iOS SDK

Integration Guide

SDK-Version 1.2.4

last update: **2021-03-29**

Document history

Date	Change
2016-12-01	First Version
2017-02-13	Pre-final version
2017-05-15	Extended for changes in SDK version 0.3
2017-05-22	Added note about the use of Let’s Encrypt SSL certificates on merchant backend
2017-06-16	Just added a new version number for first live release.
2017-06-19	Fixed spelling errors
2018-03-04	WeChat pay
2021-03-29	Update the wechat sdk version to 1.8.4 which uses WKWebview instead of UIWebview

Introduction

1. Overview

This is the documentation for the Computop iOS SDK, which describes how to integrate payments in your iOS app.

The integration of the SDK is achieved by following a list of mandatory steps as described below:

A. Configuration of the merchant account in the Computop Paygate system.
B. ‘Computop’ SDK pod installation.
C. Configuration of the SDK by inserting appropriate data retrieved from Computop.
D. Configuration of preferable payment methods.
E. Authentication against the merchant backend.
F. Insertion of the payment data.
G. Checkout.
H. Handling of errors.

1.1 List of supported payment methods

The iOS SDK currently supports the following payment methods:

- Credit Card
- Direct Debit
- PayPal
- Apple Pay
- WeChat

For more information please check below chapter 4.3.

2. Requirements

Requirements in order to be able to use the SDK:

Preparation

- Existing merchant account at Computop
- Merchant ID which you will receive from computop after creation of the merchant account
- Backend and the belonging URL on merchant side to create and deliver a auth token (see chapter 4.2)
- Website and the belonging URL's on merchant side to forward and show the status of a payment process in case of a success, failure or a notify event (see chapter 4.3.2).

Development

- Installed cocoapods on the development machine - minimum version of cocoa pods is v1.1.1
- Minimum required Xcode version is Xcode 8
- iOS 10 as minimum deployment target

Apple Pay

- Registered Apple Developer Account
- Configured app in the Apple Developer center - activated for Apple Pay
- Configured Apple Merchant ID
- Certificate Signing-Request-File received from Computop to create a Payment Processing Certificate

For more information see below in chapter 4.4 Apple Pay.

3. Installation

If not already done, please install cocoapods. Here https://guides.cocoapods.org/using/getting-started.html you will find the HowTo for that.

When CocoaPods is installed, you need to activate your Xcode project for CocoaPods in the way that is described here: https://guides.cocoapods.org/using/using-cocoapods.html

That means you have to create a Podfile inside the root directory of your project. You could do this with the following command within terminal in the root folder of your project:

```
pod init
```

Then add the following line to your Podfile:

```
pod 'Computop'
```

and run the following command within terminal in the root folder of your project:

```
pod install
```

If there comes an error like "`[!] Unable to find a specification for `Computop``" you should update your local CocoaPods repositories by

```
pod update
```

After them try again "`pod install`".

Now you should have a configured xcode workspace with the integrated Computop Framework. From now use the workspace file to open your Xcode project.

4. How to implement

You could find an demo on
https://github.com/computop/Computop-iOS/tree/master/Computop-demo

Try the demo in order to see how it works or use the following step by step guideline.

4.1 Configuration

Configure the SDK by importing the Computop class and inserting the configuration parameters you receive from Computop or parameter *merchantAppleID* that you create yourself.

The `AppDelegate` class is the appropriate place to do so.

```
#import "AppDelegate.h"
#import <Computop/Computop.h>
```

```objc
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
     CMPConfiguration.merchantID      = @"YOUR_MERCHANT_ID";
     CMPConfiguration.merchantAppleID = @"YOUR_APPLE_MERCHANT_ID";

    return YES;
}

@end
```

4.2 Authentification

One requirement for the Mobile SDK is it to insert the respective Merchant's URL in order to be able to receive the auth token. The SDK is responsible to retrieve then the token under the hood and use it appropriately when executing payment requests. An appropriate place to do that is inside `AppDelegate` with the rest of Configuration. For more information, see the Paygate documentation.

```objc
#import "AppDelegate.h"
#import <Computop/Computop.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
     CMPConfiguration.merchantID      = @"YOUR_MERCHANT_ID";
     CMPConfiguration.merchantAppleID = @"YOUR_APPLE_MERCHANT_ID";
     CMPConfiguration.authURL   = @"YOUR_AUTH_URL";

    return YES;
}

@end
```

4.3 Making a payment

Assumed your app user inserted some products into the basket, typed the shipping address and now wants to make the payment - that means you want to provide/show different payment methods to the user, so that he can choose from.

