BluePay Apple Pay – User Guide

This documentation contains a step-by-step guide on getting set up to accept In-App Apple Pay transactions within your iOS app through a BluePay gateway account. If you are interested in accepting Apple Pay transactions through an NFC enabled terminal, please visit http://www.bluepay.com/blog/how-do-i-implement-apple-pay.

If you do not yet have a BluePay gateway account, please visit http://www.bluepay.com/get-started-today to get set up with one.

Contents
1. WHY USE APPLE PAY? .................................................................................................................. 1
2. HOW DOES APPLE PAY WORK? .................................................................................................. 2
3. HOW DO I ENABLE APPLE PAY THROUGH BLUEPAY? .......................................................... 4
   1. TO OBTAIN AN APPLE MERCHANT ID: .................................................................................. 4
4. GENERATING THE CSR ............................................................................................................... 5
   1. TO GENERATE AND OBTAIN THE CSR: ............................................................................. 5
5. SUBMITTING THE CSR TO APPLE ............................................................................................ 6
6. SETTING UP APPLE PAY VIA THE WEB .................................................................................... 11
7. APPLE PAY TRANSACTIONS WITHIN THE BLUEPAY GATEWAY MANAGER........................... 13

1. WHY USE APPLE PAY?
Every time your customers hand over their credit or debit card to pay, their card number and identity are visible. With Apple Pay, instead of using their actual credit or debit card number after adding their card to Passbook, a unique Device Account Number is assigned, encrypted, and securely stored in the Secure Element - a dedicated chip in the Apple device. These numbers are never stored on Apple servers. And when they make a purchase, the Device Account Number, along with a transaction-specific dynamic security code, is used to process the payment. So, the actual credit or debit card number is never shared by Apple with merchants or transmitted with payment, which means sensitive cardholder data is never passed through BluePay at any point.

2. **How Does Apple Pay Work?**

In terms of processing for non-NFC transactions, Apple Pay now offers two separate options. **Apple Pay via in-app purchases:** the merchant’s iOS app uses the Passkit Framework to request encrypted payment data from Apple. Apple then returns the encrypted payment data. At this point, there are two methods: the SDK method and the API method. If you want the processing to be handled through your iOS exclusively, use the Apple Pay SDK which can be found at [https://github.com/jslingerland/BluePay-Apple-Pay-SDK](https://github.com/jslingerland/BluePay-Apple-Pay-SDK). If you want the processing to be done outside of the iOS app and through a script running on your server, you will want to use the API method to send the POST request to BluePay. Documentation for the API request can be found at [http://www.bluepay.com/sites/default/files/documentation/BluePay_bp20post/Bluepay20post.txt](http://www.bluepay.com/sites/default/files/documentation/BluePay_bp20post/Bluepay20post.txt).

The image on the next page outlines both processes.
From your customers’ standpoint, the first step for them to be able to process an Apple Pay transaction would be open Passbook within their Apple Pay enabled device. From here, they would add a credit or debit card. Once the card information is sent to Apple and onto the tokenization service, a tokenized device PAN (Personal Account Number) is then stored onto the user’s device in place of their actual card data. Any stored cards on the user’s phone can be used in Apple Pay purchases through your iOS app; they just need to select the stored card to use on a per transaction basis.

**Apple Pay via the web:** similar to in-app purchases, Apple pay over the web works by presenting a payment pane to the user after they click on a ‘Pay By Apple Pay’ button that sits on your web server, whether that’s through a hosted payment form or some sort of an e-commerce application. Once the user selects the stored credit card and the shipping + billing address information, they then authorize the payment by holding down the Touch ID button. If approved, the customer is then redirected to a confirmation page.
3. **HOW DO I ENABLE APPLE PAY THROUGH BLUEPAY?**

The first step is to obtain a Certificate Signing Request (CSR) to upload to Apple to enable your iOS app or webform to encrypt payment data. Note: you will need an Apple Merchant ID to be able to process Apple Pay transactions. If you already have an Apple Merchant ID, you can skip section 3.1.

