Apple Pay
Platform Integration Guide

September 2025

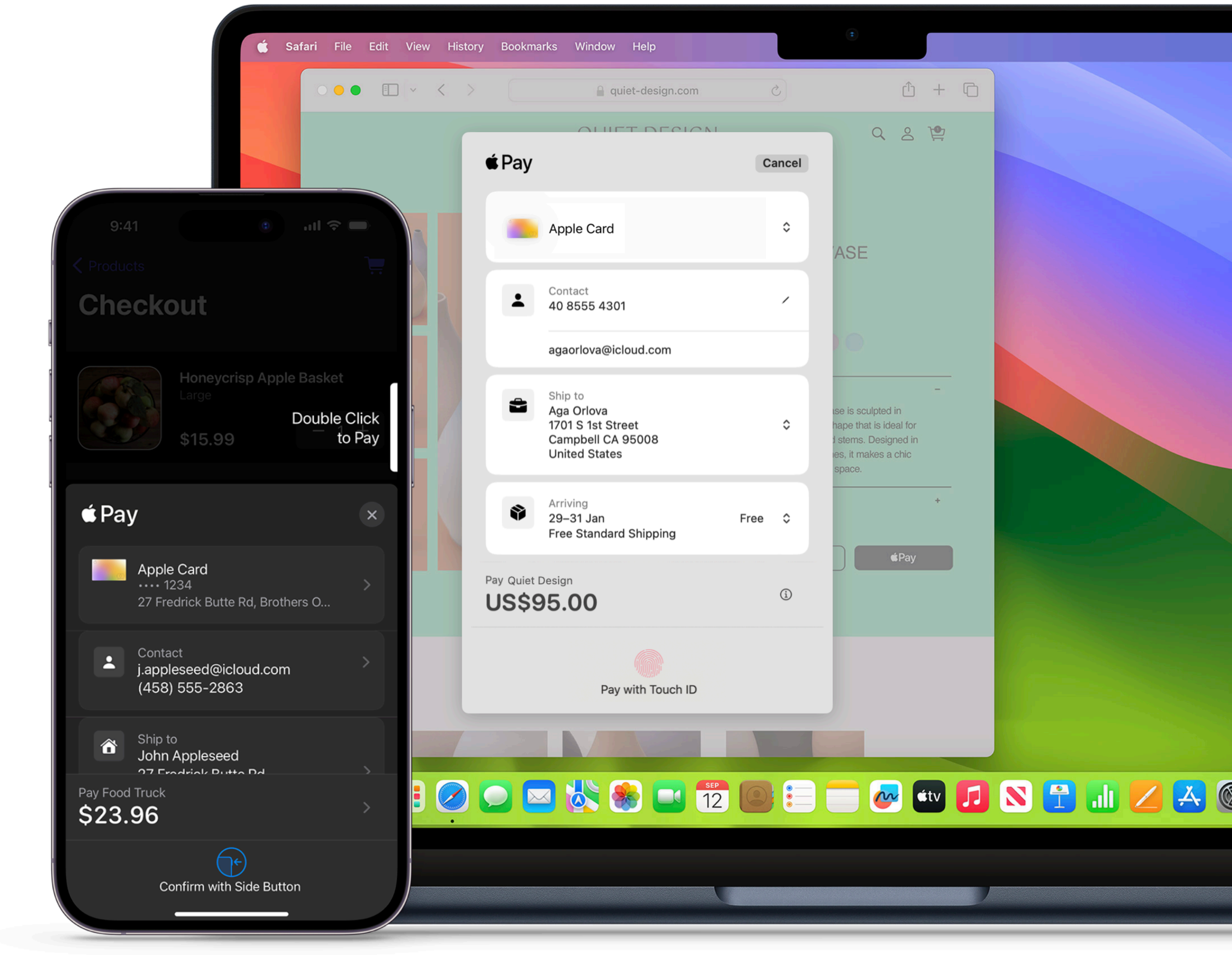
Contents

Introduction

Apple Pay provides an easy and secure way to make payments in iOS, iPadOS, and watchOS apps, and on the web when using compatible browsers. By using biometric authentication, users can quickly and securely provide their payment, shipping, and contact information to check out.

Customers love the simplicity of Apple Pay, and merchants love the increased conversion rates and new user adoption that comes with it. To experience an Apple Pay test transaction on a compatible device, visit the Apple Pay demo site at applepaydemo.apple.com.

This guide outlines the steps and considerations you need to take as a platform to enable Apple Pay payments for your merchants. If you are a merchant, please reference the Apple Pay Merchant Integration Guide.



Getting started

Before enabling Apple Pay, it is important that developers understand how Apple Pay differs from an In-App purchase, and make sure their implementation follows the guidelines. There are many ways to implement Apple Pay, with some of the most popular Payment Service Providers and E-Commerce platforms offering an Apple Pay SDK or JavaScript API as a quick and reliable way to support Apple Pay in an app or on a website.

Guidelines

Apple Pay vs In-App Purchases

Use Apple Pay in your app to sell physical goods like groceries, clothing, and appliances; for services such as club memberships, hotel reservations, and events tickets; and for donations. Use In-App Purchases to sell virtual goods such as premium content for your app, and subscriptions for digital content.

Apple Pay on the Web Acceptable Use Guidelines

Before merchants can deploy Apple Pay on their website, they must ensure that their implementation follows these guidelines. When onboarding merchants using the Web Merchant Registration API, you must also ensure these guidelines are being met by the merchant.

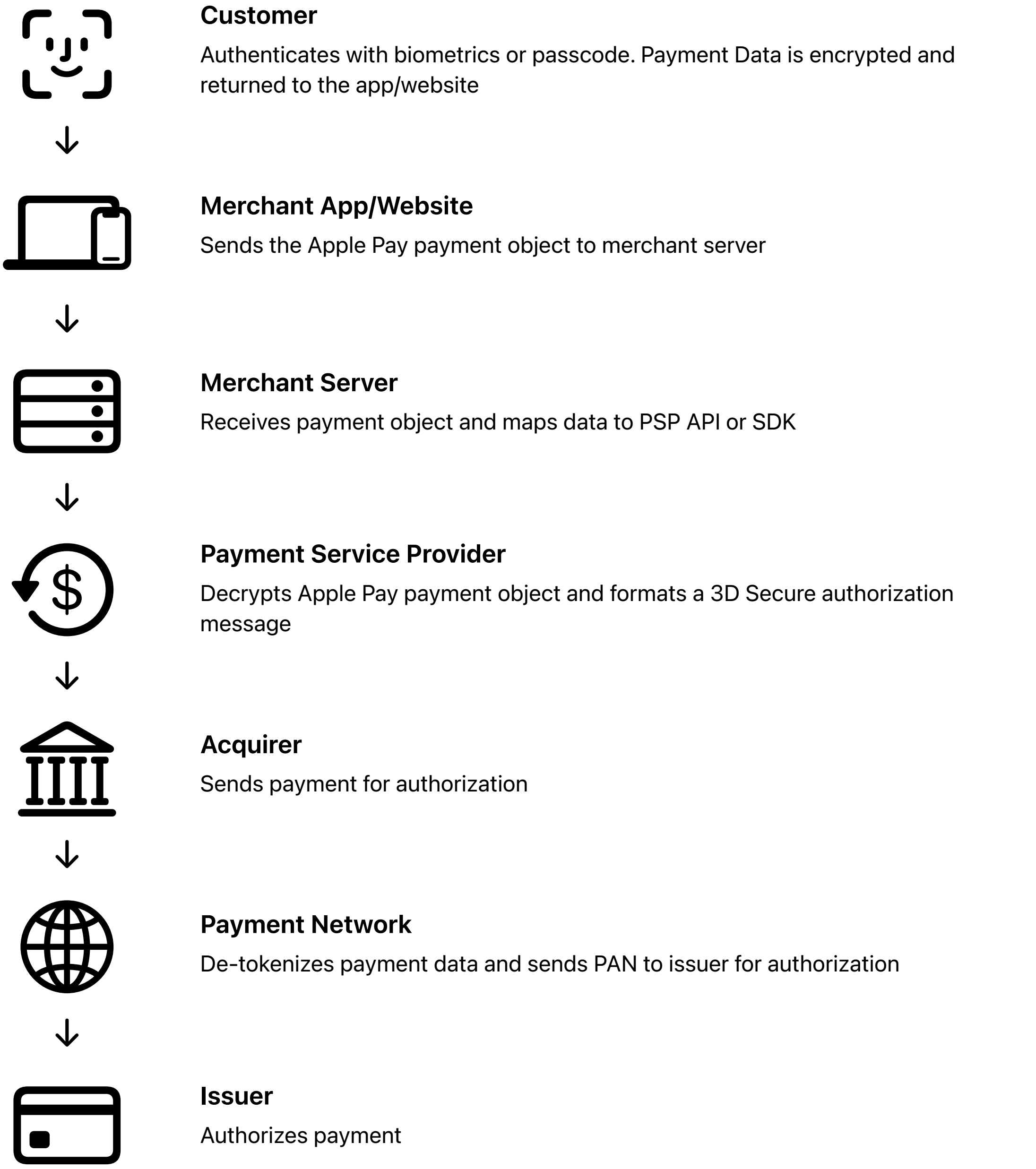
🔗 Apple Pay on the Web Acceptable Use Guidelines

