iOS SDK for Apple Pay

Table of Contents

Introduction

This document provides knowledge in building the iOS app that accepts payment in iOS devices, with built-in support for Apple Pay.

If you want to build a mobile app like Lyft or Fancy and enable people to make purchases directly in your app, our iOS libraries can help. The library also supports Apple Pay so that your users can make frictionless payments without having to enter in their credit card info.

Accepting payments in your app involves 3 steps:

1. Collecting credit card information from your customer
2. Converting the credit card information to a single-use token
3. Sending this token to your server to create a charge

Getting started

Step 1: Install the library

Using CocoaPods

We recommend using CocoaPods to install the Stripe iOS library since it makes it easy to keep your app's dependencies up-to-date.

If you have not set up CocoaPods before, we recommend following the installation instructions on the CocoaPods site, then add `pod 'Stripe'` to your Podfile, and run `pod install`.

Note: Use the `.xcworkspace` file to open your project in Xcode instead of the `.xcodeproj` file.

Using Carthage

To use Carthage, simply add `github "stripe/stripe-ios"` to your `Cartfile` and follow the

Carthage Installation instructions.

Using manual installation

We publish our SDK as a static framework that you can copy directly into your app without any additional tools. To manually install the library, do the following:

1. Head to our releases page and downloads the framework that is right for you.
2. Unzip the file you downloaded.
3. In Xcode, with your project open, click **File > Add files**.
4. Select `Stripe.framework` in the directory you just unzipped.
5. Make sure **Copy items if needed** is checked.
6. Click **Add**.
7. In your project settings, go to the **Build Settings** tab, and under **Other Linker Flags**, make sure `-ObjC` is present.

Step 2: Configure API keys

First, configure Stripe with your published API key. We recommend doing this in your `AppDelegate`'s `application:didFinishLaunchingWithOptions` method so that it will be set for the entire lifecycle of your app.

Example in Swift

```
// AppDelegate.swift

import Stripe

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?)
-> Bool {
        Stripe.setDefaultPublishableKey("pk_test_6pRNASCoBOKtIshFeQd4XMUh")
        return true
    }
}
```

Example in Objective-C

```
// AppDelegate.m

#import "AppDelegate.h"
#import <Stripe/Stripe.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    [Stripe setDefaultPublishableKey:@"pk_test_6pRNASCoBOKtIshFeQd4XMUh"];
    return YES;
}

@end
```

Note: We have placed your test publishable API key as the `StripePublishableKey` constant in the above snippet. You will need to swap it out with your live publishable key in production. You can see all your API keys in your dashboard.

Collecting credit card information

There are three ways to obtain payment details from the user:

1. Use the Apple Pay framework to access your users' stored payment information.
2. Use our pre-built form component, STPPaymentCardTextField to collect new credit card details.
3. Build your own credit card form from scratch.

Since Apple Pay supports only certain credit cards on the latest iOS devices, we recommend using Apple Pay in combination with option 2 or option 3 as a fallback on devices where Apple Pay is not available.

Using Apple Pay

With Apple Pay, you are able to access payment information stored on your customer' iOS devices.

Important note before starting
To use Apple Pay, you need to add the Apple Pay capability to your app in Xcode. This requires creating a merchant ID with Apple first, as explained in the "Getting Started with Apple Pay" documentation.

After setting up the merchant ID, you need to generate a `PKPaymentRequest` to submit to Apple. We have provided a convenient method to generate one with reasonable defaults. All you need to do is set the `paymentSummaryItems` property to an array of `PKPaymentSummaryItems` . These are analogous to line items on a receipt and are used to explain your charge to the user. For more details, see the PKPaymentRequest documentation.

