M1pay Integration

Change Log:

Revision:	Description:	Name	Date:
1.0	Creation of Document	Rasoul Nayebpour	March 4, 2019
1.0.1	Adding Channel types and Status types. Changing address of get transaction information.	Rasoul Nayebpour	March 13, 2019
1.0.2	Adding “Verifying data” code in Appendix 2 and refining code in Appendix 1	Rasoul Nayebpour	March 18, 2019
1.0.3	Updating the status of transaction	Amirarsalan Hassani Zadeh	Nov 17, 2020
1.0.4	Updating the FPX model	Mohsen Mohammadi	Jan 19, 2021

Table of Contents

1. Introduction

The M1Pay API offers merchant FPX and CardPayment channels to their customer across the globe with just one-time integration. M1Pay API designed to optimize and simplify integration effort between a merchant and various payment channels. Instead of having to integrate with multiple APIs from different payment channels, M1Pay API provides a single endpoint for all. In other words, overall integration effort can be reduced and thus, completed in much shorter time.

M1Pay API also provides a secure payment experience with server to server communication over a secure socket layer (SSL) and conforms to Representational State Transfer (RESTful) architectural style uses JSON as its data representation format.

2. Security Features

The M1Pay API services are protected for only authorized merchant with restricted hardened platform to secure payment data transmission.

Secure Socket Layer (SSL) data transport:

It’s required to use HTTPS for all interchange message between merchant and M1Pay. This is to prevent any sensitive data being reveal by unauthorized party during message exchange.

IP Address Filtering:

Merchant must register their server’s static IP addresses with M1Pay to establish a secure connection to M1Pay Wall Payment server.

Data Message Protection (Signature):

This is an application layer security in ensuring data integrity. All data in the message exchange will be hashed using a unique Private Key and output as Signature. Private Key is assign to merchant after account creation in merchant dashboard. In addition, merchants can regenerate private key at anytime in merchant dashboard. M1Pay will validate this Signature to prevent any data tampering during the message exchange. It’s also strongly recommended for merchant to perform the same validation for all response messages received from M1Pay.

OAuth2

This protocol allows merchant application to grant access to M1Pay APIs and protects M1Pay APIs against attacks by attackers. This protocol requires merchant code as username and Secret key as password to validate allowed merchant. M1Pay support team send merchant code and secret key by email

3. Get Started

3.1. Merchant Application Account

Before merchant start integrating with M1Pay API, M1Pay will provide account information below for merchant integration development process by sending email to the merchant.

Merchant Code:	A Unique code to identify merchant application or portal which integrating with M1Pay API.
Private Key:	A server-side shared private key which will use to generate signature for API communication.
Secret Key:	A key required by OAuth2 server to validate merchants

3.2. Setting up Merchant Application Account

After merchant obtained M1Pay API account from M1Pay, merchant needs to provide below information to M1Pay API support team to complete the setup for the account.

Callback URL

M1Pay will send HTTP POST request with the result of payment status to this merchant’s Callback URL. Each merchant application account refers to one Callback URL. The URL will be setup and maintain under the merchant console. The maximum length of Callback URL is 255 characters.

IP Address

Outgoing IP address of merchant payment server, for whitelisted purposes.

4. Payment Service

4.1 Surface Payment Methods Options

M1Pay API allows merchant to surface all supported payment methods to customer via M1Pay Payment Wall, or alternatively, merchant to Surface the Payment Methods Individually at merchant site.

- To surface payment methods via M1Pay Payment Wall (see figure 1), simply leave blank or skip the parameter {channel} during Payment Request.
- For Surface the Payment Methods Individually, merchant surface each individual M1Pay's payment methods on merchant site page (see figure 3). Once consumer selects a payment channel logo (e.g. FPX), merchant specific {channel} during Payment Request.

4.1.1 M1Pay Payment Wall

The Payment Wall surface all M1Pay supported payment methods in a M1Pay hosted page. Merchant only required surfacing ONE M1Pay logo at their payment methods selection page in order for customer to using all supported payment methods. M1Pay Payment Wall handles the displays of the payment options. There is no change required on merchant site when new payment methods added by M1Pay.