Understanding Apple Pay

Apple Pay is available on all iOS devices with a Secure Element — an industry-standard, certified chip designed to store payment information safely. On Desktop, users must have an Apple Pay-capable iPhone authorize the payment, or a Mac with Touch ID. Any transaction type you currently support for regular debit and credit cards can be performed with Apple Pay, including refunds.

Payment flow

Apple Pay uses device-specific tokenized credit or debit card credentials (DPAN) in place of a Payment Account Number (PAN). When users authenticate the payment using their biometric data or passcode, the tokenized card data is returned to your app or website. This token can then be passed to your Payment Service Provider (PSP) to process as you would for a typical online credit or debit card payment.



Customer

Authenticates with biometrics or passcode. Payment Data is encrypted and returned to the app/website

Merchant App/Website

Sends the Apple Pay payment object to merchant server

Merchant Server

Receives payment object and maps data to PSP API or SDK

Payment Service Provider

Decrypts Apple Pay payment object and formats a 3D Secure authorization message

Acquirer

Sends payment for authorization

Payment Network

De-tokenizes payment data and sends PAN to issuer for authorization

Issuer

Authorizes payment

Get set up for Apple Pay

To enable Apple Pay on your app or website, you need to confirm that you have the correct options configured on your server, are set up to accept Apple Pay Payments and register for an Apple Developer account.

☑ Confirm you operate in a supported region

Check the list of currently supported regions at the link below.

🔗 Countries and regions that support Apple Pay

☑ Confirm you support EMVCo Token data elements

In order to accept Apple Pay, the party processing the payments must be capable of handling the appropriate fields related to processing EMVCo tokens such as TAVV, ECI or potentially 3D Secure related data elements such as CAVV/UCAF values.

☑ Set up your Server

For Apple Pay on web only, ensure that your server meets the set up requirements for secure communications with Apple Pay.

🔗 Setting up your server

☑ Register for an Apple developer account as an organization

Both Apple Pay in apps and Apple Pay on the web require a subscription to the Apple developer Program, which must be renewed yearly. You should enroll as an organization, and you can use the same Apple developer account that you use today to publish apps to the App Store.

🔗 Enroll for an Apple Developer account

Design your Apple Pay solution

Apple Pay creates a streamlined checkout process, allowing customers to authorize payments and complete transactions promptly. Consider where and when in the customers journey would be best to utilize Apple Pay to help drive conversion and enhance the customer experience.

☑ Assess the checkout products you offer

To help drive conversion and enhance the user's experience, carefully consider the location of the Apple Pay button. The best user experiences place the Apple Pay button as early in the checkout process as possible in order to leverage Apple Pay provided information, and minimize data entry.

☑ Apply the Apple Pay design principles

There are several key Apple Pay design principles that help drive conversion, increase usage and engagement, and provide an excellent user experience. Please review our best practice recommendations in Planning for Apple Pay.

🔗 Planning for Apple Pay

☑ Review the Human Interface Guidelines

Refer to the Apple Pay Human Interface Guidelines for additional information on how to best incorporate Apple Pay in your app or website.

🔗 Apple Pay Human Interface Guidelines

Merchant Onboarding

Before merchants are able to support Apple Pay, they must first have a set of credentials registered that identifies them as a merchant able to accept Apple Pay payments. These credentials can be configured by the merchant within their own Apple Developer Account, or as a platform integrator, you can manage the Apple Pay configuration on behalf of the merchant for web transactions.

Onboarding with a merchant-managed Apple Developer Account

To allow your merchants to offer Apple Pay within their Apps you must support this method of onboarding. This option allows merchants to create and manage their own set of Apple Pay credentials which can be used across both App and Web.

- ✓ Supports native in-app Apple Pay integrations
- ✓ Merchant has more control over their certificates and keys
- ⊗ Merchant is required to have their own Apple Developer Account
- ⊗ Does not support Hosted Payment Pages.

To remove complexity, merchants often let payment processors handle decryption of the Apple Pay token. When a processor performs the decryption process, a unique key pair will need to be generated for each merchant. The processor provides the public key via a Certificate Signing Request (CSR), which the merchant uploads to the Apple Developer Portal to generate a Payment Processing Certificate.

☑ Build your merchant onboarding interface for Merchant-managed Developer Account

To streamline the onboarding process for the merchant, you should provide them the ability to generate their key pair and CSR directly from within your merchant portal.

- ✓ Download a certificate signing request to generate their Payment Processing Certificate
- ✓ Upload the generated payment processing certificate for your records

☑ Key management

Whether the payment platform is managing the keys on behalf of their merchants, or the merchant is managing this via their own Apple Developer account, payment platforms should implement mechanisms to roll keys and certificates without any downtime. Payment processors should support at least 2 active key pairs/certificates on an Apple Pay configuration at a time so that it's possible to transition from one key pair to a new set.

The key rolling process should follow the below steps (assuming existing keys are in place):

1. Generate new key pair and CSR on the payment platform
2. Upload new CSR to Apple Developer account & download new certificate
3. Upload new certificate to the payment platform
4. Activate the new keys/certificates in the Apple Developer account
5. Revoke old keys/certificates

Payment Processors should use the value of the publicKeyHash to determine which merchant public key Apple used for encryption, and then retrieve the corresponding certificate and private key for decryption. This is especially important as keys/certificates are being transitioned.

Onboarding with the Apple Pay Web Merchant Registration API

The most streamlined and scalable way to onboard merchants on the web is by using the Apple Pay Web Merchant Registration API. This is a REST API that enables payment platforms to register and manage the Apple Pay configuration on behalf of the merchant. This offers some key benefits for both the platform and merchant, including:

- ✓ Merchant is not required to create an Apple Developer Account
- ✓ Shared set of keys and certificates across your Merchant Portfolio
- ✓ Merchants can register their own domains, hosted payment pages, or both
- ✓ Domain registration does not expire
- ✓ Supports integrations on hosted pages and direct API or JS libraries
- ☒ Cannot be used for native in-app integrations

☑ Apply for access to the Apple Pay Web Merchant Registration API

Platforms enrolled in the Apple developer program as an organization can apply to use the Apple Pay Web Merchant Registration API by submitting an application form.

Web Registration API Application form

☑ Configure your credentials

Once you have been granted access to use the API, you will be able to set up a Payment Platform Integrator ID and it's associated certificates which follows the same process as a Merchant Identifier as outlined in the Apple Pay Merchant integration Guide.

Apple Pay Merchant Integration GuideStart using the Web Merchant Registration API

☑ Prepare merchant domains

Before making a Register Merchant request, you must prepare each domain included in the request for verification. When you have created your Payment Platform Integrator ID, you will be able to access the domain verification file in the Apple Developer portal.

Download and distribute this file to each merchant whose domain will be registered via the Web Merchant Registration API. If merchants will be using Apple Pay via pages hosted on your own domain, host this file on your domain also. Host your domain verification file at the following path for each domain being registered:

https://[DOMAIN_NAME]/.well-known/apple-developer-merchantid-domain-association

☑ Register a Merchant

Retrieve information about a registered merchant's current state by using the merchant's internal merchant identifier.

registerMerchant

☑ Unregister a Merchant

Unregister one or more domains associated with a previously registered merchant.

unregisterMerchant

☑ Get Merchant Details

Retrieve information about a registered merchant's current state by using the merchant's internal merchant identifier.

getDetails.

