) Name the provisioning profile and click "Generate”.



Generate a Provisioning Profile

Select Type › Configure › Generate › Download

Review, Name and Generate.

The name you provide will be used to identify the profile in the portal.

Provisioning Profile Name: DevProvisioningProfile

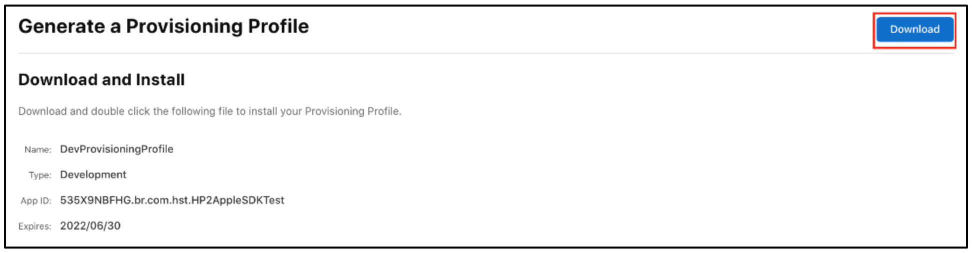
Type
Development

App ID
HP2AppleSDK Test Application(535X9NBFHG.br.com.hst.HP2AppleSDKTest)

Certificates
6 Selected

Devices
19 Selected

h) Click "Download" button, it will download the provisioning profile to your MAC.



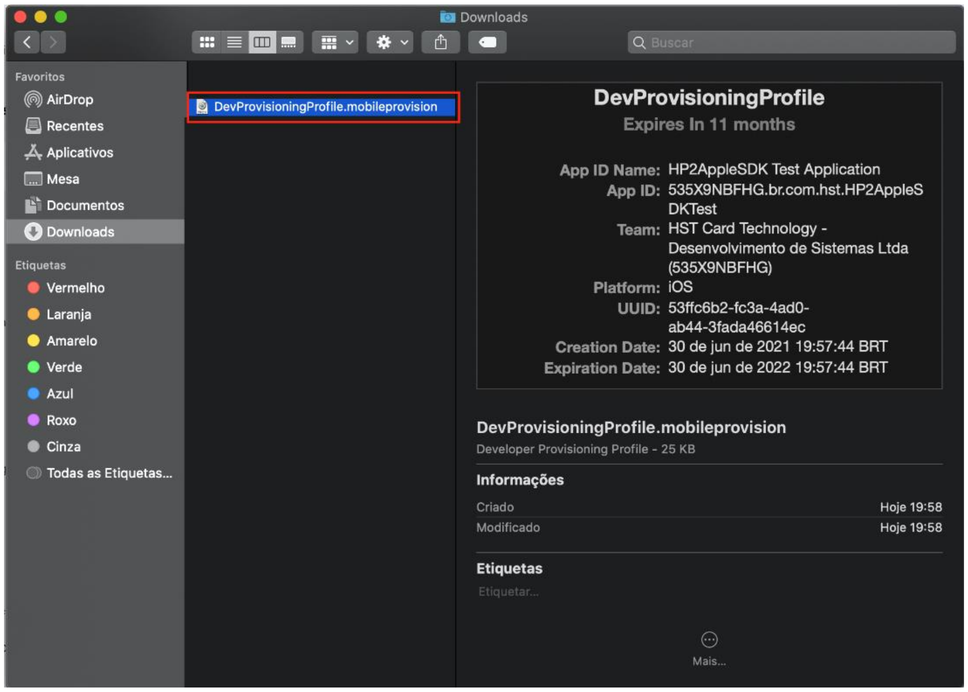
Generate a Provisioning Profile

Download and Install

Download and double click the following file to install your Provisioning Profile.

Name:	DevProvisioningProfile
Type:	Development
App ID:	535X9NBFHG.br.com.hst.HP2AppleSDKTest
Expires:	2022/06/30

i) Locate the file you`ve been downloaded and do a double click. It`ll open the Xcode.



DevProvisioningProfile
Expires In 11 months

App ID Name: HP2AppleSDK Test Application
App ID: 535X9NBFHG.br.com.hst.HP2AppleSDKTest
Team: HST Card Technology - Desenvolvimento de Sistemas Ltda (535X9NBFHG)
Platform: iOS
UUID: 53ffc6b2-fc3a-4ad0-ab44-3fada46614ec
Creation Date: 30 de jun de 2021 19:57:44 BRT
Expiration Date: 30 de jun de 2022 19:57:44 BRT

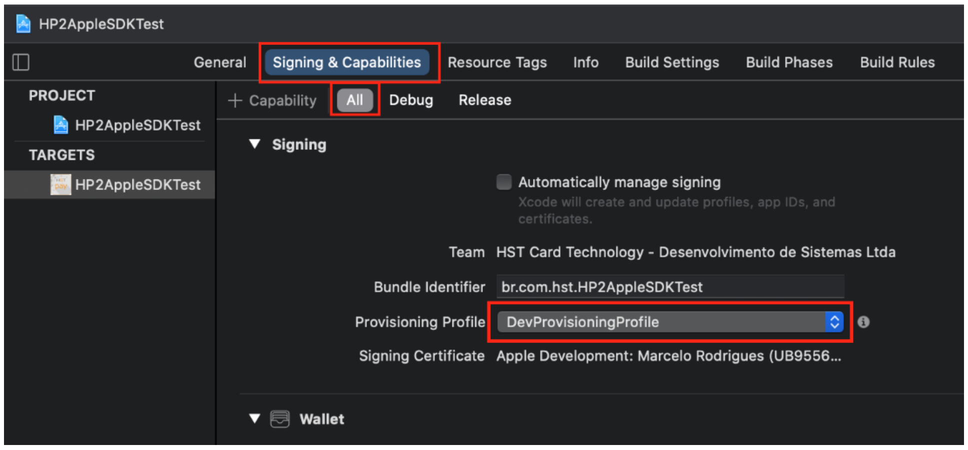
DevProvisioningProfile.mobileprovision
Developer Provisioning Profile - 25 KB

Informações

Criado	Hoje 19:58
Modificado	Hoje 19:58

Etiquetas
Etiquetar...

j) Open your project on the Xcode and select the profile in "Signing & Capabilities" -> "All" -> "Provisioning Profile".



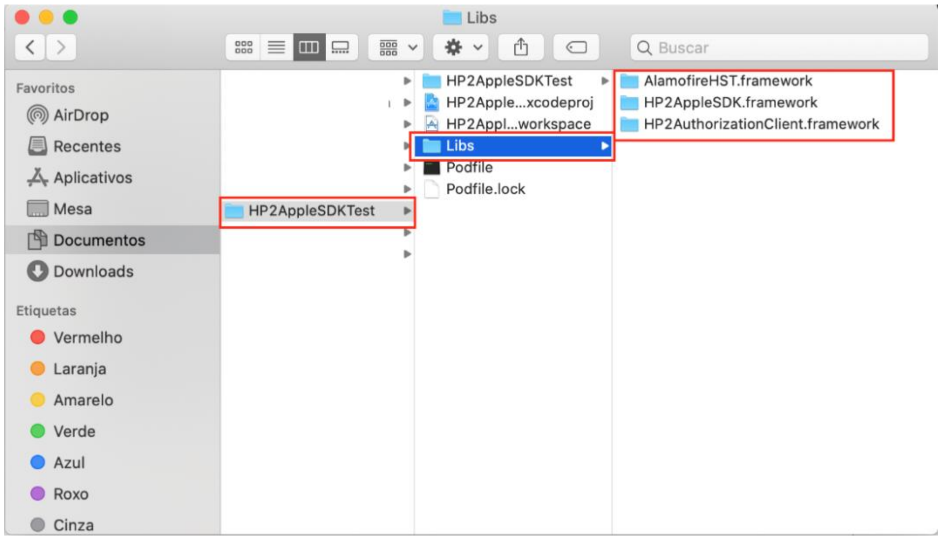
Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team HST Card Technology - Desenvolvimento de Sistemas Ltda
Bundle Identifier br.com.hst.HP2AppleSDKTest
Provisioning Profile DevProvisioningProfile
Signing Certificate Apple Development: Marcelo Rodrigues (UB9556...

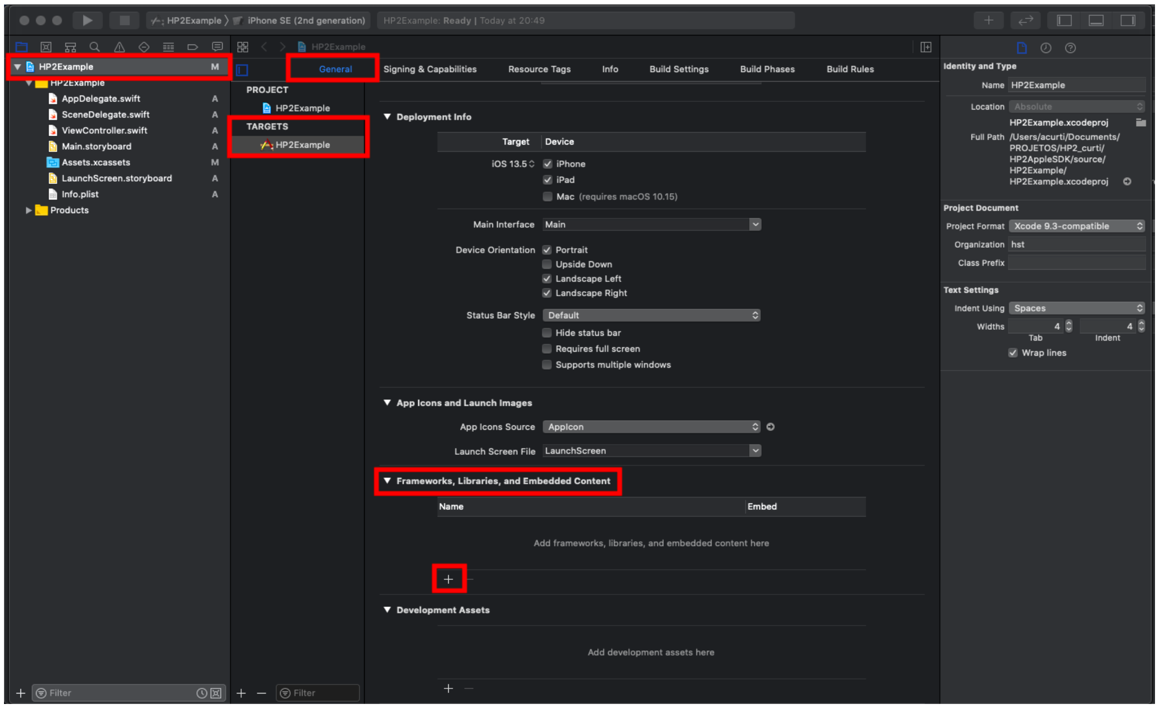
6.6. Adding HP2 frameworks to your project.

To correctly add HP2 frameworks in a Xcode project follow the steps below.

Create a folder called Libs in your project folder if it does not exist and copy HP2 Framework to this folder.



a) Open your xcode project select your *".xcodeproj*" at the top left, check if right target is selected, select general tab and in "Frameworks, Libraries, and Embedded Content" click Add (**+**).