For that you need to ask the SDK for supported payment methods:

4.3.1 Retrieve Payment Methods

Currently the SDK returns all the available methods by calling the method in the following code snippet. Merchant can use only the methods they are activated in Paygate.

```
[[Computop sharedInstance] paymentMethodsOnSuccess:^(NSArray<CMPPaymentMethod *>
*paymentMethods) {
        // set the methods as a data source
    } onFailure:^(NSError *error) {
        // do something with the error
}];
```

A good place to do this is a view controller class, which controls the view of the payment methods. So you could set this as data source for your view. In the example app we are doing this in the `PaymentViewController` class

Now in order to proceed with a payment it is necessary to configure the appropriate payment data for each payment method.

4.3.2 Configure Payment Data

Every received payment method holds a `CMPPaymentData` instance. You only need to insert all merchant's necessary payment data for the checkout (more details on PaymentData in Paygate documentation). A good way to do that is after the receive of the supported payment methods:

```
[[Computop sharedInstance] paymentMethodsOnSuccess:^(NSArray<CMPPaymentMethod *>
*paymentMethods)
{
   // set the methods as a data source
   self.paymentMethods = paymentMethods;

   for (CMPPaymentMethod *method in self.paymentMethods)
   {
      // Mandatory params
      [method.paymentData setParamWithKey:@"TransID" withValue:@"YOUR_TRANS_ID"];
      [method.paymentData setParamWithKey:@"Amount" withValue:@"YOUR_AMOUNT";
      [method.paymentData setParamWithKey:@"Currency" withValue:@"YOUR_CURRENCY"];
      [method.paymentData setParamWithKey:@"URLSuccess" withValue:@"YOUR_URL_SUCCESS"];
      [method.paymentData setParamWithKey:@"URLNotify" withValue:@"YOUR_URL_NOTIFY"];
      [method.paymentData setParamWithKey:@"URLFailure" withValue:@"YOUR_URL_FAILURE"];

       if ([method.pmID isEqualToString:@"pm_cc"]) {
           [method.paymentData setParamWithKey: @"MsgVer" withValue: @"YOUR_MSGVER"];
```

```
      }

     // Url sucess, failure of paypal may be different from other payments
      if([method.pmID isEqualToString:@"pm_paypal"]) {

      [method.paymentData setParamWithKey:@"URLSuccess" withValue:@"YOUR_URL_SUCCESS_PAYPAL"];
      [method.paymentData setParamWithKey:@"URLFailure" withValue:@"YOUR_URL_FAILURE_PAYPAL"];

      } else {

      [method.paymentData setParamWithKey:@"URLSuccess" withValue:@"YOUR_URL_SUCCESS"];
      [method.paymentData setParamWithKey:@"URLFailure" withValue:@"YOUR_URL_FAILURE"];
      }

     // Optional params
     [method.paymentData setParamWithKey:@"RefNr" withValue:@"YOUR_REF_NR"];
     [method.paymentData setParamWithKey:@"OrderDesc" withValue:@"YOUR_ORDER_DESC"];
     [method.paymentData setParamWithKey:@"AddrCity" withValue:@"YOUR_ADDR_CITY"];
     [method.paymentData setParamWithKey:@"FirstName" withValue:@"YOUR_FIRST_NAME"];
     [method.paymentData setParamWithKey:@"LastName" withValue:@"YOUR_LAST_NAME"];
     [method.paymentData setParamWithKey:@"AddrZip" withValue:@"YOUR_ADDR_ZIP"];
     [method.paymentData setParamWithKey:@"AddrStreet" withValue:@"YOUR_ADDR_STREET"];
     [method.paymentData setParamWithKey:@"AddrState" withValue:@"YOUR_ADDR_STATE"];
     [method.paymentData setParamWithKey:@"Phone" withValue:@"YOUR_PHONE"];
     [method.paymentData setParamWithKey:@"eMail" withValue:@"YOUR_EMAIL"];
     [method.paymentData setParamWithKey:@"ShopID" withValue:@"YOUR_SHOP_ID"];
     [method.paymentData setParamWithKey:@"Subject" withValue:@"YOUR_SUBJECT"];
   }
} onFailure:^(NSError *error) {
        // do something with the error
}];
```

The values of `Amount` and `Currency` will be validated during the checkout process. So you have to ensure that you are using valid data.
The currency is the currency you want to use you for the payment. You have to use three characters for currency code (DIN/ISO 4217), e.g. “EUR”.
The amount is the lowest unit of the currency you are using. That means if you are using EUR as currency, the amount needs to be in cents, e.g. an amount of 100 is 1 EUR.