☑ **Build your merchant onboarding interface for the Web Merchant Registration API**

Using the API streamlines the onboarding process for merchants by removing many of the steps required when they use they manage their own credentials. One step that remains is domain verification. To allow merchants to prepare and register their own domains, provide them with the necessary steps and files in your merchant portal.

- ✓ Download your domain verification file
- ✓ Instructions on how and where to host this on their domains
- ✓ Provide the list of domains that they want to register for use with Apple Pay

Build your Apple Pay solution

When building your Apple Pay integration, the process is largely the same as a merchant integrating Apple Pay on the web or in an App. You can follow step-by-step instructions on these processes using the Apple Pay Merchant Integration Guide.

Apple Pay Merchant Integration Guide

Integration types

Payment processors provide diverse integration options to cater to merchants, ranging from straightforward hosted payment forms, to seamless direct API integrations. To ensure an optimal Apple Pay experience, merchants should strategically place the Apple Pay button early in the checkout process, typically on the product or basket summary page. Payment processors should allow merchants to control the location of their Apple Pay experience within their checkout flow. Apple recommends direct API support or integrating Apple Pay into JavaScript or iOS code libraries for the best results.

☑ Direct API Integration

Payment processors should always support this method of integration for Apple Pay because it gives merchants complete control over how they integrate Apple Pay into their websites or apps. In this scenario, the merchant manages interactions with Apple Pay APIs and sends the Apple Pay payment data to the payment processor for consumption. Since payment processors often provide API fields for customer address and contact information, these can be extended to support encrypted payment data.

While most merchants rely on their payment processors to decrypt payment data, some may want to decrypt Apple Pay data themselves before sending it to their processors. Therefore, payment processors should consider extending their APIs to support Apple Pay payloads in both encrypted and decrypted formats.

☑ JavaScript & iOS Code Libraries

Payment processors offer these products to simplify and speed up integrations, including Apple Pay. They may wrap around Apple Pay APIs to reduce merchant effort in adopting Apple Pay.

It's recommended to build JavaScript and iOS libraries to simplify Apple Pay integration, yet give merchants control over presenting the Apple Pay button in their checkout flow.

☑ Hosted Payment Forms

Apple Pay is designed to streamline the checkout process. While it's possible to support Apple Pay on a hosted payment form, its full potential won't be realized. By the time users reach the hosted payment form, they've likely already provided their address and contact information—steps that could've been skipped if Apple Pay had been presented earlier.

If payment processors are interested in supporting Apple Pay on a hosted payment form, they should ensure that they enable onboarding merchants through the Apple Pay Web Merchant Registration API.

When integrating Apple Pay into the hosted payment form, the payment platform should follow the standard steps for integrating Apple Pay on the web. However, they should use the partnerInternalMerchantIdentifier value (created when registering a merchant via the Apple Pay Web Merchant Registration API) in all fields used for the Merchant Identifier.

☑ iframes

Apple Pay on the web uses a number of measures to ensure transactions remain secure and private. One of these measures is to ensure that a validated domain is being used during the Apple Pay payment. When using an iframe, this means that the top-level domain must be registered and verified, and this domain should be passed to the Apple Pay servers as the initiativeContext value during merchant validation. The iframe must also have the allow="payment" attribute in order to communicate correctly with the Apple Pay API's.

Offering Apple Pay

The system provides several Apple Pay button types and styles so that the merchant can choose the button type that fits best with the terminology and flow of their purchase or payment experience. In contrast to the Apple Pay buttons, the Apple Pay mark is used to communicate the availability of Apple Pay as a payment option. Don’t create your own Apple Pay button design or attempt to mimic the system-provided button designs.

☑ Present the Apple Pay button

The button will be rendered by the appropriate PassKit or Javascript API, which will display the most up to date version of the button as well as perform the appropriate localizations based on the users device settings.

🔗 Display the Apple Pay button - Apple Pay on web

🔗 Display the Apple Pay button - PassKit

☑ Apple Pay Mark

Use the Apple Pay mark to show that Apple Pay is an available payment option. The Apple Pay mark isn’t a button and shouldn’t be used to launch the payment sheet. Use only the artwork Apple provides, with no alterations other than height.

🔗 Apple Pay mark and Marketing Guidelines

☑ Check for Apple Pay Availability

To ensure that you only display the Apple Pay button to customers with a supported device, check for Apple Pay availability.

🔗 Checking for Apple Pay availability

canMakePayments()
Verifies that the device is capable of making Apple Pay payments; it doesn’t verify that the user has a provisioned card for use with Apple Pay on the device. You can call this method at any time. In App you can extend this to include a list of supported networks, and it will only return true if a card with a listed networks is available for payment.

🔗 Web - canMakePayments()

🔗 App - canMakePayments()

applePayCapabilites()
Verifies on Safari and third-party browsers that the device is capable of making Apple Pay payments. It also verifies on Safari browsers that the device has at least one payment credential provisioned in Wallet. Depending on the response, you can determine if the device supports Apple Pay and whether to display an Apple Pay button. Call this method if you want to default to Apple Pay during your checkout flow, or if you want to add an Apple Pay button to your product detail page.

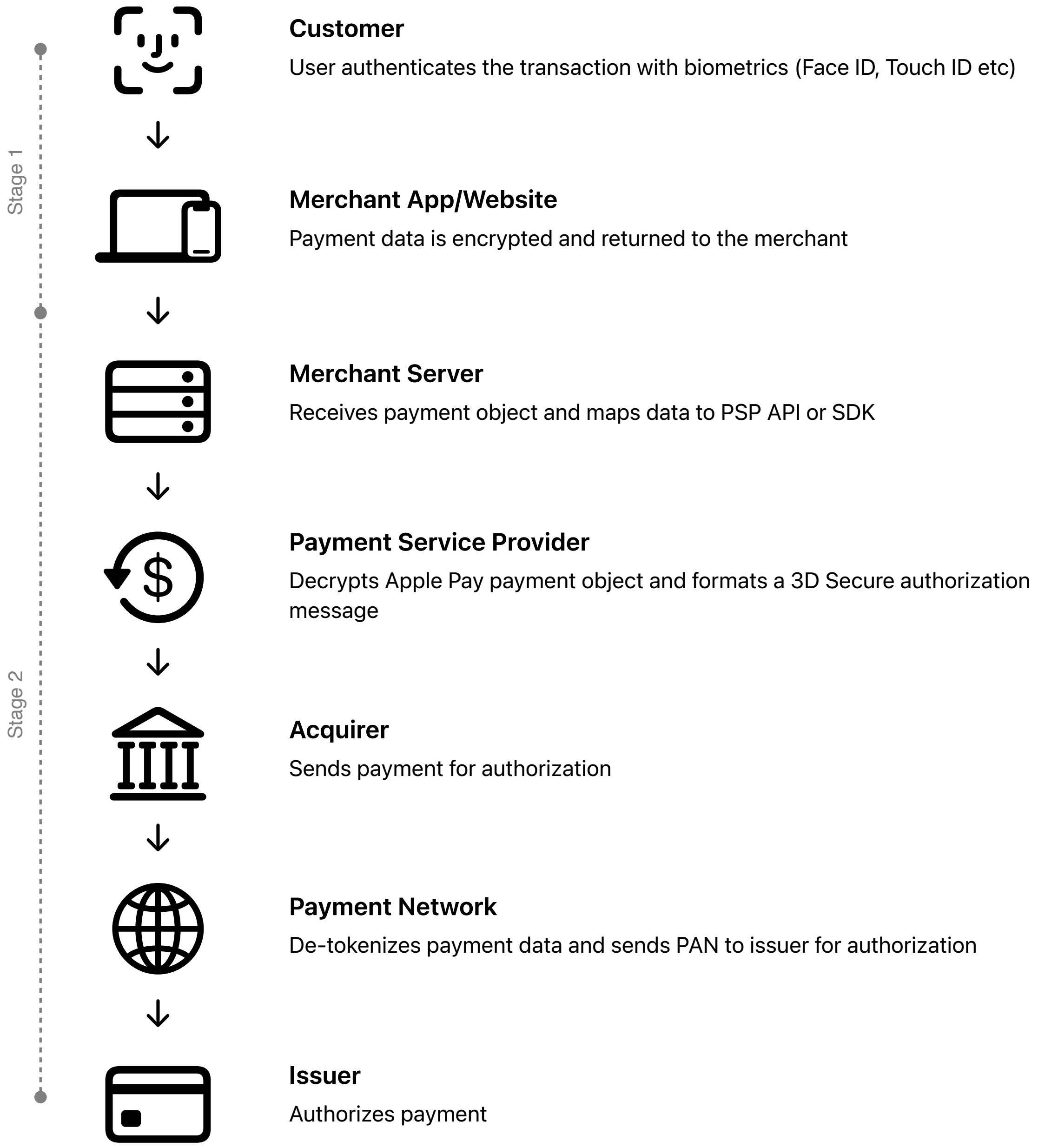
🔗 applePayCapabilities()