Now that you have created the request, the next step is to query the device to see if Apple Pay is available; that is, if your app is running on the latest hardware and the user has added a valid credit card. `YOUR_APPLE_MERCHANT_ID` is an identifier that you obtain directly from Apple. If Apple Pay is available, you should create and display the payment request view controller. To do so, follow the below code:

Example in Swift

```
// ViewController.swift

guard let request = Stripe.paymentRequestWithMerchantIdentifier("YOUR_APPLE_MERCHANT_ID") else {
    // request will be nil if running on < iOS8
    return
}
request.paymentSummaryItems = [
    PKPaymentSummaryItem(label: "Premium Llama Food", amount: 10.0)
]

if (Stripe.canSubmitPaymentRequest(request)) {
    let paymentController = PKPaymentAuthorizationViewController(paymentRequest: request)
    presentViewController(paymentController, animated: true, completion: nil)
} else {
    // Show the user your own credit card form (see options 2 or 3)

}
```

Example in Objective-C

```
// ViewController.m

PKPaymentRequest *request = [Stripe paymentRequestWithMerchantIdentifier:"YOUR_APPLE_MERCHANT_ID"];
NSString *label = @"Premium Llama Food";
NSDecimalNumber *amount = [NSDecimalNumber decimalNumberWithString:@"10.00"];
request.paymentSummaryItems = @[
[PKPaymentSummaryItem summaryItemWithLabel:label
                                    amount:amount]
];

if ([Stripe canSubmitPaymentRequest:request]) {
    PKPaymentAuthorizationViewController *paymentController;
    paymentController = [[PKPaymentAuthorizationViewController alloc] initWithPaymentRequest:paymentRequest];
    paymentController.delegate = self;
    [self presentViewController:paymentController animated:YES completion:nil];

} else {
// Show the user your own credit card form (see options 2 or 3)
}
```

Note that `ViewController` is a `PKPaymentAuthorizationViewControllerDelegate`. By implementing this protocol, the `PKPayment` is handled to return the authorization controller.

Example in Swift

```
// ViewController.swift

func paymentAuthorizationViewController(controller: PKPaymentAuthorizationViewController, didAuthorizePayment payment:
PKPayment, completion: (PKPaymentAuthorizationStatus) -> Void) {
/*
We'll implement this method below in 'Creating a single-use token'.
Note that we've also been given a block that takes a PKPaymentAuthorizationStatus.
We'll call this function with either PKPaymentAuthorizationStatusSuccess or PKPaymentAuthorizationStatusFailure
after all of our asynchronous code is finished executing.
This is how the PKPaymentAuthorizationViewController knows when and how to update its UI.
*/
handlePaymentAuthorizationWithPayment(payment, completi
on: nil)
}

func paymentAuthorizationViewControllerDidFinish(controller: PKPaymentAuthorizationViewController) {
    dismissViewControllerAnimated(true, completion: nil)
}
```

Example in Objective-C

```
// ViewController.m

- (void)paymentAuthorizationViewController:(PKPaymentAuthorizationViewController *)controller
                       didAuthorizePayment:(PKPayment *)payment
                                completion:(void (^)(PKPaymentAuthorizationStatus))completion {
/*
We'll implement this method below in 'Creating a single-use token'.
Note that we've also been given a block that takes a PKPaymentAuthorizationStatus.
We'll call this function with either PKPaymentAuthorizationStatusSuccess or PKPaymentAuthorizationStatusFailure
after all of our asynchronous code is finished executing.
This is how the PKPaymentAuthorizationViewController knows when and how to update its UI.
*/
[self handlePaymentAuthorizationWithPayment:payment completion:completion];
}

- (void)paymentAuthorizationViewControllerDidFinish:(PKPaymentAuthorizationViewController *)controller {
[self dismissViewControllerAnimated:YES completion:nil]
;
}
```

To implement optional `PKPaymentAuthorizationViewControllerDelegate` methods for customer events (such as, to recalculate shipping costs based on user selection, see the PKPaymentAuthorizationViewController documentation.

Important: Before doing the next step, make sure the controller has returned with a `PKPayment`.

Using STPPaymentCardTextField

To use our pre-built form component, you need to create a view controller called `PaymentViewController` and add an `STPPaymentCardTextField` property to it.