Deployment Info

Target: iOS 13.5 — Device: [✓ iPhone] [✓ iPad] Mac (requires macOS 10.15)

Main Interface: Main

Device Orientation: [✓ Portrait] Upside Down [✓ Landscape Left] [✓ Landscape Right]

Status Bar Style: Default

Hide status bar
Requires full screen
Supports multiple windows

App Icons and Launch Images

App Icons Source: AppIcon

Launch Screen File: LaunchScreen

Frameworks, Libraries, and Embedded Content

Name	Embed

Add frameworks, libraries, and embedded content here

Development Assets

Add development assets here

Identity and Type

Name: HP2Example

Location: Absolute

HP2Example.xcodeproj

Full Path: /Users/acurti/Documents/PROJETOS/HP2_curti/HP2AppleSDK/source/HP2Example/HP2Example.xcodeproj

Project Document

Project Format: Xcode 9.3-compatible

Organization: hst

Class Prefix:

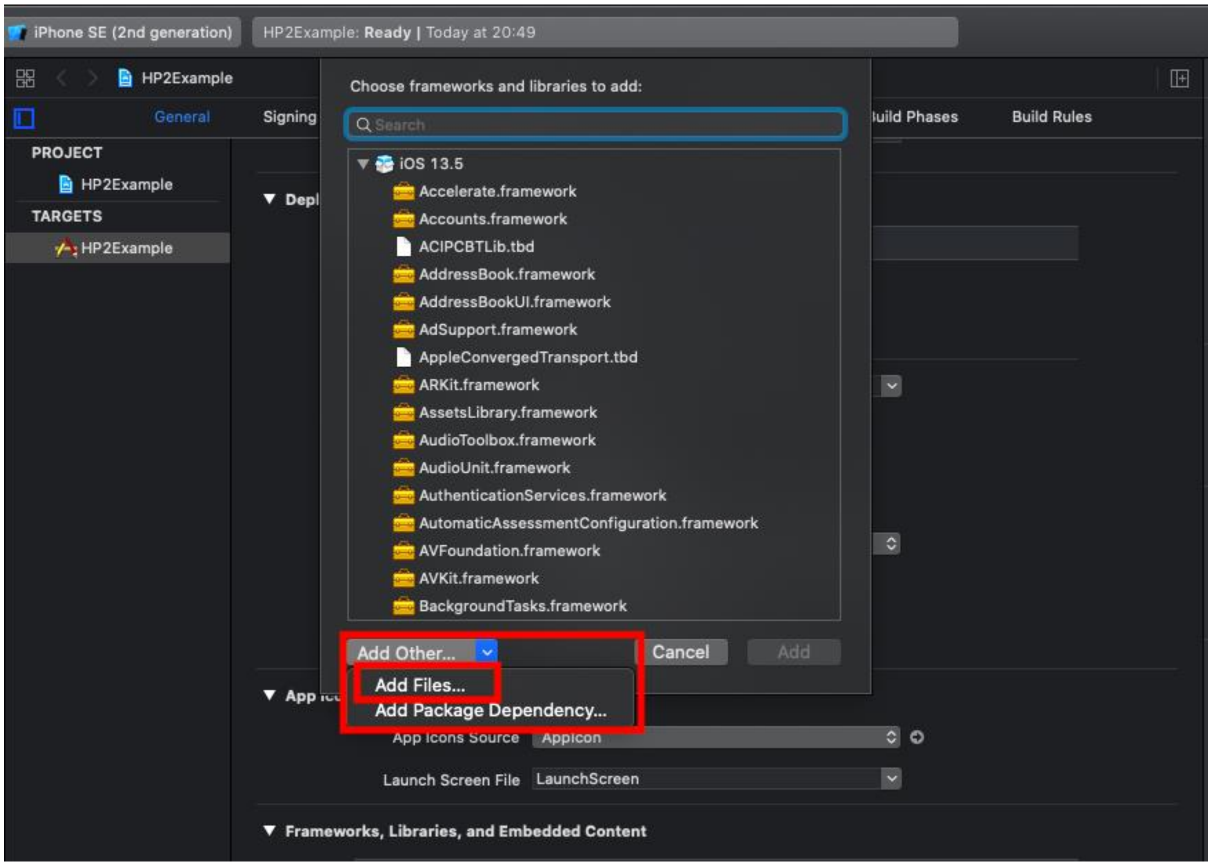
Text Settings

Indent Using: Spaces

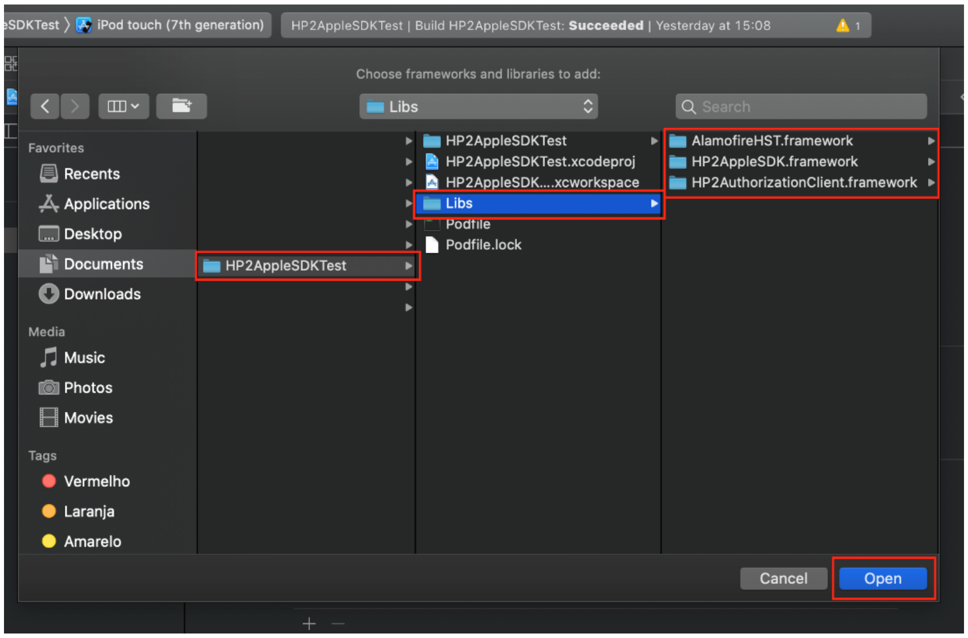
Widths: 4 Tab, 4 Indent

[✓ Wrap lines]

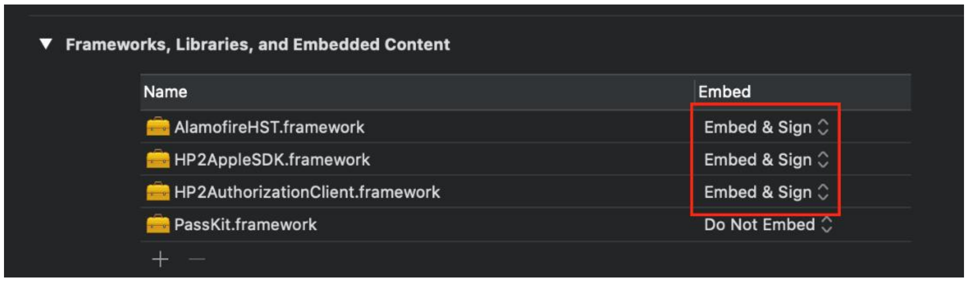
b) At the dialog screen click "*Add Other*" > "*Add Files...*".



c) Find the folder where you put the HP2 frameworks in your project folder, select all frameworks and click open.


iPod touch (7th generation)
HP2AppleSDKTest | Build HP2AppleSDKTest: Succeeded | Yesterday at 15:08
Choose frameworks and libraries to add:
Libs
Search
Favorites
Recents
Applications
Desktop
Documents
Downloads
Media
Music
Photos
Movies
Tags
Vermelho
Laranja
Amarelo
HP2AppleSDKTest
HP2AppleSDKTest
HP2AppleSDKTest.xcodeproj
HP2AppleSDK....xcworkspace
Libs
Podfile
Podfile.lock
AlamofireHST.framework
HP2AppleSDK.framework
HP2AuthorizationClient.framework
Cancel
Open

Note that all frameworks are Embed & Sign.

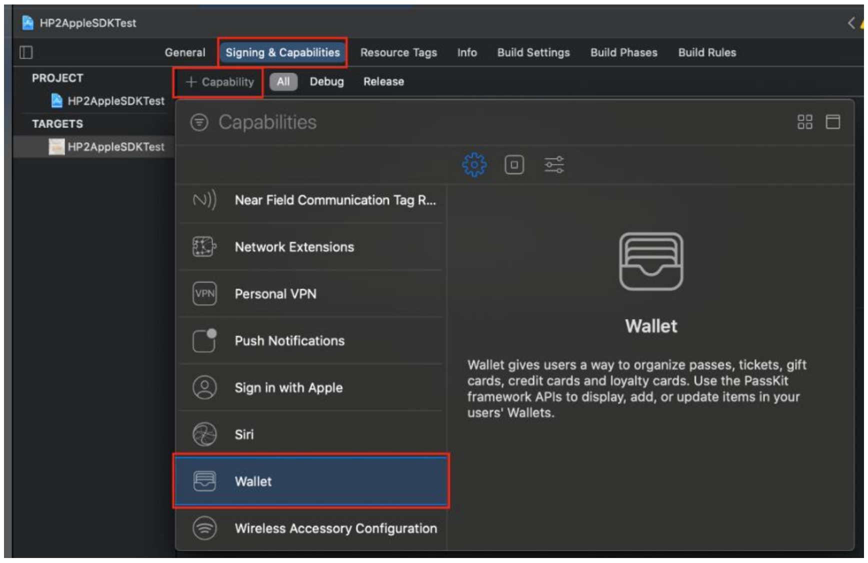


▼ Frameworks, Libraries, and Embedded Content

Name	Embed
AlamofireHST.framework	Embed & Sign
HP2AppleSDK.framework	Embed & Sign
HP2AuthorizationClient.framework	Embed & Sign
PassKit.framework	Do Not Embed

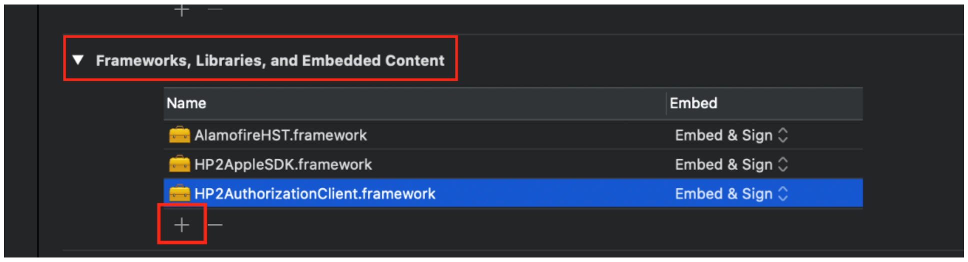
6.7. Adding Apple PassKit frameworks to your project

a) Click "*Signing & Capabilities*" tab, select "*All*", click *"+Capability*" and select "Wallet".



Wallet gives users a way to organize passes, tickets, gift cards, credit cards and loyalty cards. Use the PassKit framework APIs to display, add, or update items in your users' Wallets.

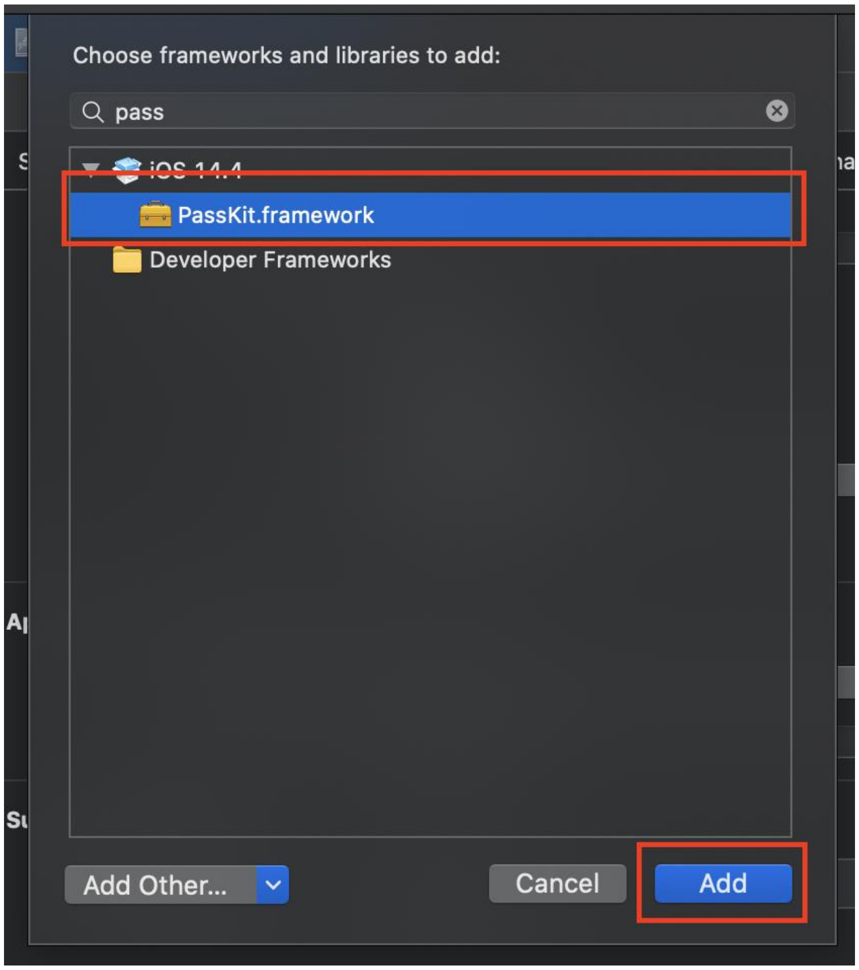
b) Select general tab and in "*Frameworks, Libraries, and Embedded Content*" click Add (**+**).