Processing Apple Pay

The diagram illustrates the flow of an Apple Pay transaction, showing the roles of different entities. These roles may vary based on the payment processor's integration method, but the interaction is divided into two stages:

Stage 1: Interacting with Apple Pay APIs to generate the payment data.

Stage 2: Using the payment data to process the payment.



Customer

User authenticates the transaction with biometrics (Face ID, Touch ID etc)

Merchant App/Website

Payment data is encrypted and returned to the merchant

Merchant Server

Receives payment object and maps data to PSP API or SDK

Payment Service Provider

Decrypts Apple Pay payment object and formats a 3D Secure authorization message

Acquirer

Sends payment for authorization

Payment Network

De-tokenizes payment data and sends PAN to issuer for authorization

Issuer

Authorizes payment

Supported Transaction Types and Business Models

Apple Pay supports most E-Commerce transaction types and offers the flexibility to accommodate simple to complex business models.

Below are examples of transaction types supported by Apple Pay.

Transaction type	Description	Example
Authorization & capture	Reserve funds on a customer’s account and transfer money to your bank	Online shopping where merchandise is readily available for shipping
Authorization & delayed capture	Reserve funds on a customer’s account and transfer money to your bank later	Pre-order, delivery service with tip, or order online to pick up in store
Authorization & capture with different amount	Reserve funds on a customer’s account and transfer money to your bank once an order is successfully completed for an amount higher or lower than authorized	Taxi, Scooter
Partial shipment	Divide a purchase into multiple payments for goods that are not shipped together	Multi-item purchase with more than one delivery
Fixed subscription	Handle repeating payments at a regular frequency and with a fixed amount	Monthly gym membership
Flexible frequency subscription	Handle payments for services where frequency is inconsistent or the user has a choice to vary frequency	Meal subscriptions, where user can skip deliveries, reschedule delivery dates, and change frequency of deliveries
Flexible amount subscription	Handle payments for services where price varies based on consumption	Utility bill, or subscription with a promotion on the first month
Managed subscription	Handle payments for services where the user can vary frequency and amount of services received	Meal delivery service with high flexibility
Canceled transactions	Reverse money for canceled, reimbursed, or disputed transactions	Chargeback, voided transaction, or refund
Card verification	Verify that the selected card is associated with a cardholder account that is valid and in good standing	$0 or $1 authorizations

Recurring Payments and Merchant tokens

A standard one-time payment with Apple Pay uses a device-specific token to securely complete the transaction. In addition to this, Apple Pay supports the use of merchant tokens to complete supported payments independent of a device. After the initial transaction, payment details can be stored to create follow-on transactions.

Different flags may be required for subsequent transactions depending on the payment network. For instance, Visa's MIT specification and MasterCard's DSRP specification outline the formatting of authorization messages for these payments. Payment processors interested in supporting these transactions should inquire with their acquiring partner or payment networks for further information.

☑ Support Merchant tokens (where applicable)

A merchant token associates a payment card, merchant, and user, allowing for continuity across multiple devices, even when a device is upgraded or a card is removed. When a payment request is created for a recurring, automatic reload, or a deferred payment, the merchant can provide essential information the customer directly in the payment sheet, potentially helping to decrease cart abandonment and increase conversion rates.

To request a merchant token, the merchant must include one of the following in their payment request:

recurringPaymentRequest
Recurring payments, such as subscriptions, can feature different payment intervals (for example, annually or monthly), and either regular or trial billing cycles.

🔗 recurringPaymentRequest

automaticReloadPaymentRequest
Automatic reload payments, such as store card top-ups, feature a balance threshold and a reload amount. The card automatically reloads with the reload amount when the account drops below the balance threshold.

🔗 automaticReloadPaymentRequest

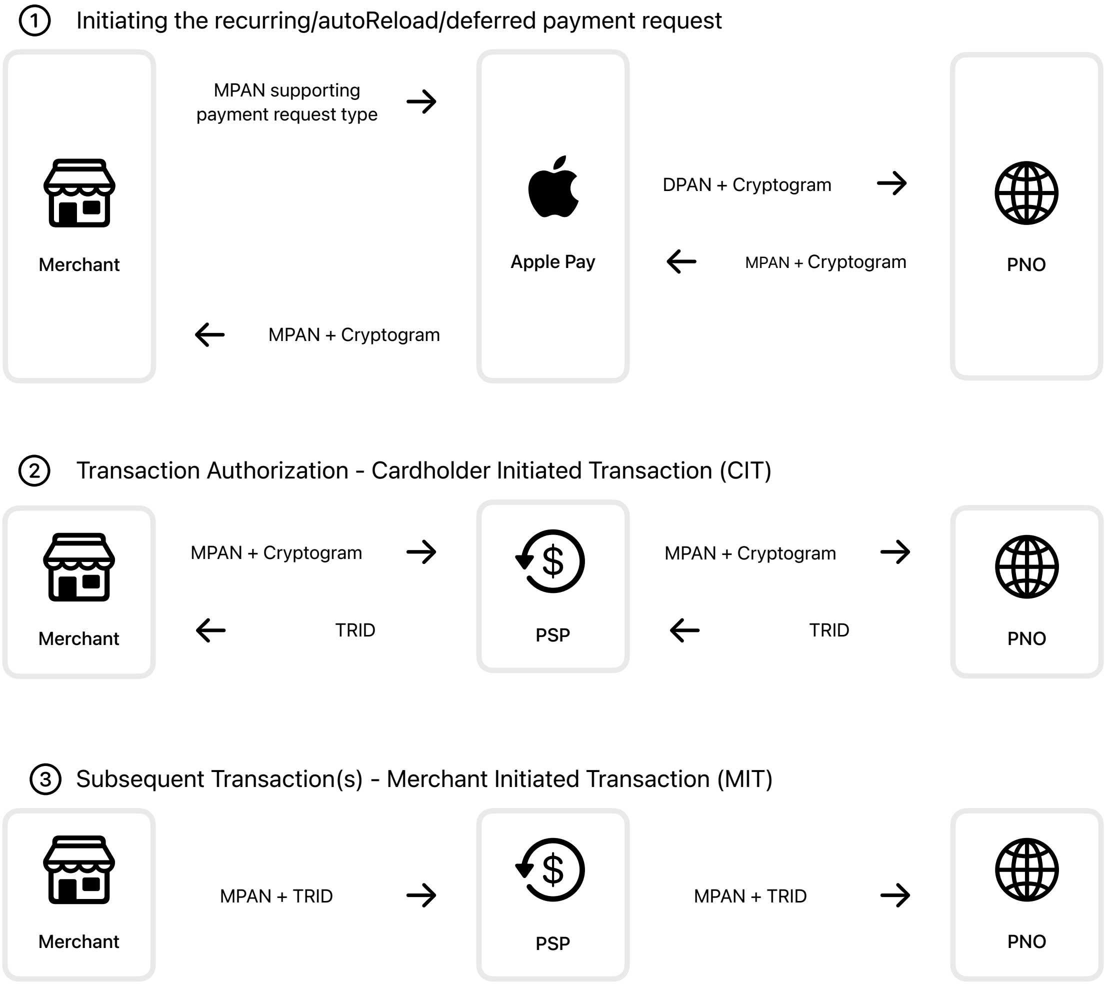
deferredPaymentRequest
Deferred payments, such as for a Hotel booking or a pre-order, allow you to specify a free cancellation period and specify the date when payment will be taken.