Example in Swift

```
// PaymentViewController.swift

class PaymentViewController: UIViewController, STPPaymentCardTextFieldDelegate {
    let paymentTextField = STPPaymentCardTextField()
}
```

Example in Objective-C

```
// PaymentViewController.m
```

```
#import "PaymentViewController.h"

@interface PaymentViewController ()<STPPaymentCardTextField Delegate>
@property(nonatomic) STPPaymentCardTextField *paymentTextField;
@end
```

To instantiate the `STPPaymentCardTextField`, set the `PaymentViewController` as its `STPPaymentCardTextFieldDelegate` and add it to your view.

Example in Swift

```
// PaymentViewController.swift

override func viewDidLoad() {
    super.viewDidLoad();
    paymentTextField.frame = CGRectMake(15, 15, CGRectGetWidth(self.view.frame) - 30, 44)
    paymentTextField.delegate = self
    view.addSubview(paymentTextField)
}
```

Example in Objective-C

```
// PaymentViewController.m

- (void)viewDidLoad {
    [super viewDidLoad];
    self.paymentTextField = [[STPPaymentCardTextField alloc] initWithFrame:CGRectMake(15, 15, CGRectGetWidth(self.view.frame) -
30, 44)];
    self.paymentTextField.delegate = self;
    [self.view addSubview:self.paymentTextField];
}
```

By adding an `STPPaymentCardTextField` to the controller, your app is enabled to accept card numbers, expiration dates, and CVCs. It will also format the input and validate that input on-the-fly.

When a user enters text into this field, the `paymentCardTextFielddidChange` method will be called on our view controller. In this callback, we can enable a save button to allow users to submit their valid cards if the form is valid.

Example in Swift

```
func paymentCardTextFieldDidChange(textField: STPPaymentCardTextField) {
    // Toggle navigation, for example
    saveButton.enabled = textField.isValid
}
```

Example in Objective-C

```
- (void)paymentCardTextFieldDidChange:(STPPaymentCardTextField *)textField { {
    // Toggle navigation, for example
    self.saveButton.enabled = textField.isValid;
}
```

Building your own form

To build your own form, make sure you will at least be able to collect your customer’s card numbers and expiration dates. We recommend you to also collect the CVC to prevent fraudulent. The user’s name and billing address are optional and would benefit you in terms of fraud protection.

Creating tokens

Our libraries shoulder the burden of PCI compliance by helping you avoid the need to send card data directly to your server. Instead, our libraries send credit card directly to our servers where we can then convert them to tokens. You can charge these tokens later in your server-side code.

Using PKPayment (Apple Pay)

After your PKPayment has arrived, you can turn it into a single-use Stripe token with a simple method call by using the following code.

Example in Swift

```
// ViewController.swift

func handlePaymentAuthorizationWithPayment(payment: PKPayment, completion: PKPaymentAuthorizationStatus -> ()) {
    STPAPIClient.sharedClient().createTokenWithPayment(payment) { (token, error) -> Void in
        if error != nil {
            completion(PKPaymentAuthorizationStatus.Failure)
            return
        }
        /*
        We'll implement this below in "Sending the token to your server".
        Notice that we're passing the completion block through.
        See the above comment in didAuthorizePayment to learn why.
        */
        createBackendChargeWithToken(token, completion: completion)
    }
}
```

Example in Objective-C

```
// ViewController.m

- (void)handlePaymentAuthorizationWithPayment:(PKPayment *) payment
                                   completion:(void (^)(PKPaymentAuthorizationStatus))completion {
    [[STPAPIClient sharedClient] createTokenWithPayment:payment
        completion:^(STPToken *token, NSError *error) {
            if (error) {
                completion(PKPaymentAuthorizationStatus Failure);
                return;
            }
            /*
            We'll implement this below in "Sending the token to your server".
            Notice that we're passing the completion block through.
            See the above comment in didAuthorizePayment to learn why.
            */
            [self createBackendChargeWithToken:token completion:completion];
        }];
}
```

Using STPCardParams

If you choose to use `STPPaymentCardTextField` or your own form, you can assemble the data into an `STPCardParams` object. After you have collected the card number, expiration dates, and CVC, package them up in an `STPCardParams` object and invoke the `createTokenWithCard` method on the `STPAIClient` class, instructing the library to send off the credit card data to Stripe and return a token.