`URLSuccess` and `URLFailure` are the URL’s to which the SDK redirect the status of a payment process. These URL’s normally point to a HTML site on your merchant backend to show the status to the user of your app.

Attention
Currently iOS (especially the mobile Safari browser which is used by the SDK) doesn't support the execution of requests against a HTTP/2 backend which uses Let’s Encrypt SSL certificates. For more infromation see
https://community.letsencrypt.org/t/letsencrypt-cert-not-working-for-safari-with-http-2/25576/5.

That means, when your merchant backend uses HTTP/2 (e.g. NodeJS) in combination with a Let's Encrypt certificate to show the success or failure status, this SDK is not able to do that.
You should disable HTTP/2 or use another SSL certificate on your merchant backend.

For the rest of the parameters you should take a look into the Paygate documentation. Some of the parameters need to be defined by you and are mandatory.

Now you know the available payment methods and you have configured them. It's time to show the payment methods to the user.

4.3.3 Show Payment Methods to the user

The `CMPPaymentMethod` object is a value object retrieved from the SDK with the above described method. It contains all vital information regarding a payment method needed from the SDK to show the respective payment form and complete a transaction. In addition it contains a *localizedDescription* and an *image*.

In the demo project, the payment methods are presented in a tableView, including ApplePay. In the following code snippet is demonstrated the population of a cell, providing a title and an image of the respective payment method.

```
PaymentMethodTableViewCell * cell = [self.tableView
dequeueReusableCellWithIdentifier:@"PaymentMethodTableViewCell" forIndexPath:indexPath];

CMPPaymentMethod *paymentMethod = [[self paymentMethods] objectAtIndex:indexPath.row];

cell.labelTitle.text = paymentMethod.localizedDescription;
[cell.paymentImageView setImage:paymentMethod.image];
```

4.3.4 Checkout

After the configuration of the SDK is completed and all necessary payment data are imported, you can proceed with the checkout for a selected payment method.

Start the checkout by instantiating a `CMPCheckout` object. The `CMPCheckout` class is a top-level class that facilitates the payment procedure. It is responsible for validating payment data and instantiating a `CMPCheckoutViewController` object when a new payment is triggered by passing the respective `paymentMethod` including the `paymentData`.

```
CMPCheckout *checkout = [[CMPCheckout alloc] init];
```

Proceed with the checkout by presenting a `CMPCheckoutViewController` which is a subclass of UIViewController encapsulating all the views' stack.

```
CMPPaymentMethod *paymentMethod = [[self paymentMethods] objectAtIndex:indexPath.row];

[self.checkout instantiateCheckoutViewControllerWithPaymentMethod:paymentMethod
                                onSuccess:^(CMPCheckoutViewController *checkoutViewController) {
                                   checkoutViewController.delegate = self;
                                   // show the new chekoutViewController
                                   // maybe push it on navigation stack
                                 } onFailure:^(NSError *error) {
                                   // handle the error
                                 }];
```

Receive results from checkout by conforming to `CMPCheckoutViewControllerDelegate` and implementing its methods:

```
- (void)checkoutDidAuthorizePaymentForPaymentData:(id<CMPPaymentDataProtocol>)paymentData
withResponse:(CMPPaymentRespose *)response

- (void)checkoutDidFailToAuthorizePaymentForPaymentData:(id<CMPPaymentDataProtocol>)paymentData
withError:(NSError *)error withResponse:(CMPPaymentRespose *)response

- (void)checkoutDidCancelForPaymentData:(CMPPaymentData *)paymentData;
```