🔗 deferredPaymentRequest

When any of these request types are used to initiate an Apple Pay transaction, a merchant token is automatically requested by Apple, and where supported by the issuer, returned in the Apple Pay payload. If an issuer does not yet support merchant tokens, Apple Pay defaults back to returning the DPAN as it would for a regular Apple Pay transaction, but the customer still benefits from having the updated information in the payment sheet.

If you are supporting these transaction types, ensure that you have built in the functionality for your merchants to provide the additional fields necessary for these payment request types through your product offerings.

MPAN Creation and Authorization Flow



1 Initiating the recurring/autoReload/deferred payment request

2 Transaction Authorization - Cardholder Initiated Transaction (CIT)

3 Subsequent Transaction(s) - Merchant Initiated Transaction (MIT)

Merchant Token Management API

Merchant tokens are also designed to help prevent or resolve billing issues by giving visibility into important payment lifecycle updates, such as account status, bank card art, and card expiration date. The Apple Pay Merchant Token Management API allows you retrieve and manage these lifecycle updates.

When your app or website creates a payment request using one of the supported request types, it passes a notification URL in the tokenNotificationURL parameter. If a life-cycle event affects the token, Apple Pay sends a notification with an event identifier to that tokenNotificationURL. The details of the event can be retrieved by requesting the details from the Apple Pay server with the event identifier.

Receiving and Handling Merchant Token Notifications

When a life-cycle event occurs to a card associated with a merchant token, Apple Pay sends a **GET** request to the endpoint included in the tokenNotificationURL parameter of the original payment request.

```
GET https://merchant.example.com/tokenapi/notification/merchantToken/{eventId}
```

When you receive this request, use the eventId included in the path to format a **POST** request to retrieve the details of the event. This will include both the Merchant Identifier from the original transaction, alongside the specific eventId that has been triggered.

```
POST https://apple-pay-gateway.apple.com/paymentServices/v1/merchantId/{merchantId}/merchantToken/
event/{eventId}
```

Receiving and handling merchant token notifications

In response, you should receive a merchantTokenEventResponse object containing the details of the event. There are currently 3 event types that can be triggered.

UNLINK
The token have been revoked, and will be rejected if sent for processing. When receiving this event type the merchant may want to reach out to the customer to ask them to update their payment details before the next payment cycle.

UPDATED_METADATA
The metadata, such as the FPAN suffix, the expiry date, or the token itself has been updated.

UPDATED_CARD_ART
The card art for the underlying card has been updated.

```
{
  statusCode: 200,
  merchantTokenEvent: {
    merchantTokenIdentifier: 'MAPLAB001M317N593274cf108544f98420d15c5c2',
    eventType: 'UNLINK',
    reason: 'UNLINKDEVICE'
  }
}
```

merchantTokenEventResponse

Apple Pay Payment Object

When a customer confirms a transaction, the secure element on a device will create a payment object that contains a payment token with the encrypted payment data, alongside any additional contact data requested from the customer.

The Secure Element encrypts the token’s payment data using either elliptic curve cryptography (ECC) or RSA encryption. The encryption algorithm is selected by the Secure Element based on the payment request, with most regions using ECC encryption. RSA is used only in regions where ECC encryption is unavailable due to regulatory concerns. Unless advised otherwise, payment processors should focus on ECC encryption/decryption.

```
{
  "billingContact": {
    "addressLines": [
      "1 First Street"
    ],
    "administrativeArea": "London",
    "country": "United Kingdom",
    "countryCode": "GB",
    "familyName": "Appleseed",
    "givenName": "John",
    "locality": "London",
    "phoneticFamilyName": "",
    "phoneticGivenName": "",
    "postalCode": "AB12 3CD",
    "subAdministrativeArea": "",
    "subLocality": ""
  },
  "shippingContact": {
    "addressLines": [
      "1 First Street"
    ],
    "administrativeArea": "London",
    "country": "United Kingdom",
    "countryCode": "GB",
    "emailAddress": "john.appleseed@apple.com",
    "familyName": "Appleseed",
    "givenName": "John",
    "locality": "London",
    "phoneticFamilyName": "",
    "phoneticGivenName": "",
    "postalCode": "AB12 3CD",
    "subAdministrativeArea": "",
    "subLocality": ""
  },
  "token": {
    "paymentData": {
      "data": "K+kSR...BbAj",
      "signature": "MIAG...AAA=",
      "header": {
        "publicKeyHash": "2xR1...PGQ=",
        "ephemeralPublicKey": "MFkw...1w==",
        "transactionId": "ba9c...7298"
      },
      "version": "EC_v1"
    },
    "paymentMethod": {
      "displayName": "NetworkName 1234",
      "network": "NetworkName",
      "type": "credit"
    },
    "transactionIdentifier": "ba9c...7298"
  }
}
```

Decrypting the Payment Data

The Secure Element encrypts the token’s payment data using either elliptic curve cryptography (ECC) or RSA encryption. The encryption algorithm is selected by the Secure Element based on the payment request, with most regions using ECC encryption. RSA is used only in regions where ECC encryption is unavailable due to regulatory concerns. Unless advised otherwise, payment processors should focus on ECC encryption/decryption.

Payment Token Format Reference

Payment Processors should use the value of the publicKeyHash to determine which merchant public key Apple used for encryption, and then retrieve the corresponding certificate and private key for decryption. This is especially important as keys/certificates are being transitioned.

For a standard Apple Pay transaction that uses a device token, the decrypted payment information will be formatted as shown below, and contains all of the necessary data for processing the payment.

```
{
  "paymentDataType": "3DSecure",
  "applicationExpirationDate": "YYMMDD",
  "applicationPrimaryAccountNumber": "0001...1234",
  "currencyCode": "826",
  "deviceManufacturerIdentifier": "54...56",
  "transactionAmount": "10000",
  "paymentData": {
    "onlinePaymentCryptogram": "INX0...AwA=",
    "eciIndicator": “XX"
  }
}
```

When a merchant token has been issued, there will also be additional data within the decrypted payload related to that token.

```
{
  "paymentDataType": "3DSecure",
  "applicationExpirationDate": "YYMMDD",
  "applicationPrimaryAccountNumber": "0001...5678",
  "currencyCode": "826",
  "deviceManufacturerIdentifier": "54...56",
  "transactionAmount": "10000",
  "merchantTokenIdentifier": “MAPL...1654”,
  "merchantTokenMetadata": {
    "cardArt": [
      {
        "name": "card.png",
        "type": "image/png",
        "url": "https://...d91b"
      }
    ],
    "cardMetadata": {
      "longDescription": "Long Card Name",
      "shortDescription": "Short Name",
      "cardCountry": "GB",
      "fpanSuffix": "1234"
    }
  },
  "paymentData": {
    "onlinePaymentCryptogram": "TIA0...bdE=",
    "eciIndicator": “XX"
  }
}
```

Authorizing the Payment

Once the user has authorized themselves on the payment sheet the app / website will receive the Apple Pay payment object outlined in the previous section. At this point the user has verified their identity using Touch ID or Face ID but a payment has not yet taken place.

The merchant needs to pass the token information to the payment processor who can then create an authorization via the payment networks. As stated previously, the merchant can decrypt the payment data themselves, but more commonly the payment processor will manage this complexity on the merchant's behalf.

Constructing the Authorization Message

Once the payment data has been decrypted, it needs to be sent for authorization. Apple Pay has been designed to use similar data formatting as a 3D Secure transaction. Creating a successful authorization should be a case of mapping the Apple Pay fields to payment fields used for card/EMVCo token payments.

Some acquirers/networks may use existing 3D Secure fields whilst others may have dedicated fields for EMVCo tokens. The tables below are only for guidance, always check with your acquiring/payment network partner for the correct mapping.