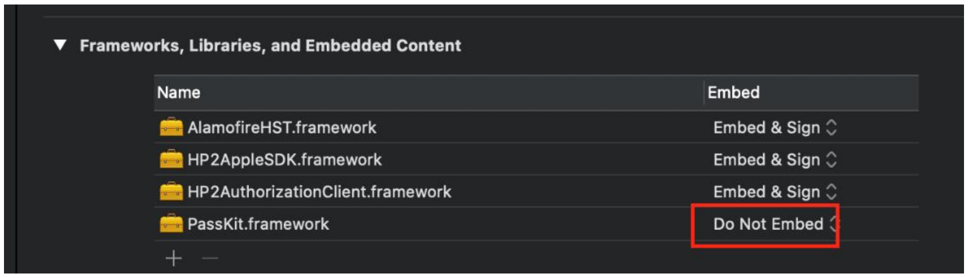
▼ Frameworks, Libraries, and Embedded Content

Name	Embed
AlamofireHST.framework	Embed & Sign
HP2AppleSDK.framework	Embed & Sign
HP2AuthorizationClient.framework	Embed & Sign

c) Search for pass kit, select PassKit framework and click Add.



Note the Passkit framework id listed but not Embeded.



▼ Frameworks, Libraries, and Embedded Content

Name	Embed
AlamofireHST.framework	Embed & Sign
HP2AppleSDK.framework	Embed & Sign
HP2AuthorizationClient.framework	Embed & Sign
PassKit.framework	Do Not Embed

6.8. Adding key to Issuer Application for card provisioning

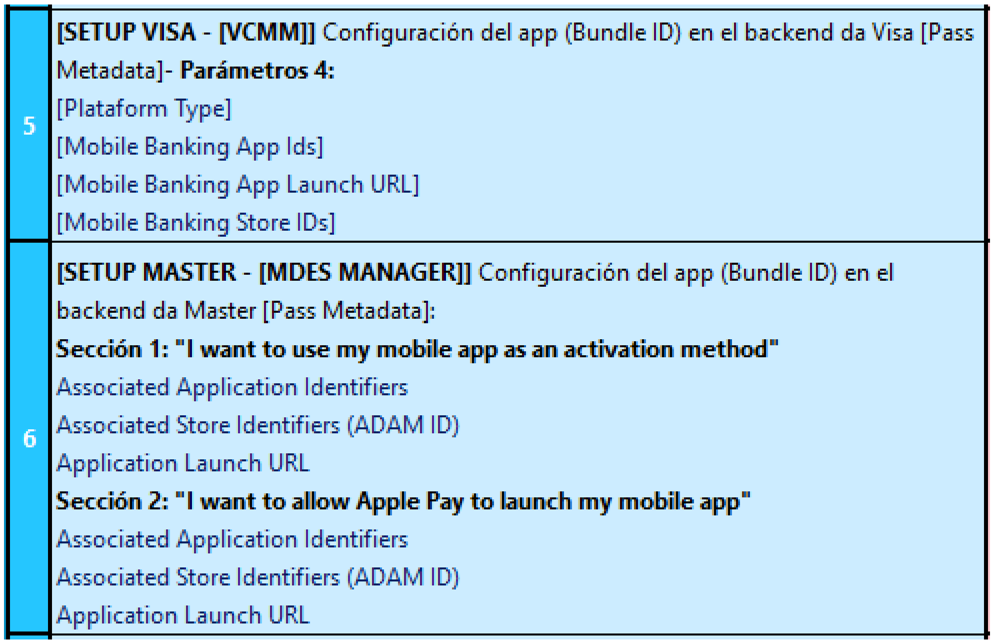
To show the Apple Wallet screen is necessary to add the key **com.apple.developer.payment-pass-provisioning** using value "*true*" on the entitlement file in the XCode project.

Example (red lines must be added):

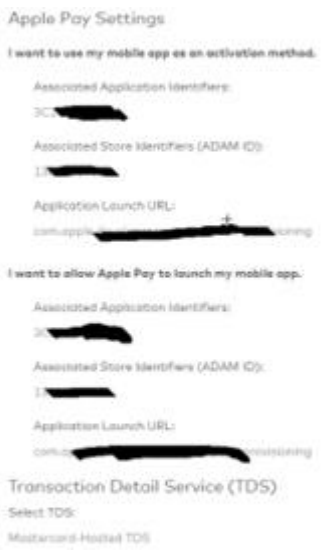
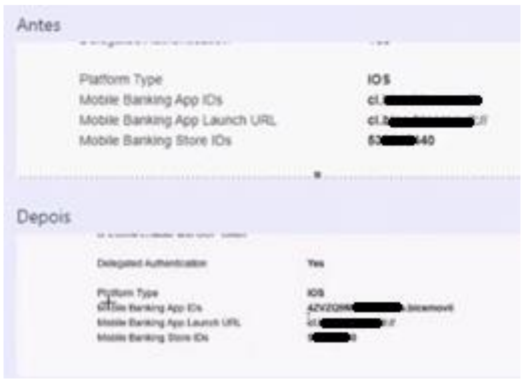
```
<key>com.apple.developer.pass-type-identifiers</key>
<array>
     <string>$(TeamIdentifierPrefix)*</string>
</array>
<key>com.apple.developer.payment-pass-provisioning</key>
<true/>
```

6.9. VCMM and MDES Manager Setup

Visa Card Metadata Manager (VCMM) and the MDES Manager platform require specific configurations which are described at following:



5	**[SETUP VISA - [VCMM]]** Configuración del app (Bundle ID) en el backend da Visa [Pass Metadata]- **Parámetros 4:** [Plataform Type] [Mobile Banking App Ids] [Mobile Banking App Launch URL] [Mobile Banking Store IDs]
6	**[SETUP MASTER - [MDES MANAGER]]** Configuración del app (Bundle ID) en el backend da Master [Pass Metadata]: **Sección 1: "I want to use my mobile app as an activation method"** Associated Application Identifiers Associated Store Identifiers (ADAM ID) Application Launch URL **Sección 2: "I want to allow Apple Pay to launch my mobile app"** Associated Application Identifiers Associated Store Identifiers (ADAM ID) Application Launch URL

MDES Manager	VCMM
 Apple Pay Settings **I want to use my mobile app as an activation method.** Associated Application Identifiers: 3C[redacted] Associated Store Identifiers (ADAM ID): 1[redacted] Application Launch URL: com.apple[redacted]ioning **I want to allow Apple Pay to launch my mobile app.** Associated Application Identifiers: 3[redacted] Associated Store Identifiers (ADAM ID): 1[redacted] Application Launch URL: com.[redacted]provisioning Transaction Detail Service (TDS) Select TDS: Mastercard-Hosted TDS	 Antes Platform Type: IOS Mobile Banking App IDs: cl.[redacted] Mobile Banking App Launch URL: cl.[redacted]:// Mobile Banking Store IDs: 53[redacted]40 Depois Delegated Authentication: Yes Platform Type: IOS Mobile Banking App IDs: 4ZVZQ[redacted].bicemovil Mobile Banking App Launch URL: cl.[redacted]:// Mobile Banking Store IDs: [redacted]

7. Cryptography information

The encrypted Card Info submitted by issuers in method executeProvisioningOfEncryptedCard (0) contain sensitive information that must be encrypted using JWE (JSON Web Encryption), **if the app is not using this method this section must be ignored.** The following site can be used to help find a security library to implement the JWE protocol https://jwt.io/ (HST is not responsible by this implementation or any third-party library) or refer to https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-40 for more details and complete specifications on JWE.

The general approach for encrypting data using JWE and RSA/AES is as follows:

- It is expected to be used the compact serialization style, wherein elements are separated by a "."(period).
- All fields are Base64URL-encoded.
- HST uses the hybrid encryption scheme. In this method, an RSA 2048 key is used to encrypt.
- a random symmetric key. The random symmetric key is then used to encrypt the text.
- Use the RSA-OAEP-256 algorithm, encrypting the random symmetric key.
- Use the A256GCM algorithm for the encryption of text with an Initialization Vector (IV). Size of IV should be 96 bits.
- Authentication Tag will be generated as an additional output of the A256GCM encryption.
- Size of this field is 128 bits.
- JWE header is used to pass AAD (Additional Authentication Data) for the A256GCM
- operation.
- All string-to-byte and byte-to-string conversions are done with UTF-8 charset.

Ideally the opensource library will deal with all steps above where it is needed to pass only the input (key, plain data, IV) and the output will be base64URL string serialized and separated by ".".

This output must be sent to HST's SDK as it is.

HST will share through secure channel the digital certificate in PEM format which contains the RSA public key used to encrypt the random symmetric key. This certificate is unique per environment per issuer, one for sandbox and another for production.

Since the mobile environment it is more exposed to attackers, HST suggests the encryption should be performed in backend service and the encrypted data should be delivered ready to mobile app pass it HST's SDK.

The SDK will send the encrypted payload to HST's backend which will use an HSM (Hardware Secure Module) to process the encrypted data. HST stores the private key on its HSM which is unextractable.

HST's backend will generate the Apple push payload formatted following all the encryption standards required by Apple, Visa, Mastercard and Amex which will be sent to apple after the SDK receives from HST's Backend.

The plain text must be in the following format, considering camel case. The CVV2 is not requested for push provisioning flow.

Sample Test Data

The following data can be used to validate cryptographic operations in case the issuer is using the method executeProvisioningOfEncryptedCard.

Plain Text (ASCII)

```
{"PAN":"1234123412341234","cardholderName":"Jose","expirationDate":{"month":"05","year":"2024"}}
```

AES Key (Hex) – 32 bytes (64 caracters)

```
A8AA8DBF16EA510D943A7DB6CCCEAB8E20D3AEC1CB057C7186C842A529B775B6
```

IV (Hex)

```
3B097E347861737FCC4B6822
```

Plain Text Encrypted (HEX)

```
98F16B7048B82F395A968F11D8765F2C7581AB95AB6D95AC85DC302E70DE44BE6F811A9241A7A09766C
A7BACD079D9701C1C46692D18D44DD858D098D9C544C91CBA3115DCDDB41A3931416C0A270EAF7782
14724A7ED0E699E2B4409111B6E8
```

Base64URL (JWE Header)

```
eyJ0eXAiOiJKT1NFIiwiZW5jIjoiQTI1NkdDTSIsImlhdCI6IjE2MjUwNTc4OTYiLCJhbGciOiJSU0EtT0FFUC0yNTYiLCJ
raWQiOiIxMjM0NTYifQ
```

JWE Header (ASCII)

```
{"typ":"JOSE","enc":"A256GCM","iat":"1625057896","alg":"RSA-OAEP-256","kid":"123456"}
```

Base64URL (JWE Encrypted Key)

```
R5dtTyLsFavdKgEEQOnTSwca-m33oYProaQsCLlk3izBGrbmj7y4VA01_LUbJE2ny15mHdOs59Y-
kcwls0Iar0i_YkxHcIfG8kyuIYd7H_6BaAxD8lcirUkGqR8h40IXkc64iBc33j4ThyObbnxjMF3Wnnj9XDCPIRxJC2f7e
bW1dveKqs15AkP5i89vJFMRQAusumUsgc5Ov30dqcCN6vG7viQNC_WjWHjHRM3YO4NLqLIvrBfYGRpkiMkzh
WKSCUGrEg-gfN6i5m0DVVYKT4kJyAsZbRrc7GQMEWch1sqFTMCVFzgIris-6J7Sy-
YzD4LG4q600xuMoRCDF9SQYw
```