1. **TO OBTAIN AN APPLE MERCHANT ID:**
   1) If you do not yet have one, navigate to the Certificates, Identifiers, & Profiles area of the Member Center at the [https://developer.apple.com/membercenter](https://developer.apple.com/membercenter) web site.
   2) Under Identifiers, click on Merchant IDs and then click on the New button.
   3) Enter a description for the Merchant ID and the value for the Merchant ID itself. Note: your Merchant ID must begin with “merchant”.

![Register Merchant ID](image)

4) Click Continue and then the Register button. If successful, you should see a Registration complete page.
4. **GENERATING THE CSR**

A Certificate Signing Request (CSR) must be submitted to Apple to receive a necessary payment entitlement certificate.

1. **TO GENERATE AND OBTAIN THE CSR:**

1) Log into the BluePay Gateway Manager at [https://secure.bluepay.com](https://secure.bluepay.com)

2) Navigate to Tools -> Integration -> Apple Pay CSR

3) In the Apple Merchant ID field, type in your Apple Merchant Identifier **exactly** as it is shown in the developer Member Section of Apple. If these values do not match, BluePay will not be able to decrypt transaction data for your gateway account.
4) Click the Generate Apple CSR button

5) Click the Download button and save this file anywhere to your local machine.

5. **SUBMITTING THE CSR TO APPLE**

   1) Navigate to the Certificates, Identifiers, & Profiles area of the Member Center at the [https://developer.apple.com/membercenter](https://developer.apple.com/membercenter) website.
2) Under Certificates, Identifiers & Profiles, click on **Manage your certificates, App IDs, devices, and provisioning profiles.**

3) Under iOS Apps, click on **Certificates.**

4) Click on the Add button, then the radio button next to **Apple Pay Certificate**, then click on the Continue button.
5) Select your merchant ID that you used to generate the CSR from inside of the BluePay Gateway Manager, then click on the Continue button on this page and the next.
6) Click the Choose File... button, select the CSR that was generated from BluePay, and finally click on the Generate button.

7) If successful, you should see a Your Certificate is Ready page. Click the Download button to download the certificate.
Apple Pay In-app: For development purposes, you will need to double click on this .cer file to install in Keychain Access on your Mac. At this point, you’re all set for in-app development.

Apple Pay via the web: Apple added an additional step for processing Apple Pay through Safari. For the following, refer to Apple’s site for this: https://developer.apple.com/account/ios/identifier/merchant. You’ll also need to obtain a Merchant Identity Certificate from Apple. Follow the instructions below to create the CSR file and then to get the Certificate.

Click "Create Certificate"

Follow the "Create a CSR file. (Optional)" method then hit "Continue"

You'll see at the top of the next page that the act of using Keychain Access to create a CSR, actually creates a private key and certificate (aka public key) pair. These are both kept in keychainaccess on your mac. The public key/cert is also saved to disk when you create it, it's this xxx.certSigningRequest file which you'll upload to apple next

Once you upload your public key (xxx.certSigningrequest file), apple will use it to generate your Apple Pay Merchant Identity (certificate) - a file called merchant_id.cer

download this merchant_id.cer file, and double-click it to insert it into keychain access.app. This should automatically get appended to the existing entry for your Private key in keychain access.app

right-click that certificate (probably named "Merchant ID: merchant....") from within keychain access.app (you may need to expand the private key entry to see the certificate under it) and select "Export 'Merchant ID merchant....'". This will default to exporting a xxxx.p12 file to your desktop.

Now, convert this .p12 file into .pem format by running the following command in your terminal:

`openssl pkcs12 -in apple-pay-cert.p12 -out apple-pay-cert.pem -nodes -clcerts`
Once you have your Merchant Identity Certificate in .pem format, you’ll need to send BluePay this file so that we can validate you with Apple for every Apple Pay transaction.

To send BluePay this certificate, send an email to integrationsupport@bluepay.com and we’ll guide you through the process to send us this file securely.