Apple Pay Field	Representative Acquirer Field
data.applicationPrimaryAccountNumber	PAN
data.applicationExpirationDate (YYMMDD)	Expiration Date
data.paymentData.onlinePaymentCryptogram	TAVV / CAVV / UCAF
data.paymentData.eciIndicator	ECI

American Express

Payments using an American Express card may require slightly different mapping to that outlined above. In addition to the above mapping, please use the following table for guidance on mapping Amex payment fields.

Apple Pay Field	Amex Card Field
data.deviceManufacturerIdentifierToken	Token Requestor ID
data.paymentData.onlinePaymentCryptogram	Token Block A
[all zeros] *fill field value with zeros*	Token Block B

Good practice guidance when processing Apple Pay transactions

Apple Pay offers multiple layers of security beyond many other payment methods. However, you can apply the same industry best practices for risk mitigation to Apple Pay as you do for other E-commerce payment methods.

Apple Pay uses device based tokens (DPAN) so PANs cannot be used as a unique key and each transaction generates a unique cryptogram in place of the CV2. There are many standard checks that are still applicable to Apple Pay.

Velocity and Pattern Checks

A velocity check monitors specific data elements to check and compare a customer's shopping history with current purchases in order to identify any irregularities in their purchasing behaviour. This includes sudden bursts of high activity considered inconsistent with standard purchase behaviours for your industry or customer base.

Consider using purchase frequency, repeat billing/delivery address for high value purchases, cardholder name, email, name and IP address - particularly if combinations of these details are used in quick succession. Actions you could take based on a high risk transaction are:

- Decline the transaction.
- Delay shipping and complete outbound checks on address and customer details.

Delivery Checks

Consider implementing risk indicators for delivery methods such as the delivery address being different from the billing address, temporary addresses, instructions to leave goods on doorsteps (or similar), export address or requests for fast delivery. In countries where applicable, use Address Verification Services (AVS) to verify the billing address is correct. Consider looking at data provided by the user and check if it's linked to previous purchases (email addresses, phone numbers, billing and shipping addresses linked to other users). Multiple transactions could be correlated against another order.

Maintain a “Bad Transactions” List

Maintain a list of historical transactions for all payment methods which have either looked suspicious or resulted in chargebacks and compare the details of new transactions against these. If one or more of the transaction details match against historically bad transactions it may be worth putting the transaction aside for manual review or declining it outright. This data may also be available via your Payment Service Provider’s (PSP) Risk Management tool.

Email and IP address Checks

Ensure that email addresses are correct. You can use the Apple Pay Error Handling API's to ensure a customer adheres to your email rules. If the domain looks ok, send an email to the address to confirm it exists. Check the IP address coming from the device to verify the IP country matches the billing country and country where the card was issued.

Country Checks

If you identify certain countries as high risk, you can use the supportedCountries property in the Apple Pay payment request to limit payment cards to those that were issued in specific countries. The supportedCountries list does not affect the currency used for the transaction, and it applies to all payment cards in Wallet. You can also consider limiting IP addresses from high risk regions.

Testing

The Apple Pay sandbox environment enables merchants and developers to test their implementation of Apple Pay using test credit and debit cards. This testing process helps verify the customer experience and assess the ability to decrypt the transaction-specific payment data. To provision these test cards to a test device, the user must sign in to iCloud using a Sandbox Tester account.

🔗 Apple Pay Sandbox testing

It is also important to test Apple Pay in your production environment using real cards to ensure the end-to-end transaction flow is working as expected.

Test your integration on multiple devices and browsers

Apple Pay is now available on Safari, third party browsers in iOS and MacOS, as well as on other computing platforms. Check that the Apple Pay button is displayed on all the relevant devices and locations, and the you are able to support third party browsers/devices.

Data Mapping and formatting

Confirm that you are able to map all of the relevant data included in the Apple Pay payload, including the shipping and billing contact fields. Ensure you can handle this data in the various possible formats. For example, phone numbers may contain numeric values as well as special character such as “+”.

Responding to Events

When you are supporting handling of Payment Sheet event, such as when the shipping contact or method is updated, ensure you have tested your event handling process and that you are formatting and responding using the relevant completion method.

Test all integration types

Test Apple Pay functions as expected across all of the integration types you are offering. Pay close attention to integrations involving iframes to ensure they do not cause any errors during validation.

Documentation

Payment processors should provide detailed technical documentation to their merchants for all the ways they support Apple Pay via their platform. It is recommended that payment processors provide links to any existing content on the Apple Pay developer website, instead of recreating similar content on their own site.

Where possible, it is recommended that payment processors use code examples and step-by-step instructions to clearly explain the requirements for integrating Apple Pay.

Reporting

It is recommended that payment processors identify Apple Pay transactions in all transaction information and reporting made available to merchants.

Marketing Toolkit

Let your merchants know that Apple Pay is available with messaging across all of your channels. Highlight the ease of use, security, and privacy that Apple Pay brings to making payments. With the canMakePayments API, merchants can target their messaging to customers with devices that support Apple Pay. The Marketing Toolkit contains guidelines, tips, and templates to help create and implement marketing campaigns.

🔗 Apple Pay Marketing toolkit for PSP's

Frequently asked questions

Where does the customer information come from in the payment sheet?
The information comes from Wallet & Apple Pay defaults in Settings, if available, as well as the My Card in Contacts. It could also come from previous Apple Pay transactions. You can set up your My Card by going to Settings > Contacts > My Info.

Is the customer information that comes from Apple Pay verified by Apple?
Customer information is shared as-is, and is not verified by Apple. You will need to validate it on your platform and communicate through the Apple Pay API if fields should be corrected. For more information visit the Error Handling section of this guide.

What customer information can I pull from Apple Pay?
Customer information includes shipping and billing address, name, phone number and email address.

What does the Apple Pay payment token contain?
The structure, format, and data included in the PKPaymentToken can be found in the Payment Token Format Reference. PKPaymentToken only contains information on processing the payment; the customer information is included in the PKPayment object. The PKPayment encapsulates the customer information and the PKPaymentToken.

🔗 Payment Token Format Reference

Can I use the same Apple developer account for all countries I process payments in?
Yes; you do not need to have separate Apple developer accounts for multiple markets. You can also use the same Apple Merchant ID or Platform Integrator ID if you wish.