Base64URL (JWEInitialization Vector)

```
Owl-NHhhc3_MS2gi
```

Base64URL (JWE Ciphertext)

```
mPFrcEi4Lzlalo8R2HZfLHWBq5WrbZWshdwwLnDeRL5vgRqSQaegl2bKe6zQedlwHBxGaS0Y1E3YWNCY2cVEyR
y6MRXc3bQaOTFBbAonDq93ghRySn7Q5pnitECREbbo
```

Base64URL (JWE Authentication Tag)

DO8EEY4JRq6GUutbVrIClA

JWE format

Base64URL (UTF8 (JWE Header)) || '.' || Base64URL (JWE Encrypted Key) || '.' || Base64URL (JWE Initialization Vector) || '.' || Base64URL (JWE Ciphertext) || '.' || Base64URL (JWE Authentication Tag)

JWE

eyJ0eXAiOiJKT1NFIiwiZW5jIjoiQTI1NkdDTSIsImlhdCI6IjE2MjUwNTc4OTYiLCJhbGciOiJSU0EtT0FFUC0yNTYiLCJ
raWQiOiIxMjM0NTYifQ.R5dtTyLsFavdKgEEQOnTSwca-
m33oYProaQsCLlk3izBGrbmj7y4VA01_LUbJE2ny15mHdOs59Y-
kcwls0Iar0i_YkxHcIfG8kyuIYd7H_6BaAxD8lcirUkGqR8h40IXkc64iBc33j4ThyObbnxjMF3Wnnj9XDCPIRxJC2f7e
bW1dveKqs15AkP5i89vJFMRQAusumUsgc5Ov30dqcCN6vG7viQNC_WjWHjHRM3YO4NLqLIvrBfYGRpkiMkzh
WKSCUGrEg-gfN6i5m0DVVYKT4kJyAsZbRrc7GQMEWch1sqFTMCVFzgIris-6J7Sy-
YzD4LG4q600xuMoRCDF9SQYw.Owl-
NHhhc3_MS2gi.mPFrcEi4Lzlalo8R2HZfLHWBq5WrbZWshdwwLnDeRL5vgRqSQaegl2bKe6zQedlwHBxGaS0Y1E
3YWNCY2cVEyRy6MRXc3bQaOTFBbAonDq93ghRySn7Q5pnitECREbbo.DO8EEY4JRq6GUutbVrIClA

Digital Certificate

```
-----BEGIN CERTIFICATE-----
MIIEfjCCA2agAwIBAgIJAJ4WXKmHsQh2MA0GCSqGSIb3DQEBCwUAMIHTMQswCQYD
VQQGEwJVUzEQMA4GA1UECAwHRkxPUklEQTEOMAwGA1UEBwwFTUlBTUkxITAfBgNV
BAoMGEhTVCBDYXJkIFRlY2hub2xvZ3kgTFREQTEfMB0GA1UECwwWSFNUUGF5IElz
c3VlciBTZXJ2aWNlczExMC8GA1UEAwwoSXNzdWVyU2VydmVyIFNhbmRib3ggRW5j
cnlwdCBLZXkgR2VuIDEuMDErMCkGCSqGSIb3DQEJARYcdmljdG9yX25hc2NpbWVu
dG9AaHN0LmNvbS5icjAeFw0xODA4MjkyMDM5NDRaFw0yODA4MjYyMDM5NDRaMIHT
MQswCQYDVQQGEwJVUzEQMA4GA1UECAwHRkxPUklEQTEOMAwGA1UEBwwFTUlBTUkx
ITAfBgNVBAoMGEhTVCBDYXJkIFRlY2hub2xvZ3kgTFREQTEfMB0GA1UECwwWSFNU
UGF5IElzc3VlciBTZXJ2aWNlczExMC8GA1UEAwwoSXNzdWVyU2VydmVyIFNhbmRi
b3ggRW5jcnlwdCBLZXkgR2VuIDEuMDErMCkGCSqGSIb3DQEJARYcdmljdG9yX25h
c2NpbWVudG9AaHN0LmNvbS5icjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBANt9RR7DDDL+fDTlMa5TquvYN3zuUma7UbCJaSk9MVF/6ciJFhdXK3VPTCDl
YQH+GHXXLh9LfKqttBjwkMstDYi7wDso43BIIQOxGcZIEmbGaxgdWTj0eKsaCWid
K1IZhejWUvGPDoR1MLxG9caqQPajdVwlkm/dzTTAB4EvSI7VL+PrxliafGqomkvJ
2A1/Cnkl/NvLr1YLKcf2mVQzzI1Wd12PeTq9r3afRBOkGJbUOVHhvcQf7Kpg5gSH
YqL6tk918ibB+TYkTuVX0X+KcpwH61D2BJOTD2E1gt/hMhkZzwKOeQGTApHWXGrD
/ttsU0pAk40agwNVjtj93XCn5XsCAwEAAaNTMFEwHQYDVR0OBBYEFCv3gaHF9Z83
W4j1o/t9J2J1hRzKMB8GA1UdIwQYMBaAFCv3gaHF9Z83W4j1o/t9J2J1hRzKMA8G
A1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAFzIwWq8bHRvy6U4+qBB
tdQrGIuUvbsMf+4hUNkTFpnvsO+ojt5YLzCqMLcSUz5vZg2dTpV8KpoQvQNUMJFJ
KBPRyca5qKLp8Si350563gnyQtFdpmGOccD76fdRGVqeiY2qXRcMTCg+4c4j0Sn8
+cwf0d4XlHCG17968pPnZqPh5n/wQzYDpzbkJ9KhGGbve8c6hglHnoAjtNGeY7dE
Hm+RI5LJs5FOFeauexlW4anndq/hCvw4khZ79M6lo9eW7r9HgN3aL89QoCnt8uDd
BXo42/eNJgKN5np2sYxilAvCXfn3SmUxFB/QuBZmhhGRbfQFUHVSyBRgq8OfD8Y9
cL8=
-----END CERTIFICATE-----
```

Private Key

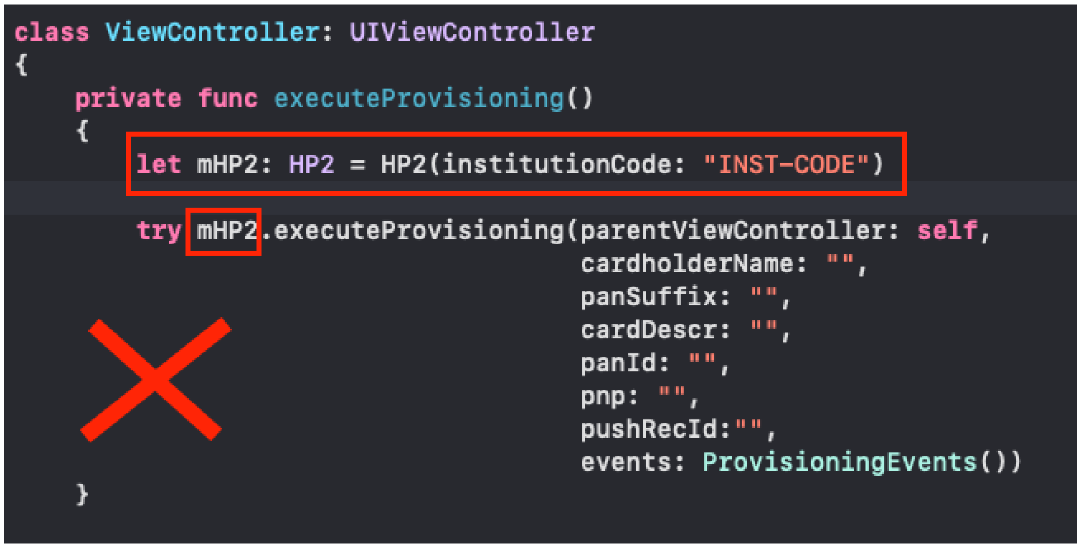
```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA231FHsMMMv58NOUxrlOq69g3fO5SZrtRsIlpKT0xUX/pyIkW
F1crdU9MIOVhAf4YddcuH0t8qq20GPCQyy0NiLvAOyjjcEghA7EZxkgSZsZrGB1Z
OPR4qxoJaJ0rUhmF6NZS8Y8OhHUwvEb1xqpA9qN1XCWSb93NNMAHgS9IjtUv4+vG
WJp8aqiaS8nYDX8KeSX828uvVgspx/aZVDPMjVZ3XY95Or2vdp9EE6QYltQ5UeG9
xB/sqmDmBIdiovq2T3XyJsH5NiRO5VfRf4pynAfrUPYEk5MPYTWC3+EyGRnPAo55
AZMCkdZcasP+22xTSkCTjRqDA1WO2P3dcKflewIDAQABAoIBAHsnGrVb82hZag+z
2eBaibizJM8wWYPS7s8DCsJc12NHRkGCyaZm/rdfuvNqQLgBfnAAQJzGmjiaJkco
E9YsfT+PORj4pETH895CbJfYsJqCEm0BaqKOaXZ3sXfFWB1EvOIb/4YSIq8noiTC
G/1QYgsBRERjwHqMFSvX414Uzb11zoJVc8tLgutwdbVzBPtqiX5mPWEGotuacEHt
JnTxhIWWjcRsLRdBna5tjjZSNJoanZqnwNg2zwuhPtd+je92h7pcVocmcO7Lq1FS
8TZQ9Di49E6pnbaRMhpMUbd4raSgklMrl4U/+avKwaD+4KoEFVH5vF5oR40kwkxI
eus+MKECgYEA+Xgt48cxD1wne5tM5f8A5ngo/5o3X4fVY2fDcQWTf1rehGy08onP
oPuhb8C1mnzZtM0wN1u7+7wflKeR/s8tx5LY41km6ypTy7eeBvdIJ4egAzx4ssJM
VY9peGhM/4cSpPdnhiSy92jPgnUw1r/OxQawRRLPD/iZiG1pAIq5jp0CgYEA4Twt
ydWdWIM+lbIDvL1uwzpiIdMjDsQ7UUaDzOz3SUbjFPoyX65ZX5ln824ir5V8mCRN
+XkSxX/zN4QEwG7vN9uGBZ3655SYm7SEx7X/OCjkrttHIE3C3UXKu2Heg5/vqx4T
L6zl1+Uh01VWjKh9V06R58alvpTpFfrH21c0vPcCgYBZa2Vvht+j9NFGMhVvP2dQ
NPPlIp8EhAjN18yrP2duN+EYsGpvoUwmFOv5xaaSmHvkncPRo/UOt5DOVP40yohL
R+ysGTGlC4f7tnZTYuGfIbMOVeehk5mO9ZfFjAVFdmINdYzK/W6U1iHDTkRSRXBV
GR+nsqU7wU0vJKCFjDQU1QKBgQCYVL8TFqpIRtt5GuB0MhpLZ50fC1FWl1kW6v4R
BuNoZ80FTwHqvFwtz+8CMKa84flWXJdv9na/pH22Ok+MZnrb6FiITAR5w4JDLJ/x
AiIOtXmU2TENpAn65Uzr1pFLrEvIC5smT2VT0uBBsiHLF9lNxxHfE/bdIpFSED5D
FoPQmQKBgGJF1geB/HWPiBm7+oMTsIMeiWxvL94MO5ExoTScrT0j3ehM8P4GIyRD
MbRIDr04JzcHB/6vzU+PJaQIow9VihIjOFjPKZZX/prwEwHXusCtKVhAFeKXXXz5
SThzNrrUTzTQlSElspPpZVPJK71ioobN+0Pr8+hzv7hgdLQL3Io4
-----END RSA PRIVATE KEY-----
```

8. FAQ – Frequently Asked Questions

Below it is presented some frequently asked questions and troubleshooting issues previously reported during SDK integration.