4.4 Apple Pay

4.4.1 Configuration

4.4.1.1 Apple Pay Developer page

- Enable "Apple Pay" service for the appID.
- Create "Merchant ID" (under "Identifiers"). It is recommended to use reverse domain style for the format of the "Merchant ID" that starts with merchant (i.e. for bundleID: "com.exozet.ComputopDemo" the merchantID should be something like "merchant.com.exozet.ComputopDemo").
    - This "Merchant ID" is passed to SDK as merchantAppleID parameter

- Create a certificate for the “Merchant ID” by uploading the CSR provided by Computop and send the certificate back to Computop. (under “Identifiers/Merchant ID” select MerchantID & click edit, then create Payment Processing Certificate by uploading the CSR provided by Computop). Please contact helpdesk@computop.com to obtain a CSR.
- For more information, please see Computop Apple Pay documentation

**If you see a warning in Keychain Access that the certificate was signed by an unknown authority or that it has an invalid issuer, make sure you have the WWDR intermediate certificate - G2 and the Apple Root CA - G2 installed in your keychain. You can download them from apple.com/certificateauthority.*

4.4.1.2 Xcode project

- To enable Apple Pay for your app in XCode, open the Capabilities pane. Select the switch in the Apple Pay row, and then select the merchant ID you want the app to use.
- Insert “`Merchant ID`” in the AppDelegate.

```
#import "AppDelegate.h"
#import <Computop.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
     CMPConfiguration.merchantAppleID = @"YOUR_MERCHANT_ID";

    return YES;
}

@end
```

4.4.1.3 Testing Apple Pay Transactions

Use the Apple Pay Sandbox environment to test your transactions with test payment cards.

- In iTunes Connect, create a test account. This account works for both App Store and Apple Pay testing.
- On a valid test device, log into iCloud using the test account.
- In the Wallet app, add a new card using manual entry.

4.4.2 Usage

4.4.2.1 Payment data

As described in 4.3.2 you should have configured the Apple pay payment data parameters.

4.4.2.2 Present PKPaymentAuthorizationViewController

Start Apple Pay by instantiating a `CMPApplePay` object. The `CMPApplePay` class is a top-level class that facilitates the Apple Pay payment procedure. It is responsible for validating payment data and instantiating a `PKPaymentAuthorizationViewController` object when a new payment is triggered by passing the respective `paymentData` and `paymentMethod`.

```
CMPApplePay *applePay = [[CMPApplePay alloc] init];
applePay.delegate = self;
```

Setup `PKPaymentSummaryItem` objects:

```
PKPaymentSummaryItem *paymentSummaryItem1 = [[PKPaymentSummaryItem alloc] init];
paymentSummaryItem1.label  = @"SUMMARY_ITEM_1_LABEL";
paymentSummaryItem1.amount = 'SUMMARY_ITEM_1_AMOUNT';

PKPaymentSummaryItem *paymentSummaryItem2 = [[PKPaymentSummaryItem alloc] init];
paymentSummaryItem2.label  = @"SUMMARY_ITEM_2_LABEL";
paymentSummaryItem2.amount = 'SUMMARY_ITEM_2_AMOUNT';

PKPaymentSummaryItem *paymentSummaryItemTotal = [[PKPaymentSummaryItem alloc] init];
paymentSummaryItemTotal.label  = @"Total";
paymentSummaryItemTotal.amount = 'TOTAL_AMOUNT';
```

Present the `PKPaymentAuthorizationViewController`:

```
CMPPaymentMethod *applePayPaymentMethod = [[Computop sharedInstance] paymentMethodForID:
@"pm_applepay"];

NSArray* supportedNetworks = @[PKPaymentNetworkVisa, PKPaymentNetworkMasterCard,
PKPaymentNetworkAmex, PKPaymentNetworkDiscover];


[self.applePay
instantiatePKPaymentAuthorizationViewControllerWithPaymentMethod:self.paymentData
            withPaymentSummaryItems: /*your array with PaymentSummaryItems*/
              withSupportedNetworks:supportedNetworks
  withRequiredShippingAddressFields:self.selectedPKShipping
 paymentAuthorizationViewController:^(PKPaymentAuthorizationViewController
                                *applePayViewController) {
                                      // show apples pay view controller
                                      [self presentViewController:applePayViewController
                                                         animated:true
                                                       completion:nil];
                                 } onFailure:^(NSError *error) {
                                       // handle the error
                                 }];
```