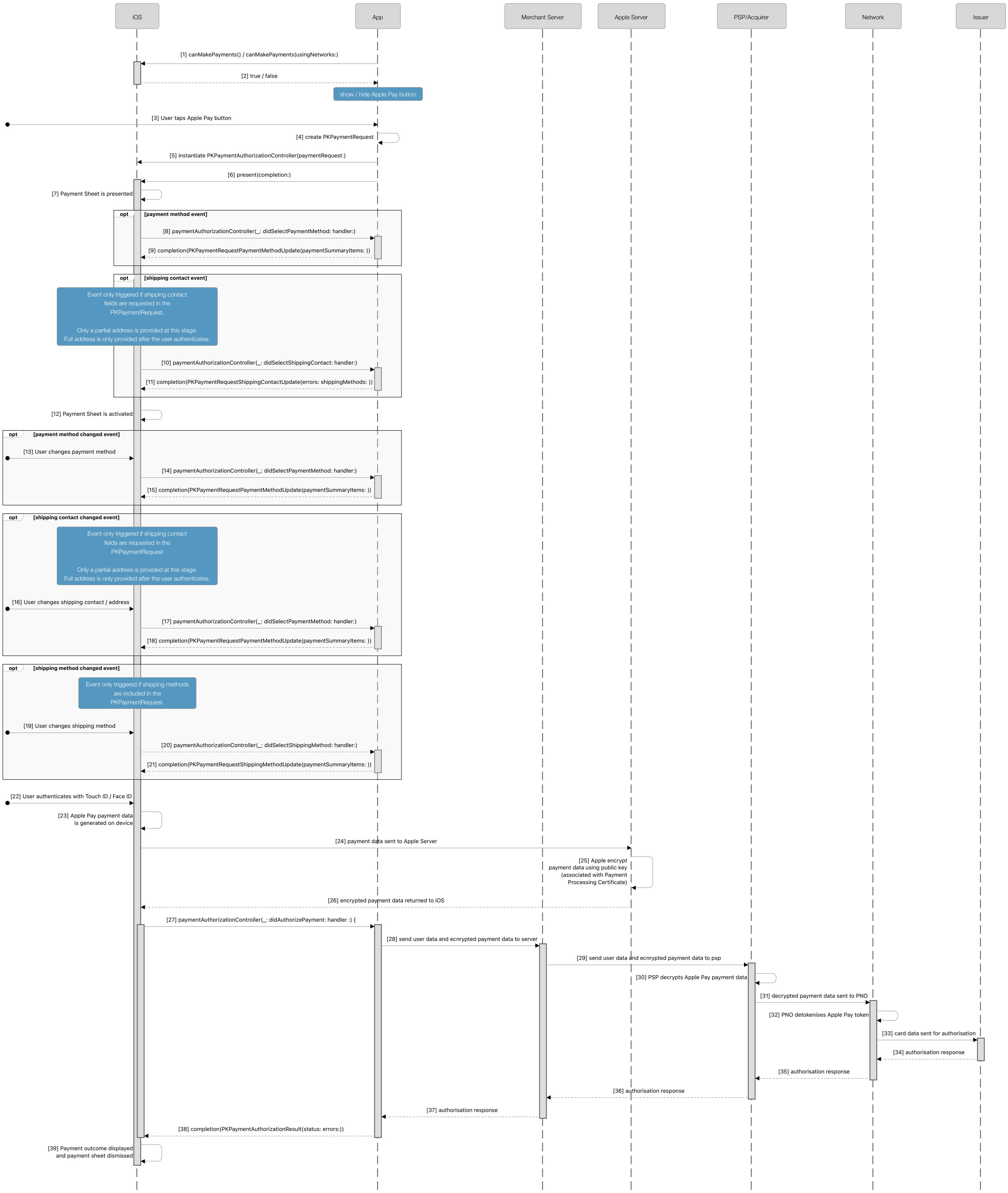
Troubleshooting

Further information and troubleshooting steps to debug common issues with Apple Pay can be found in the below guide.

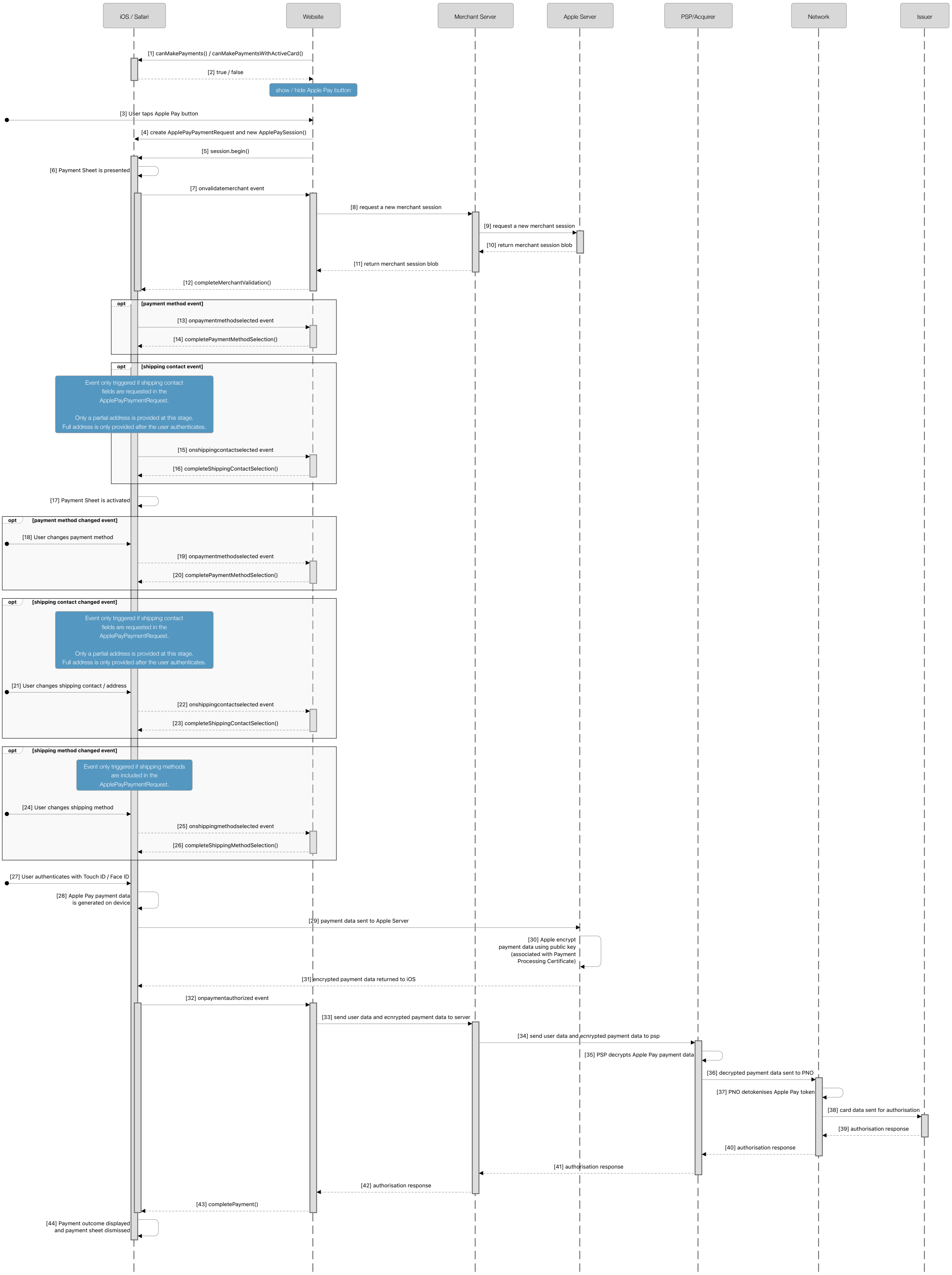
🔗 Apple Pay on the Web Troubleshooting Guide