1. **Why does the screen control in iOS not return to the application, even if the user clicks on the Cancel Option?**

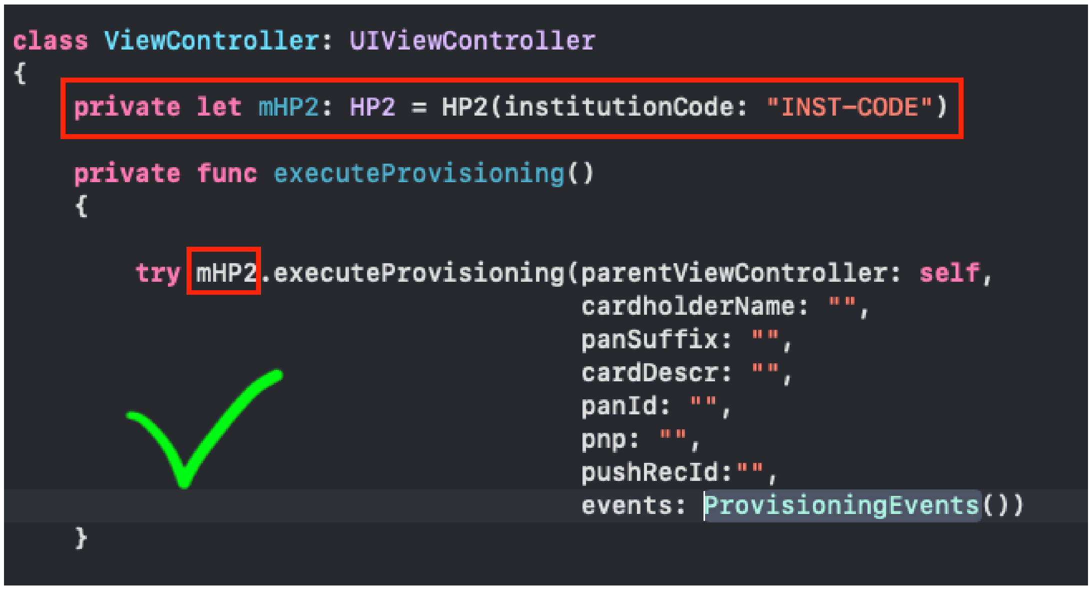
   A possible solution is to convert the variable (instance of HP2 SDK) in a class variable as showed in the examples as follows:



```
class ViewController: UIViewController
{
    private func executeProvisioning()
    {
        let mHP2: HP2 = HP2(institutionCode: "INST-CODE")

        try mHP2.executeProvisioning(parentViewController: self,
                                     cardholderName: "",
                                     panSuffix: "",
                                     cardDescr: "",
                                     panId: "",
                                     pnp: "",
                                     pushRecId:"",
                                     events: ProvisioningEvents())
    }
```

Figure 5 - Not recommended instantiation of the variable



```
class ViewController: UIViewController
{
    private let mHP2: HP2 = HP2(institutionCode: "INST-CODE")

    private func executeProvisioning()
    {

        try mHP2.executeProvisioning(parentViewController: self,
                                     cardholderName: "",
                                     panSuffix: "",
                                     cardDescr: "",
                                     panId: "",
                                     pnp: "",
                                     pushRecId:"",
                                     events: ProvisioningEvents())
    }
```

Figure 6 - Recommended instantiation as a class variable

2. **A card has already added in a device, but it is incorrectly listed as available for digitization yet. How to prevent this behavior?**

   To avoid this inconsistency, it is necessary to indicate the ***panID*** input parameter when invoking the method ***executeProvisioning*** or ***executeProvisioningOfEncryptedCard***. Doing this, Apple will apply a filter removing from the provisioning options the device where the card is already digitized. For each card already provisioned, ***panID*** value is returned using ***getCards*** method.
   See the sequence flow diagram available in section 4.1.3 - #Adding a card in another device for further details.

3. **How to know if there is an Apple Watch paired with the iPhone?**

   There are 3 possible scenarios:

   - **Card has not been provisioned yet**

     It must be displayed in the Issuer App the provisioning option as 'Add to Apple Wallet" – it is not necessary to check the pairing with Apple Watch, since iOS will automatically display both devices for card provisioning if there is a Watch paired with the user's iPhone. To know if a card has not been provisioned yet, call ***getCards*** method and check the list of cards returned. If there is no reference to the card, it has not been provisioned yet, neither on the iPhone nor Apple Watch. See the diagram available in section 4.1.2 - #Adding a new card in Apple Pay for more details.

   - **Card has already been provisioned in a device**

     If it is still possible to provision a card in a device, either iPhone or Apple Watch, the Issuer App must display the provisioning option as "Add to Apple Wallet" or "Add to Apple Watch".

     To verify if the card can be provisioned in a device, ***getCards*** method must be invoked. If the card is returned in the card list, ***isRemote*** parameter indicates where the card is provisioned – *false* for iPhone or *true* for Apple Watch.

     Then, using the ***panID*** information (returned in ***getCards***) as an input parameter, call ***isAvailableForCard*** method. If this method returns *true*, it indicates that there is still a device available for the card provisioning.

     In this step, it is possible to know if the remaining device for card provisioning is an iPhone (if the parameter ***isRemote*** returned false), or if it is an Apple Watch (if ***isRemote*** returned true)

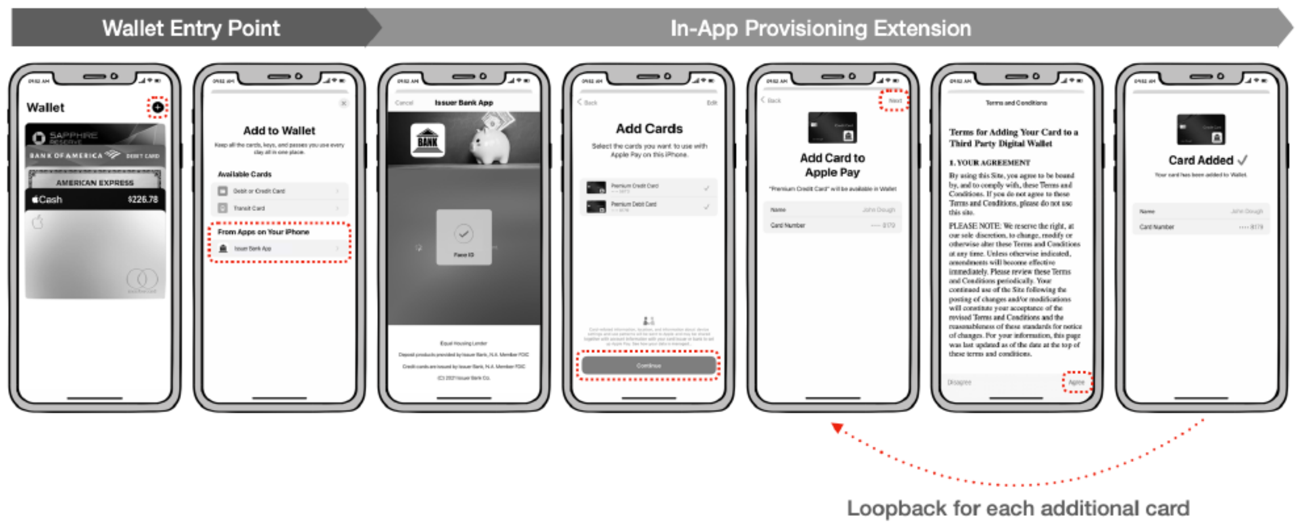
Furthermore, whenever that there is identified that a card is provisioned in a device using ***getCards*** method, is necessary to indicate the ***panID*** information when invoking the method ***executeProvisioning*** or ***executeProvisioningOfEncryptedCard***. This way, Apple will remove from the provisioning options the device where the card is already digitized. Check section 4.1.3 – #Adding a card in another device for the detailed flow.

- **Card has been provisioned in all available devices**

  In this case it is necessary to display in the Issuer App "Added to Apple Wallet". To verify if the card has been already provisioned, ***getCards*** method must be invoked. If the card is returned in the card list (containing the ***panID*** info), call ***isAvailableForCard*** method, informing the ***panID*** as an input parameter. If this method returns false, it indicates that there is no device available/enabled for the provisioning of this card. Check section 4.1.4 – #Card already provisioned in all available devices for the detailed reference to the sequence flow.

9. Appendix A: In-App Provisioning Extension

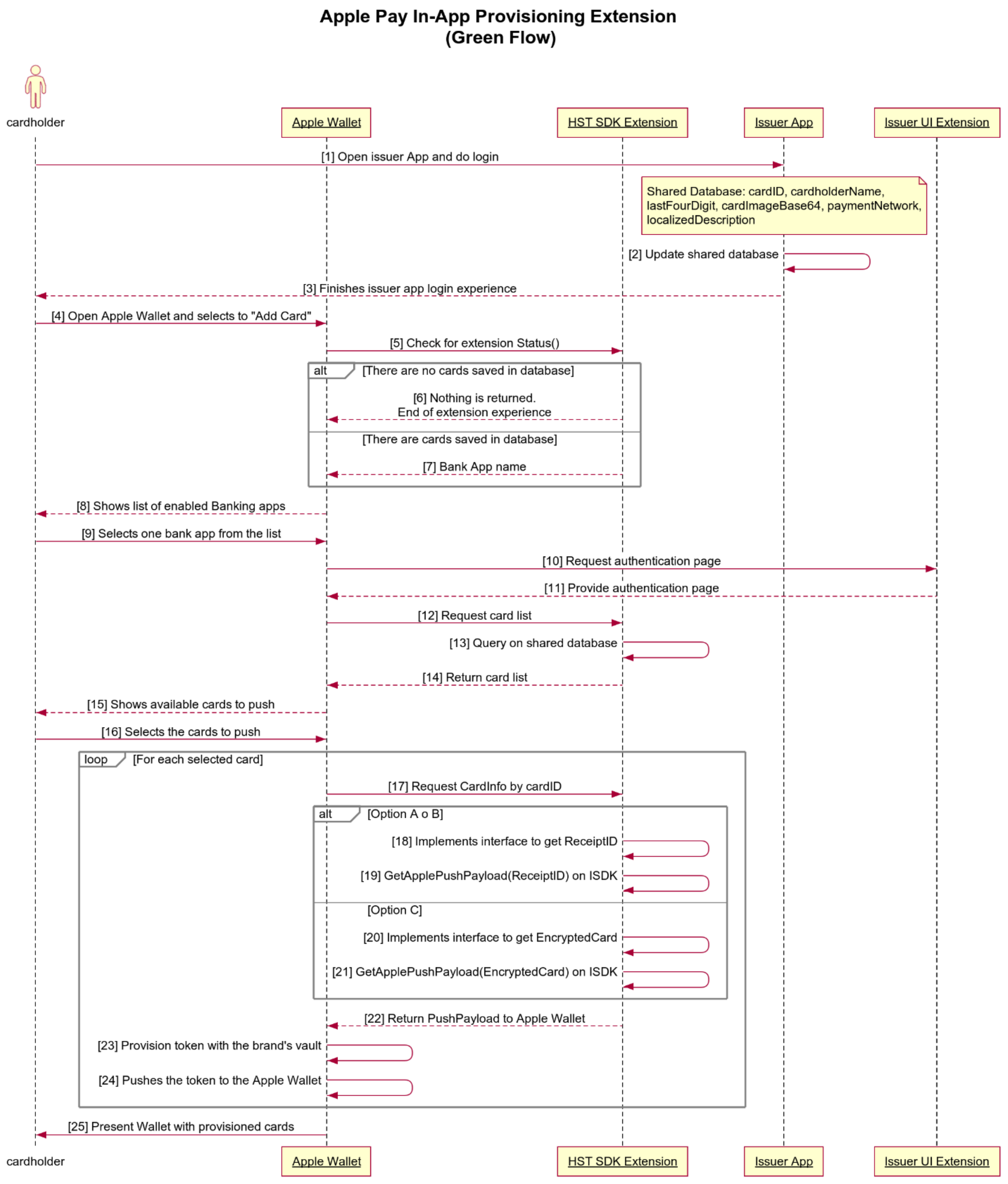
The Apple Pay In-App Provisioning Extension is an optional feature which enables the Apple Pay to pull the card info from the Issuer App. This section shows how to enable and configure this extension in Issuer SDK. Before explaining the technical details to enable and configure this optional feature we make a short introduction of the fundamental aspects of this use case.



The image was extracted from Apple's documentation and illustrates the whole process supported by this extension. Once with the Apple Wallet opened, the cardholder can "add a new card" into it, when in the presence of an enabled extension in issuer App the cardholder will see the bank mobile application listed an available source to push his card.