6. SETTING UP APPLE PAY VIA THE WEB

Configuring your payment form to process Apple Pay transactions is fairly straightforward. Within your web form, you’ll need to reference the BluePay Apple Pay javascript SDK which can be found at the following URL: https://www.bluepay.com/sites/default/files/documentation/Apple_Pay_SDK/applepay.js. Simply include a src attribute within your web form like so:

```html
<script src="https://www.bluepay.com/sites/default/files/documentation/Apple_Pay_SDK/applepay.js"></script>
```

Firstly, check that the window.ApplePaySession class exists and is available within the user’s web browser. If so, make a call to canMakePayments to ensure that the user’s Apple device has the ability to accept Apple Pay.

```javascript
If (window.ApplePaySession && ApplePaySession.canMakePayments()) {
    // Apple Pay is good to go for this device
}
```

This should all be done before the Apple Pay button is displayed to the user. To go a step further using the above example, make a call to ApplePaySession.canMakePaymentsWithActiveCard which will check to make sure that the user has at least one stored credit card on their device that can be used for Apple Pay.

Next, you’ll need to instantiate a paymentRequest object. This is required to create an ApplePaySession. More information on the paymentRequest object can be found on Apple's API reference.

When creating the paymentRequest object, you’ll need to set the following values:

- `currencyCode`
- `countryCode`
- `supportedNetworks`
- `merchantCapabilities`

An example is shown below:

```javascript
var paymentRequest = {
    currencyCode: 'USD',
};
```
countryCode: 'US',
requiredBillingContactFields: ['postalAddress', 'email', 'name', 'phone'],
lineItems: [{label: 'Test Item #1', amount: '2.99'}, {label: 'Test Item #2', amount: '3.00'}],
total: {
  label: displayName,
  amount: '5.99'
},
supportedNetworks: ['amex', 'masterCard', 'visa', 'discover', 'jcb'],
merchantCapabilities: ['supports3DS', 'supportsEMV', 'supportsCredit', 'supportsDebit']
};

The next step is to send a validation request to Apple for the web server that your payment form is sitting in. To do this, you’ll need to include a session.onvalidatemerchant callback. If the validation fails, make sure that you throw an error and display a notification to the user. The SDK already handles this for you, but you can override the callback to add something like an alert or dialog box for the user.

```
session.onvalidatemerchant = function (event) {
  var promise = performValidation(event.validationURL);
  promise.then(function (merchantSession) {
    session.completeMerchantValidation(merchantSession);
  }, function (reason) {
    console.log("Validate merchant failed. Reason: " + reason);
    session.abort();
  });
}
```

You also need to set up a session.onpaymentauthorized callback. This is called by Safari when the user authorizes a payment request through your Apple Pay button. The example below will redirect the user to a receipt page after the user authorizes the payment and the transaction has been successfully processed through BluePay. If an error was encountered or if the payment was declined, the Apple Pay window is closed and the transaction result message is displayed to the user.

```
session.onpaymentauthorized = function (event) {
  var promise = sendPaymentToken(event.payment.token, paymentRequest);
  promise.then(function (success) {
    var status;
    if (success)
      applePayPane.hide();
      successDiv.show();
      status = ApplePaySession.STATUS_SUCCESS;
      session.completePayment(status);
      window.location = approvalURL;
  }, function (errorDecline) {
    resultDiv.text(errorDecline);
    resultDiv.show();
  });
}
```
status = ApplePaySession.STATUS_FAILURE;
session.completePayment(status);
}
}

Finally, to actually show the payment sheet to the user, call the begin method of the session object.

Note: you can ONLY call this method from an onclick method or some other explicit request from the user.

To see a running example of Apple Pay via the web, take a look at https://secure.bluepay.com/interfaces/shpf?SHPF_FORM_ID=applepay

7. **APPLE PAY TRANSACTIONS WITHIN THE BLUEPAY GATEWAY MANAGER**

All transactions processed as an Apple Pay transaction will show up like any other transaction in your transactions list. Refunding/voiding/capturing transactions should be done within the BluePay Gateway Manager, and are not supported by the BluePay Apple Pay in-app or JS SDKs.