API Diagrams

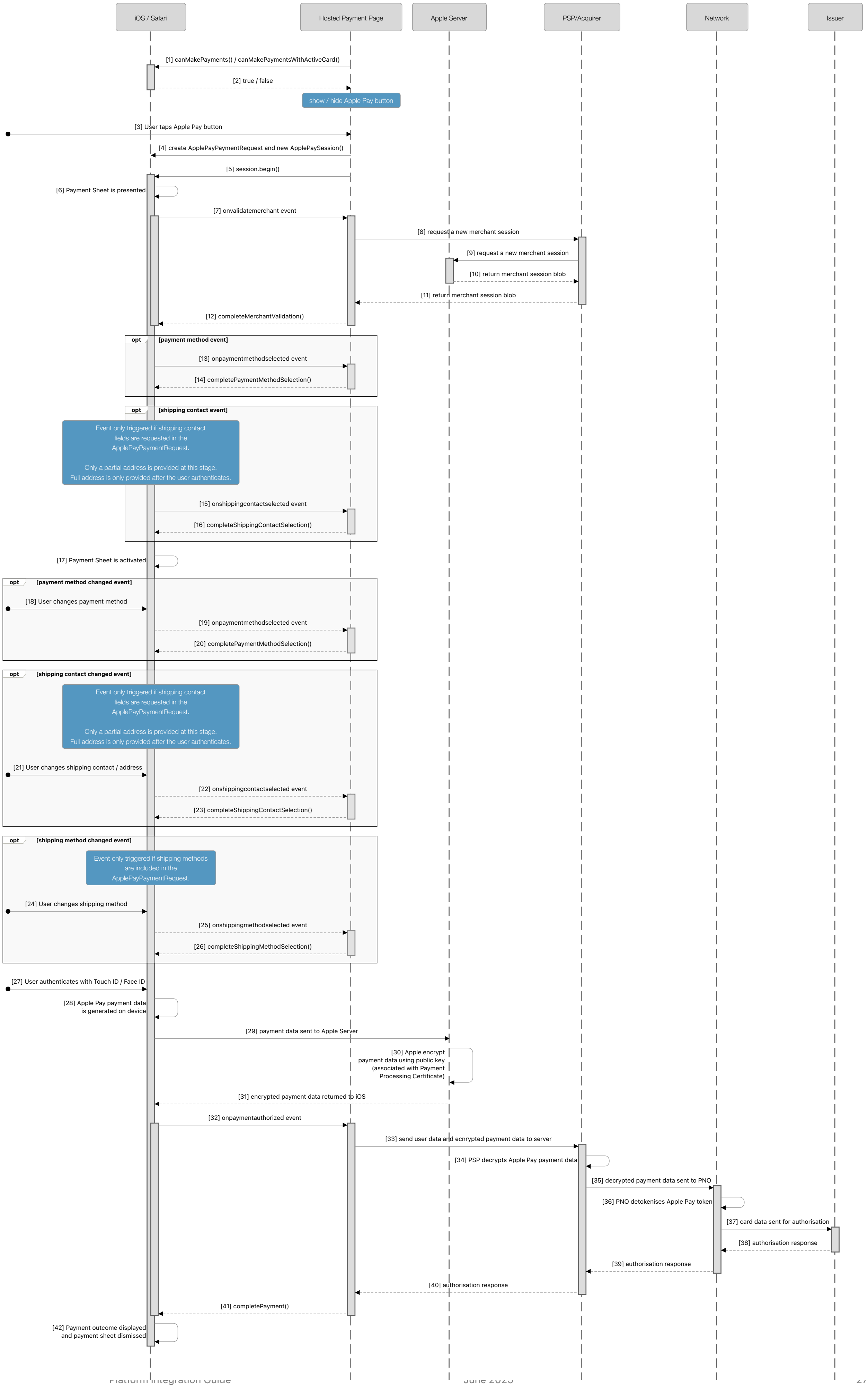
Apple Pay in app


iOS
App
Merchant Server
Apple Server
PSP/Acquirer
Network
Issuer
[1] canMakePayments() / canMakePayments(usingNetworks:)
[2] true / false
show / hide Apple Pay button
[3] User taps Apple Pay button
[4] create PKPaymentRequest
[5] instantiate PKPaymentAuthorizationController(paymentRequest:)
[6] present(completion:)
[7] Payment Sheet is presented
opt
[payment method event]
[8] paymentAuthorizationController(_: didSelectPaymentMethod: handler:)
[9] completion(PKPaymentRequestPaymentMethodUpdate(paymentSummaryItems:))
opt
[shipping contact event]
Event only triggered if shipping contact fields are requested in the PKPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[10] paymentAuthorizationController(_: didSelectShippingContact: handler:)
[11] completion(PKPaymentRequestShippingContactUpdate(errors: shippingMethods:))
[12] Payment Sheet is activated
opt
[payment method changed event]
[13] User changes payment method
[14] paymentAuthorizationController(_: didSelectPaymentMethod: handler:)
[15] completion(PKPaymentRequestPaymentMethodUpdate(paymentSummaryItems:))
opt
[shipping contact changed event]
Event only triggered if shipping contact fields are requested in the PKPaymentRequest
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[16] User changes shipping contact / address
[17] paymentAuthorizationController(_: didSelectPaymentMethod: handler:)
[18] completion(PKPaymentRequestPaymentMethodUpdate(paymentSummaryItems:))
opt
[shipping method changed event]
Event only triggered if shipping methods are included in the PKPaymentRequest.
[19] User changes shipping method
[20] paymentAuthorizationController(_: didSelectShippingMethod: handler:)
[21] completion(PKPaymentRequestShippingMethodUpdate(paymentSummaryItems:))
[22] User authenticates with Touch ID / Face ID
[23] Apple Pay payment data is generated on device
[24] payment data sent to Apple Server
[25] Apple encrypt payment data using public key (associated with Payment Processing Certificate)
[26] encrypted payment data returned to iOS
[27] paymentAuthorizationController(_: didAuthorizePayment: handler :) {
[28] send user data and ecnrypted payment data to server
[29] send user data and ecnrypted payment data to psp
[30] PSP decrypts Apple Pay payment data
[31] decrypted payment data sent to PNO
[32] PNO detokenises Apple Pay token
[33] card data sent for authorisation
[34] authorisation response
[35] authorisation response
[36] authorisation response
[37] authorisation response
[38] completion(PKPaymentAuthorizationResult(status: errors:))
[39] Payment outcome displayed and payment sheet dismissed

Apple Pay on the web


iOS / Safari
Website
Merchant Server
Apple Server
PSP/Acquirer
Network
Issuer
[1] canMakePayments() / canMakePaymentsWithActiveCard()
[2] true / false
show / hide Apple Pay button
[3] User taps Apple Pay button
[4] create ApplePayPaymentRequest and new ApplePaySession()
[5] session.begin()
[6] Payment Sheet is presented
[7] onvalidatemerchant event
[8] request a new merchant session
[9] request a new merchant session
[10] return merchant session blob
[11] return merchant session blob
[12] completeMerchantValidation()
opt
[payment method event]
[13] onpaymentmethodselected event
[14] completePaymentMethodSelection()
opt
[shipping contact event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[15] onshippingcontactselected event
[16] completeShippingContactSelection()
[17] Payment Sheet is activated
opt
[payment method changed event]
[18] User changes payment method
[19] onpaymentmethodselected event
[20] completePaymentMethodSelection()
opt
[shipping contact changed event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[21] User changes shipping contact / address
[22] onshippingcontactselected event
[23] completeShippingContactSelection()
opt
[shipping method changed event]
Event only triggered if shipping methods are included in the ApplePayPaymentRequest.
[24] User changes shipping method
[25] onshippingmethodselected event
[26] completeShippingMethodSelection()
[27] User authenticates with Touch ID / Face ID
[28] Apple Pay payment data is generated on device
[29] payment data sent to Apple Server
[30] Apple encrypt payment data using public key (associated with Payment Processing Certificate)
[31] encrypted payment data returned to iOS
[32] onpaymentauthorized event
[33] send user data and ecnrypted payment data to server
[34] send user data and ecnrypted payment data to psp
[35] PSP decrypts Apple Pay payment data
[36] decrypted payment data sent to PNO
[37] PNO detokenises Apple Pay token
[38] card data sent for authorisation
[39] authorisation response
[40] authorisation response
[41] authorisation response
[42] authorisation response
[43] completePayment()
[44] Payment outcome displayed and payment sheet dismissed

Apple Pay on the web:
PSP hosted payment page, merchant registered via API


iOS / Safari
Hosted Payment Page
Apple Server
PSP/Acquirer
Network
Issuer
[1] canMakePayments() / canMakePaymentsWithActiveCard()
[2] true / false
show / hide Apple Pay button
[3] User taps Apple Pay button
[4] create ApplePayPaymentRequest and new ApplePaySession()
[5] session.begin()
[6] Payment Sheet is presented
[7] onvalidatemerchant event
[8] request a new merchant session
[9] request a new merchant session
[10] return merchant session blob
[11] return merchant session blob
[12] completeMerchantValidation()
opt
[payment method event]
[13] onpaymentmethodselected event
[14] completePaymentMethodSelection()
opt
[shipping contact event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[15] onshippingcontactselected event
[16] completeShippingContactSelection()
[17] Payment Sheet is activated
opt
[payment method changed event]
[18] User changes payment method
[19] onpaymentmethodselected event
[20] completePaymentMethodSelection()
opt
[shipping contact changed event]
Event only triggered if shipping contact fields are requested in the ApplePayPaymentRequest.
Only a partial address is provided at this stage. Full address is only provided after the user authenticates.
[21] User changes shipping contact / address
[22] onshippingcontactselected event
[23] completeShippingContactSelection()
opt
[shipping method changed event]
Event only triggered if shipping methods are included in the ApplePayPaymentRequest.
[24] User changes shipping method
[25] onshippingmethodselected event
[26] completeShippingMethodSelection()
[27] User authenticates with Touch ID / Face ID
[28] Apple Pay payment data is generated on device
[29] payment data sent to Apple Server
[30] Apple encrypt payment data using public key (associated with Payment Processing Certificate)
[31] encrypted payment data returned to iOS
[32] onpaymentauthorized event
[33] send user data and ecnrypted payment data to server
[34] PSP decrypts Apple Pay payment data
[35] decrypted payment data sent to PNO
[36] PNO detokenises Apple Pay token
[37] card data sent for authorisation
[38] authorisation response
[39] authorisation response
[40] authorisation response
[41] completePayment()
[42] Payment outcome displayed and payment sheet dismissed