Given the cardholder have selected his banking app, it is going to provide a custom login page to authenticate the user, then a list of the eligible cards to be pushed into Apple Wallet, by green flow. The cardholder may select one or multiple cards for the push and after accepting each card terms and conditions, individually, every card will be pushed to the wallet, in the state of ready to use.

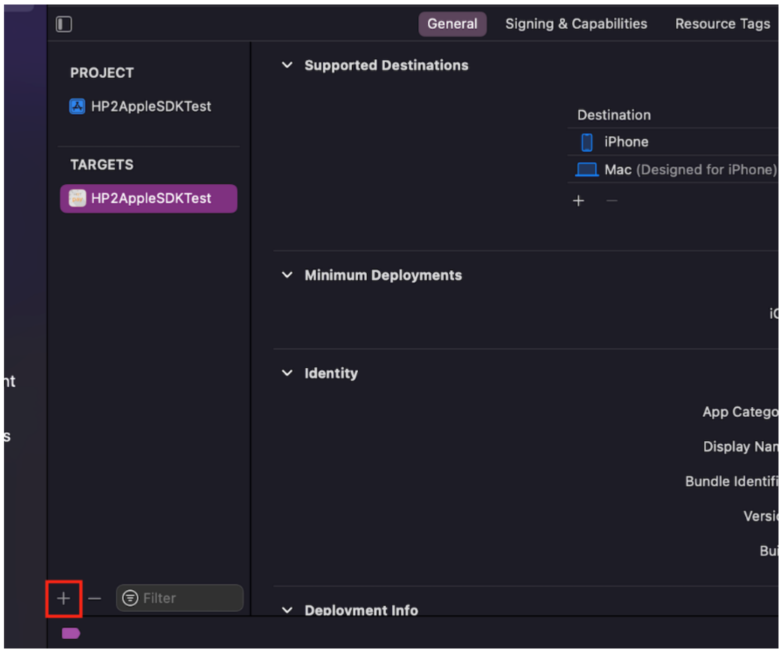
The following diagram represents step-by-step in a sequence diagram flow what is happening in backstage, highlighting the synchronism between each application. For instance, from the diagram we note that there are two situations when a login is required, both under responsibility of the issuer application, but the Apple Wallet will require from the Issuer App an additional UI extension.


Apple Pay In-App Provisioning Extension
(Green Flow)
cardholder
Apple Wallet
HST SDK Extension
Issuer App
Issuer UI Extension
[1] Open issuer App and do login
Shared Database: cardID, cardholderName, lastFourDigit, cardImageBase64, paymentNetwork, localizedDescription
[2] Update shared database
[3] Finishes issuer app login experience
[4] Open Apple Wallet and selects to "Add Card"
[5] Check for extension Status()
alt
[There are no cards saved in database]
[6] Nothing is returned. End of extension experience
[There are cards saved in database]
[7] Bank App name
[8] Shows list of enabled Banking apps
[9] Selects one bank app from the list
[10] Request authentication page
[11] Provide authentication page
[12] Request card list
[13] Query on shared database
[14] Return card list
[15] Shows available cards to push
[16] Selects the cards to push
loop
[For each selected card]
[17] Request CardInfo by cardID
alt
[Option A o B]
[18] Implements interface to get ReceiptID
[19] GetApplePushPayload(ReceiptID) on ISDK
[Option C]
[20] Implements interface to get EncryptedCard
[21] GetApplePushPayload(EncryptedCard) on ISDK
[22] Return PushPayload to Apple Wallet
[23] Provision token with the brand's vault
[24] Pushes the token to the Apple Wallet
[25] Present Wallet with provisioned cards
cardholder
Apple Wallet
HST SDK Extension
Issuer App
Issuer UI Extension

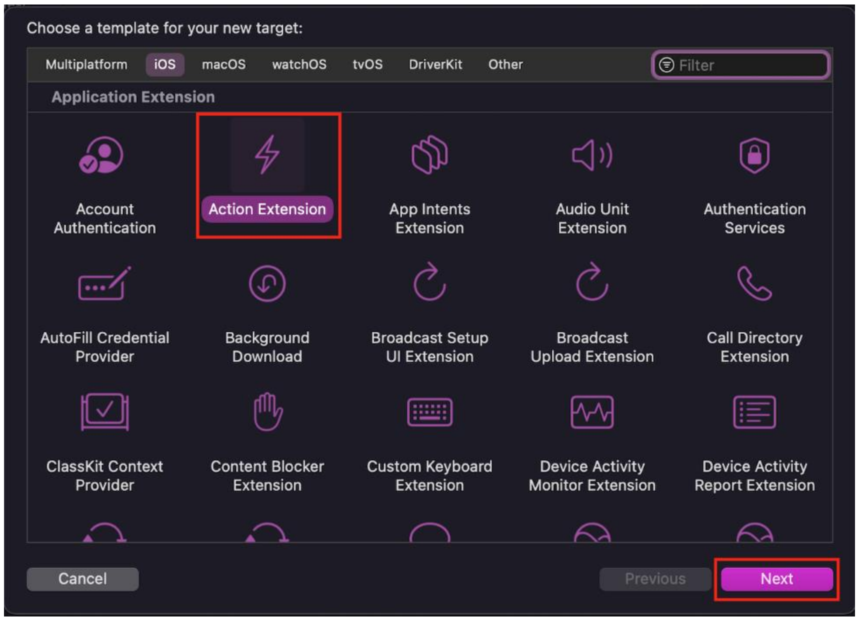
Important: All the process occurs in the Apple Wallet, the *extension* present in the Issuer App is responsible to provide the custom login page, and the available card list based on the cardholder's authentication. In addition, the extension is also responsible for retrieving the *CardInfo* given the *CardID* on the issuer's backend, so this step requires issuer code to be inserted on the extension.

9.1. Adding an extension into the project

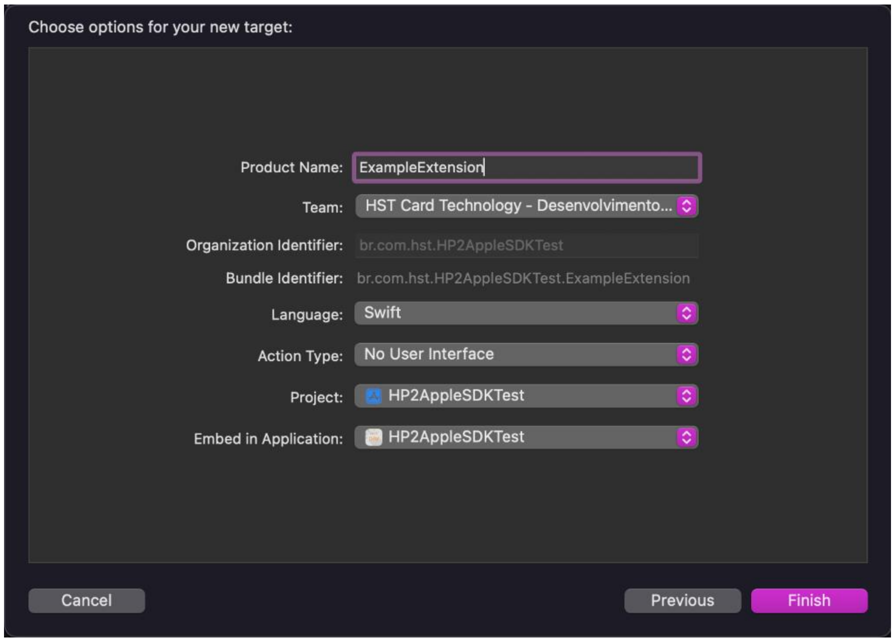
In the Targets section of the project, at the bottom right, select the "+" symbol.



Then select the Action Extensions template and proceed to the next step:


Choose a template for your new target:
Multiplatform
iOS
macOS
watchOS
tvOS
DriverKit
Other
Filter
Application Extension
Account Authentication
Action Extension
App Intents Extension
Audio Unit Extension
Authentication Services
AutoFill Credential Provider
Background Download
Broadcast Setup UI Extension
Broadcast Upload Extension
Call Directory Extension
ClassKit Context Provider
Content Blocker Extension
Custom Keyboard Extension
Device Activity Monitor Extension
Device Activity Report Extension
Cancel
Previous
Next

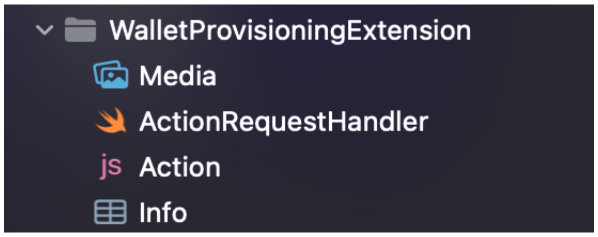
On the next screen, fill in a desired name for the extension in the "Product Name" field, then if necessary, modify the value in the "Action Type" field and select the "No User Interface" option.



Choose options for your new target:

Product Name: ExampleExtension
Team: HST Card Technology - Desenvolvimento...
Organization Identifier: br.com.hst.HP2AppleSDKTest
Bundle Identifier: br.com.hst.HP2AppleSDKTest.ExampleExtension
Language: Swift
Action Type: No User Interface
Project: HP2AppleSDKTest
Embed in Application: HP2AppleSDKTest

Once the extension is created, the extension folder with the following structure will be created in the project:



```
WalletProvisioningExtension
    Media
    ActionRequestHandler
    Action
    Info
```

9.2. Configuring an extension into the project

The *ActionRequestHandler.swift* file is the one that should be replaced by the file provided by HST, containing the implementation of the necessary methods of the extension.

The next configuration to be performed according to Apple's documentation is in the **Info.plist** file generated in the extension folder. It must contain the following two keys:

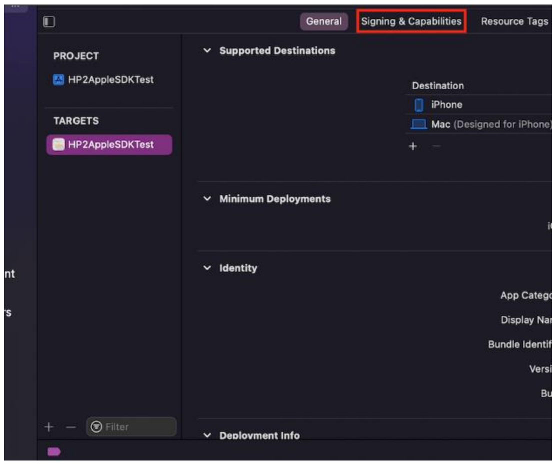
`NSExtensionPointIdentifier`	`String`	`com.apple.PassKit.issuer-provisioning`
`NSExtensionPrincipalClass`	`String`	`ExtensionHandler`

In the generated *Info.plist*, it is noted that there is already a key in the *NSExtension* dictionary with the name of *NSExtensionPrincipalClass* with the value

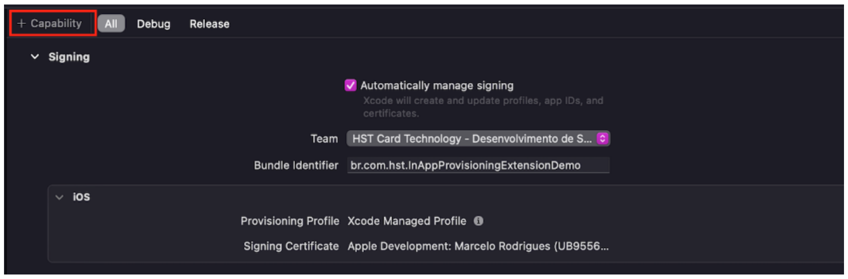
"$(PRODUCT_MODULE_NAME).ActionRequestHandler". This value should always point to the main class of the extension, in this case the ActionRequestHandler class.swift.

In the same dictionary, the NSExtensionPointIdentifier key must have the value of "com.apple.PassKit.issuer-provisioning". The file has this key generated automatically, being necessary only to change the value of it as indicated in the documentation.