To do so, add the below code to your program.

Example in Swift

```
@IBAction func save(sender: UIButton) {
    if let card = paymentTextField.card {
        STPAPIClient.sharedClient().createTokenWithCard(card) { (token, error) -> Void in
            if let error = error {
                handleError(error)
            }
            else if let token = token {
                createBackendChargeWithToken(token) { status in
```

```
                ...
            }
        }
    }
}
```

Example in Objective-C

```
- (IBAction)save:(UIButton *)sender {
    [[STPAPIClient sharedClient]
        createTokenWithCard:self.paymentTextField.card completion:^(STPToken *token, NSError *error) {
            if (error) {
                [self handleError:error];
            } else {
                [self createBackendChargeWithToken:token completion:^(PKPaymentAuthorizationStatus status) {
                    ...
                }];
            }
    }];
}
```

Note: In the example above, the `createTokenWithCard` is called when a save button is tapped. It is important that the `createToken` is not called before user has finished entering their card details.

It is up to you to handle error messages and show activity indicators while creating the token.

Sending the token to your server

The block you gave to `createToken` in the previous steps is called whenever Stripe returns with a token (or error). To charge the card, you need to send the token off to your server.

Here is how it looks for a token created with Apple Pay.

Example in Swift

```
// ViewController.swift

func createBackendChargeWithToken(token: STPToken, completion: PKPaymentAuthorizationStatus -> ()) {
    let url = NSURL(string: "https://example.com/token")!
    let request = NSMutableURLRequest(URL: url)
    request.HTTPMethod = "POST"
    let body = "stripeToken=(token.tokenId)"
    request.HTTPBody = body.dataUsingEncoding(NSUTF8StringEncoding)
    let configuration = NSURLSessionConfiguration.ephemeralSessionConfiguration()
    let session = NSURLSession(configuration: configuration)
    let task = session.dataTaskWithRequest(request) { (data, response, error) -> Void in
        if error != nil {
            completion(PKPaymentAuthorizationStatus.Failure)
        }
        else {
            completion(PKPaymentAuthorizationStatus.Success)
        }
    }
    task.resume()
}
```

Example in Objective-C

```
// ViewController.m

- (void)createBackendChargeWithToken:(STPToken *)token
                          completion:(void (^)(PKPaymentAuthorizationStatus))completion {
    NSURL *url = [NSURL URLWithString:@"https://example.com /token"];
    NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
    request.HTTPMethod = @"POST";
    NSString *body = [NSString stringWithFormat:@"stripeToken=%@", token.tokenId];
    request.HTTPBody = [body dataUsingEncoding:NSUTF8StringEncoding];
    NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessionConfiguration];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration];
    NSURLSessionDataTask *task =
```

```
    [session dataTaskWithRequest:request
        completionHandler:^(NSData *data,
                            NSURLResponse *response,
                            NSError *error) {
            if (error) {
                completion(PKPaymentAuthorizationStatusFailure);
            } else {
                completion(PKPaymentAuthorizationStatusSuccess);
            }
        }];
    [task resume];
}
```

Note: The completion callback above is Apple Pay-specific. If you are not using Apple Pay, the code is still mostly the same, but you would want to implement custom error and success handling.

On the server, implement an endpoint that will accept the `stripeToken` parameter. Make sure any communication with your server is SSL-secured to prevent eavesdropping.

After you have a Stripe token representing a card on your server, charge it, save it for charging later, or sign up for a subscription.

For more details, see the full example application.