CMPApplePayDelegate

Implement `CMPApplePayDelegate` protocol's methods in order to get notified of navigation actions and on ApplePay payment's result:

```
- (void)applePayDidDismiss {
}

- (void)applePayDidAuthorizePaymentForPaymentData:(id<CMPPaymentDataProtocol>)paymentData
withResponse:(CMPPaymentRespose *)response {
}

-
(void)applePayDidFailToAuthorizePaymentForPaymentData:(id<CMPPaymentDataProtocol>)paymentData
withError:(NSError *)error withResponse:(CMPPaymentRespose *)response {
}

- (void)applePayPaymentDidSelectPaymentMethod:(PKPaymentMethod *)paymentMethod
completion:(void (^)(NSArray<PKPaymentSummaryItem *> *))completion {
}

- (void)applePayPaymentDidSelectShippingContact:(PKContact *)contact completion:(void
(^)(PKPaymentAuthorizationStatus, NSArray<PKPaymentSummaryItem *> *))completion {
}
```

4.4.3 iOS Human Interface Guidelines

The UI components and interactions with ApplePay should follow Apple's iOS Human Interface Guidelines. Please read the official documentation provided by Apple.

4.5 WeChat Pay

For making payments with WeChat developer should use WeChat class with the public method: startPaymentWithPaymentData.

```
CMPWeChat *weChat = [[CMPWeChat alloc] init];

    [weChat startPaymentWithPaymentData:paymentMethod
                                success:^(NSData *data) {
    } failure:^(NSError *error) {

    }];
```

4.6 PayPal

Developer first should declare the scheme that he will use to return to the merchant App. This should be done as following:

4.6.1 Register a URL Type

1. In XCode, click on your project in the Project Navigator and navigate to App Target > Info > URL Types
2. Click [+] to add a new URL type
3. Under URL Schemes, create a URL scheme that your app will respond to. The scheme must start with your app's Bundle ID and dedicated for use with your PayPal integration.

Example custom url:
com.merchant.MerchantApp.PayPalReturn

4.6.2 Redirection from PayPal

For this payment method, the SafariViewController is used. To get back from the SafariViewController to your app after payment is done (or canceled), you have to redirect to a custom uri from your merchant backend.

The flow after payment is done/canceled:

- Computop will call your merchant backend using URLSuccess/URLFailure you specified.
- On your merchant backend you have to redirect from your URLSuccess/URLFailure to the custom uri to get back to our app, e.g.:
    - URLSuccess → com.merchant.MerchantApp.PayPalReturn://return
    - URLFailure → com.merchant.MerchantApp.PayPalReturn://cancel

It is important to use the values **return** and **cancel** as path segments for success and failure.

4.6.3 Update the AppDelegate

To respond to requests made to the custom URL type, the application openURL method must be implemented in the application delegate. The openURL method should send a notification to Computop in order to complete or cancel the PayPal checkout. The notification keys must be same as follows:

```
- (BOOL)application:(UIApplication *)application  openURL:(NSURL *)url   sourceApplication:(NSString
*)sourceApplication
        annotation:(id)annotation {
    if ([url.scheme localizedCaseInsensitiveCompare:@"com.merchant.MerchantApp.PayPalReturn"] ==
NSOrderedSame) {
        if([url.host localizedCaseInsensitiveCompare:@"return"] == NSOrderedSame) {
            NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
            [nc postNotificationName:@"completePayPalCheckout" object: self userInfo: nil];
        }
        else {
            NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
            [nc postNotificationName:@"cancelPayPalCheckout" object: self userInfo: nil];
        }
        NSLog(@"%@", url.host);
        return YES;
    }

     return NO;


}
```

Important note: Developer should add to main project *-l"PPRiskComponent"* under the Build settings → Other linker flags the Magnes library used by Computop.

Since the Magnes library requires the location to send the payload risk to PayPal developer should activate it under info.plist by adding *Privacy - Location When In Use Usage Description* and some description text.

To init PayPal and call the payment method developer should use the following code:

```
CMPayPal *payPal = [[CMPayPal alloc] init];

[payPal startPaypalPay:paymentMethod
            success:^(NSData *data) {
//Handle the succes
                    } failure:^(NSError *error) {
 //Handle the failure
                    }];
```