Also in the Targets section, select your application and go to the "Signing & Capabilities" section:



In this section, select the "+ Capability" option in the upper left corner:

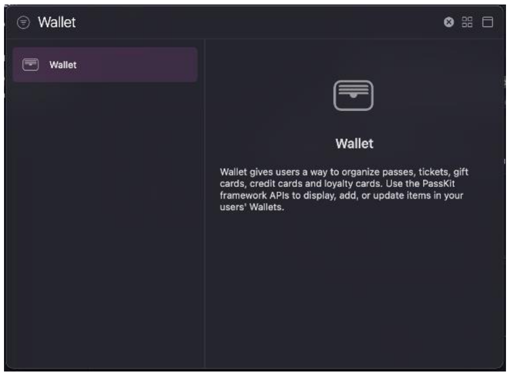


Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: HST Card Technology - Desenvolvimento de S...
Bundle Identifier: br.com.hst.InAppProvisioningExtensionDemo

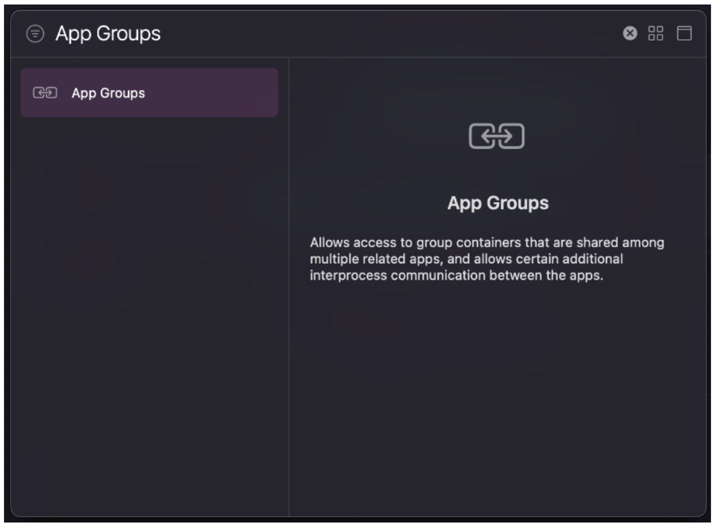
Provisioning Profile: Xcode Managed Profile
Signing Certificate: Apple Development: Marcelo Rodrigues (UB9556...

In the search bar displayed, search for Wallet and add to your project by pressing enter after selecting the searched option "Wallet":



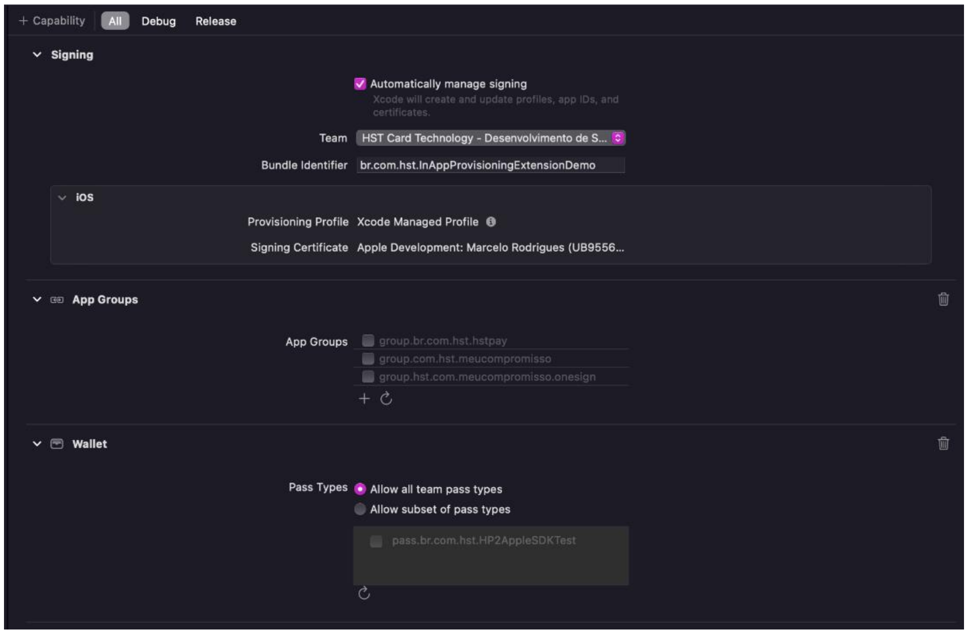
Wallet gives users a way to organize passes, tickets, gift cards, credit cards and loyalty cards. Use the PassKit framework APIs to display, add, or update items in your users' Wallets.

Select the "+ Capability" button again and in the same search bar type "App Groups", pressing Enter to add to your Target after selecting the searched option "App Groups":



Allows access to group containers that are shared among multiple related apps, and allows certain additional interprocess communication between the apps.

Your "Signing & Capabilities" section should look like this, with the two Capabilities added:



[✓] Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: HST Card Technology - Desenvolvimento de S...
Bundle Identifier: br.com.hst.InAppProvisioningExtensionDemo

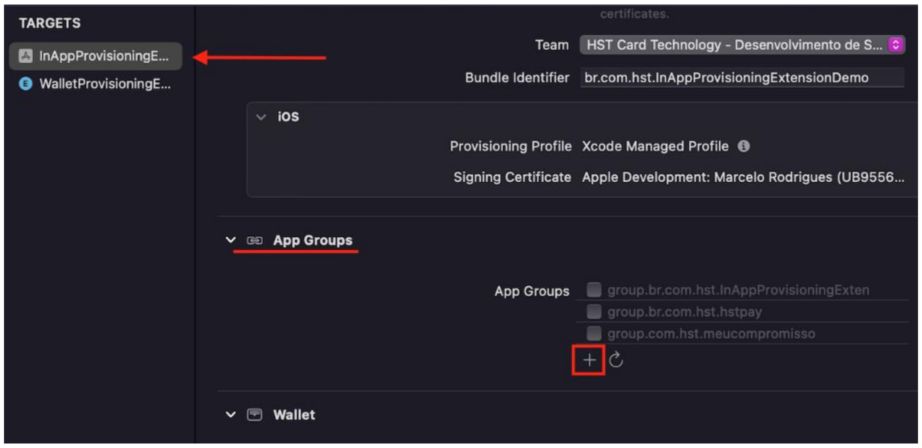
iOS
Provisioning Profile: Xcode Managed Profile
Signing Certificate: Apple Development: Marcelo Rodrigues (UB9556...

App Groups:
- [ ] group.br.com.hst.hstpay
- [ ] group.com.hst.meucompromisso
- [ ] group.hst.com.meucompromisso.onesign

Pass Types: [✓ Allow all team pass types] Allow subset of pass types
- [ ] pass.br.com.hst.HP2AppleSDKTest

You must add the "App Groups" Capability in the "Signing & Capabilities" section in the Target of the Extension you added earlier, as this Capability will be required for the implementation of the database shared between the Application and the Extension.

The next step is to set up a shared Group ID between the project's Targets (Application and Extension). Select the App in the Targets section and in the Capability section called "App Groups" select the "+" symbol:



TARGETS

InAppProvisioningE...

WalletProvisioningE...

certificates.

Team: HST Card Technology - Desenvolvimento de S...

Bundle Identifier: br.com.hst.InAppProvisioningExtensionDemo

iOS

Provisioning Profile: Xcode Managed Profile

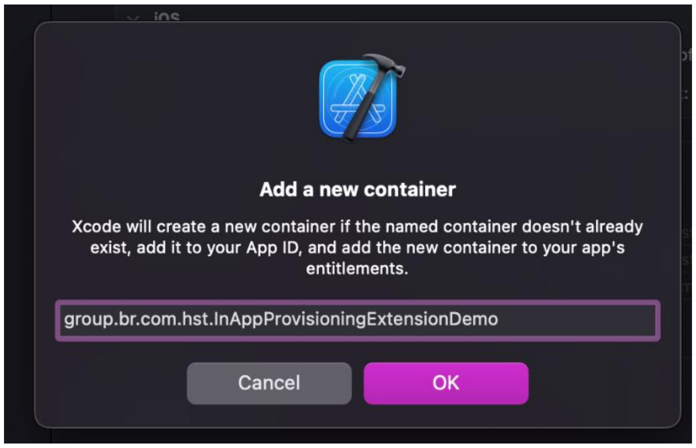
Signing Certificate: Apple Development: Marcelo Rodrigues (UB9556...

App Groups

App Groups:
- group.br.com.hst.InAppProvisioningExten
- group.br.com.hst.hstpay
- group.com.hst.meucompromisso

Wallet

On the next screen write the desired name for your Group ID. A suggestion is to concatenate "group" and the Bundle ID of your application:



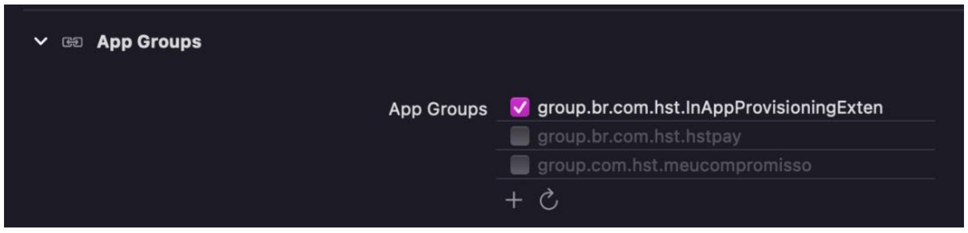
Add a new container

Xcode will create a new container if the named container doesn't already exist, add it to your App ID, and add the new container to your app's entitlements.

group.br.com.hst.InAppProvisioningExtensionDemo

Once created it should appear in your list of App Groups within the Capability section "App Groups". Keep it selected in your App and ensure that the same Group ID is selected in the same section of this Capability in the Target of your extension.

This will ensure that the two Targets will be able to share data and will also generate an .entitlements file in each Target's folder:



App Groups

App Groups:
- [✓] group.br.com.hst.InAppProvisioningExten
- group.br.com.hst.hstpay
- group.com.hst.meucompromisso

Additionally, the file **Model.xcdatamodeld**, provided by HST, must be added to Issuer App Xcode project.

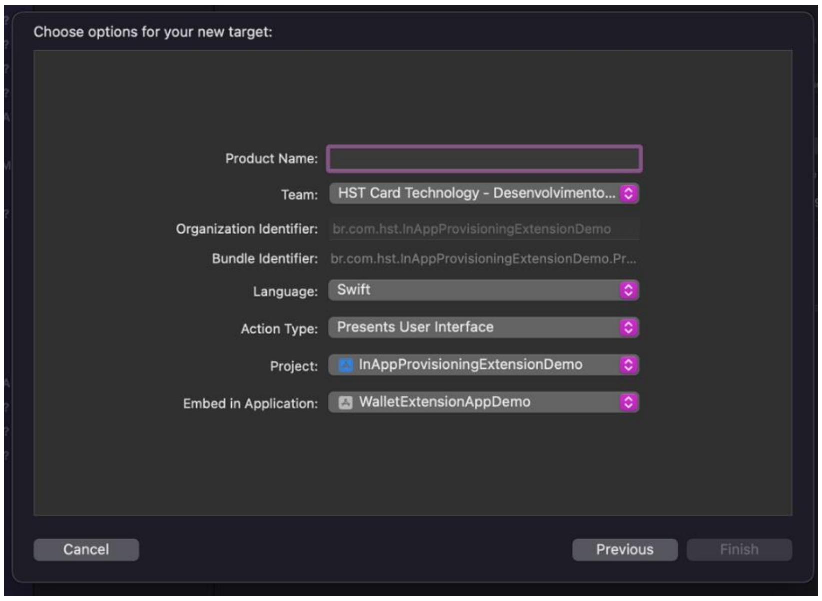
9.3. In-App Provisioning Authentication UI Extension

The Appendix A has introduced a guide to implement the In-App Provisioning Extension present in the Issuer SDK, describing in detail how to install, configure and enable this feature provided by HST. However, the In-App Provision Extension requires an independent authentication UI module that should be provided by the IssuerApp maintainer to permit the cardholder authentication. Please notice that this UI Extension is independent and doesn't have access to any resource from the ISDK neither the ISDK In-App Provisioning Extension, such as the shared database.

This section provides a guide based on a sample of the UI Extension built to supply the authentication and should be customized accordingly the issuer authentication rules. The goal here is to specify how the UI Extension should be crafted to process the authentication request and responses expected by the Apple Pay protocols, it should be able to indicate successful and failed authentications.

9.3.1. Adding an extension into the project

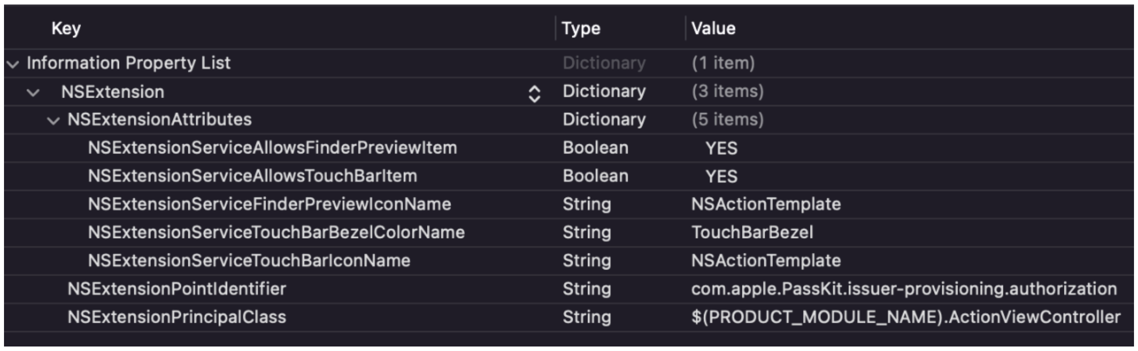
The UI Extension must be created and added to the same project which holds the main extension and the issuer app. It must be added as new Action Extension and should set the Action Type as "Presents User Interface".


Choose options for your new target:

Product Name:
Team: HST Card Technology - Desenvolvimento...
Organization Identifier: br.com.hst.InAppProvisioningExtensionDemo
Bundle Identifier: br.com.hst.InAppProvisioningExtensionDemo.Pr...
Language: Swift
Action Type: Presents User Interface
Project: InAppProvisioningExtensionDemo
Embed in Application: WalletExtensionAppDemo

Please be aware, these steps will add an extension with similar structure than the In-App Provisioning extension, but with a View Controller instead of a Handler and a Storyboard.

The **Info.plist** must be configured in this way:



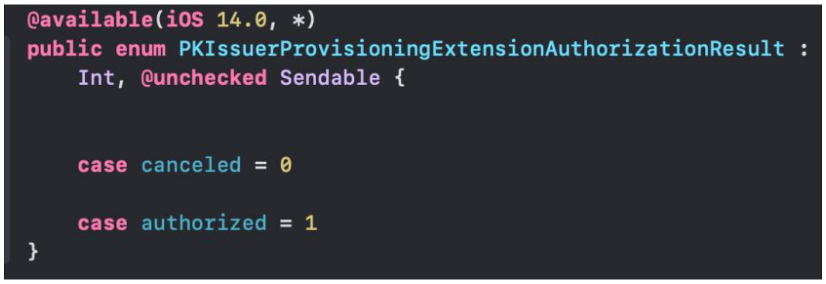
Key	Type	Value
Information Property List	Dictionary	(1 item)
NSExtension	Dictionary	(3 items)
NSExtensionAttributes	Dictionary	(5 items)
NSExtensionServiceAllowsFinderPreviewItem	Boolean	YES
NSExtensionServiceAllowsTouchBarItem	Boolean	YES
NSExtensionServiceFinderPreviewIconName	String	NSActionTemplate
NSExtensionServiceTouchBarBezelColorName	String	TouchBarBezel
NSExtensionServiceTouchBarIconName	String	NSActionTemplate
NSExtensionPointIdentifier	String	com.apple.PassKit.issuer-provisioning.authorization
NSExtensionPrincipalClass	String	$(PRODUCT_MODULE_NAME).ActionViewController

The key configurations are:

- **NSExtensionPointIdentifier**: it must be configured with “com.apple.PassKit.issuer-provisioning.authorization”
- **NSExtensionPrincipalClass**: this key must have the same name of the main View Controlle, in this sample it is called, ActionViewController
- **NSExtensionMainStoryBoard**: optional key, only present when the UI is presented through Storyboard, the value must be the same of the Storyboard filename.

The UI Extension must also have the file .entitlement with the key “com.apple.developer.payment-pass-provisioning” set with the value “YES (TRUE)”.

The main View Controller class must have heritance from “PK Issuer Provisioning Extension Authorization Providing” class, enabling access to the method “var completionHandler: ((PK Issuer Provisioning Extension Authorization Result) -> Void)?”. This variable is responsible to report the main In-App Provisioning Extension the user authentication result (success or failure), it should be requested after the authentication process.



```
@available(iOS 14.0, *)
public enum PKIssuerProvisioningExtensionAuthorizationResult :
    Int, @unchecked Sendable {


    case canceled = 0

    case authorized = 1
}
```

At this same class, it is required the following method implementations:

- func beginAuthorization(completion: @escaping(PKPaymentAuthorizationResult) -> Void): Is the method where to implement the authentication business logic.
- func isAuthorizationValid(forAuthorization authorization: PKPaymentAuthorizationController) -> Bool: is the method to report the authentication result.

9.4. In-App Provisioning Extension Interfaces

Besides the main extension class that should be replaced by the class provided by HST, the extension project should also have a class that will handle the methods of the interface.

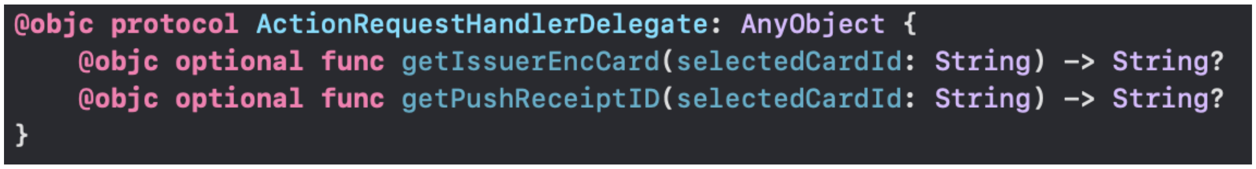
These are the methods of the interface:

@objc protocol ActionRequestHandlerDelegate: AnyObject {

  @objc optional func getIssuerEncCard(selectedCardId: String) -> String?

  @objc optional func getPushReceiptID(selectedCardId: String) -> String?

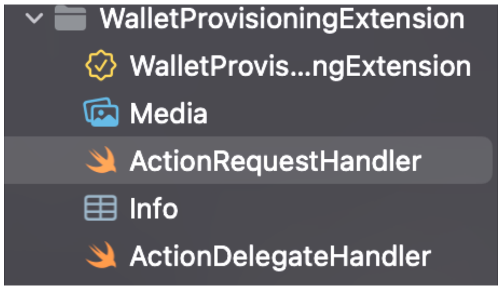
}



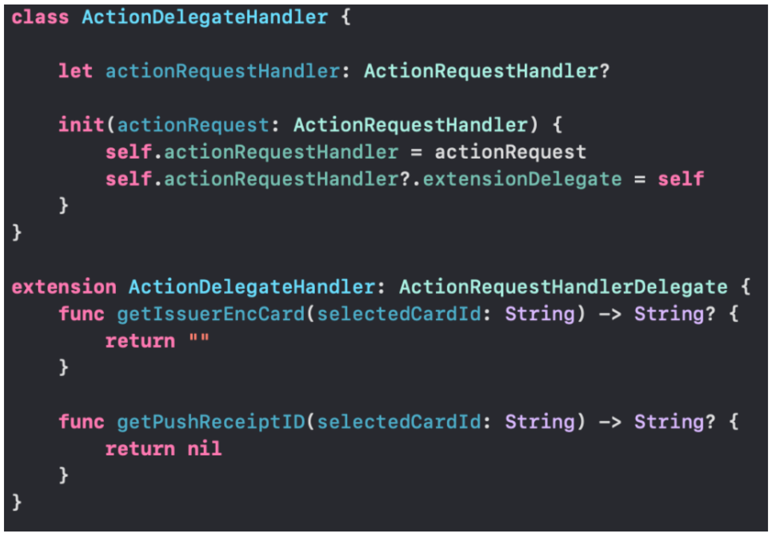
```
@objc protocol ActionRequestHandlerDelegate: AnyObject {
    @objc optional func getIssuerEncCard(selectedCardId: String) -> String?
    @objc optional func getPushReceiptID(selectedCardId: String) -> String?
}
```

Therefore, the extension project should have the reference of a class that will implement this interface and will return the data to the extension forward it to the SDK. The extension main class already calls this class and do the connection when the data returned are necessary for the extension operation.

The class in question is "*ActionDelegateHandler*":



This class has the following structure:



```swift
class ActionDelegateHandler {

    let actionRequestHandler: ActionRequestHandler?

    init(actionRequest: ActionRequestHandler) {
        self.actionRequestHandler = actionRequest
        self.actionRequestHandler?.extensionDelegate = self
    }
}

extension ActionDelegateHandler: ActionRequestHandlerDelegate {
    func getIssuerEncCard(selectedCardId: String) -> String? {
        return ""
    }

    func getPushReceiptID(selectedCardId: String) -> String? {
        return nil
    }
}
```

In the methods *getIssuerEncCard* or *getPushReceiptID*, the Issuer should perform the necessary implementation when receives the card ID selected by the extension, returning the Encrypted Card data or the Push Receipt ID.

<u>Notes</u>:

- *getIssuerEncCard* is used when the Issuer implements the **Option C for integration**, whereas *getPushReceiptID* is used when **Option A or B** is chosen.
- An example of this class will be provided to the Issuer along with Issuer SDK (HP2).

10. Revision History

Date	Version	Description	Author
02/28/2020	1.0 (Alpha)	Document creation	Victor Nascimento, Alexandre Rosa, João Rodrigues, José A. Ramos
06/01/2020	1.1	Document revision – classes and enumerations adjusted	Alexandre Rosa, Bruno Vianna, José A. Ramos
06/25/2020	1.2	Document revision - Apple Integration Requirements instructions added (*section 3.3*) - Adjust Initialize method return from "true or false" to "none"	Alexandre Rosa, Bruno Vianna, José A. Ramos
09/04/2020	1.3	Document revision - Update diagrams - Option C included - Revision information - Methods and classes adjusted	Eduardo Cunha, Alexandre Rosa
06/30/2021	1.4	Updated Section 6 Entitlements Added section 7 – Cryptography information. Removed Merchant ID configuration Updated HP2Errors and Card type	Ivan Ortiz, Victor Nascimento
12/22/2021	1.5	- Changed the versions of Apple Documents references in the Introduction section; - Added a new section containing use cases and sequence flow diagrams; - The order of CardType Enum elements was changed (SECURE_ELEMENT and ANY); - HP2Errors Class – Added new error codes from 1 to 7; - HP2ErrorDetails Class – Added new specific error codes from -1 to 7; - Added new section for Frequently Asked Questions.	José A. Ramos
10/14/2022	1.6	- Added three new subsections in section 6 – "iOS and Swift minimum versions", "How to	José A. Ramos

		get Adam ID", "How to get Team ID" and "Adding key to Issuer Application for card provisioning".	
01/19/2022	1.7	- Added a new institutionCode parameter in the Execute Provisioning and Execute Provisioning of Encrypted Card methods.	Rafael Gomes
05/04/2023	1.8	- Enhanced description in alternative option flows description. - Added optional In-App Provisioning Extension at Appendix A - section 9	Danilo S. Silva, Ivan A. Ortiz, José A. Ramos, Rossana R. Silva
07/13/2023	1.8.1	- Adjusted descriptions in Appendix A – In-App Provisioning Extension. - Added section 9.6.	Danilo S. Silva, José A. Ramos, Rossana R. Silva
12/11/2023	1.9	- Data preparation Option A and Option B is now deprecated to new implementations. The section 3.1.2.1, section 3.1.2.2 and section 5 must not be considered for new projects. - Added details on vault's manager platform section 6.9. For In-App Provisioning Extension: - Added new class CardDataModel – section 4.3.8. - Added new method updateDataBase – section 4.2.7. - Removed sections "Adding a shared database" and "Configuring a shared database" from Appendix A.	Danilo S. Silva, Ivan A. Ortiz, José A. Ramos