Below figure depicts a sample of M1Pay Payment Wall screen:

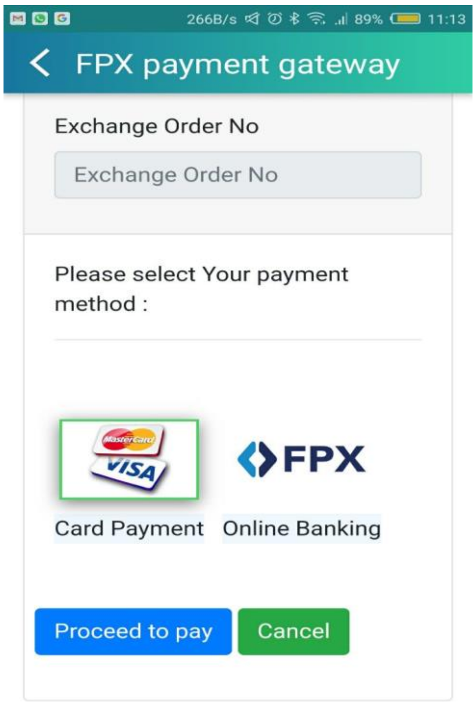

Figure 1: Sample M1Pay Wall Screen

4.2 Merchant Integration Steps

Sending request to M1Pay payment wall and getting the result by the merchant, requires four steps.

4.2.1 Calling M1Pay OAuth2 API

In step one merchant needs to send merchant code and secret key to M1Pay OAuth2 server to validate and if they allow to access M1Pay APIs, M1Pay OAuth2 server will return a token back to merchant. This token is required for next step.

Environment	**Service URL**
Sandbox	https://keycloak.m1pay.com.my/auth/realms/master/protocol/openid-connect/token
Production	https://keycloak.m1pay.com.my/auth/realms/m1pay-users/protocol/openid-connect/token

Request Header	
HTTP Method	POST
Content type	x-www-form-urlencoded

Request Body Message:

Parameters	**Description**
grant_type	“client_credentials”
client_id	Merchant code as described in section 3.1
client_secret	Secret key as described in section 3.1

Response Body Message:

Parameters	**Description**
access_token	A token required for further step

4.2.2 Calling M1Pay Payment Wall API, Create Transaction

In step two, merchant should send access_token received in step one to initiate a payment request:

Environment	Service URL
Sandbox	https://gateway-uat.m1pay.com.my/m1paywall/api/transaction
Production	https://gateway.m1pay.com.my/wall/api/transaction

Request Header	
HTTP Method	POST
Content type	JSON format
Authorization	"Bearer " + token received from step one

Request Body Parameters:

Parameters	Description
transactionAmount	The price of product or service that merchant wants to sell
merchantId	Merchant code as described in section 3.1
transactionCurrency	Currently it needs constant value "MYR", but it will change in next releases
merchantOrderNo	Optional unique value to keep track of transaction
exchangeOrderNo	Extra optional unique value to keep track of transaction
productDescription	Description about product or service that merchant wants to sell, Limited to 30 characters and the following characters are allowed: a-z A-Z 0-9 . -
fpxBank	Constant value 1 to choose bank on our FPX confirmation page, To skip FPX confirmation page and get redirected to bank page, you need to pass valid bankId as fpxBank value. You may get the bank list from the following APIs: UAT: https://gateway.m1payall.com/m1payfpx/api/bank-list/B2C Prod: https://gateway.m1pay.com.my/fpx/api/bank-list/B2C
emailAddress	Optional email address of customer
signedData	Signing data by private key provided by M1Pay. For more information refer to Appc
channel	Selected channel by customer. (channel values are: - ONLINE_BANKING - CARD_PAYMENT - UMOBILE - EMONEI - ALIPAY)
phoneNumber	Phone number of the customer (Mandatory only for UMOBILE channel)
skipConfirmation	Optional Boolean value to skip payment channel confirmation page.(currently applicable for Alipay channel)

Response Body Parameters:

Parameters	**Description**
Location	Address of M1Pay confirmation page that will show transaction information. The merchant needs to redirect to whatever returning back from server.
transactionId	Unique identifier generated by M1Pay wall to keep track of transaction
token	One time token that needs to send for getting transaction information. It will expire after each use.

4.2.3 Calling M1Pay Payment Wall API, Get Transaction Information

By calling this API, merchant could access transaction information and status:

Environment	**Service URL**
Sandbox	https://gateway-uat.m1pay.com.my/m1paywall/api/m-1-pay-transactions/{transactionID}
Production	https://gateway.m1pay.com.my/wall/api/m-1-pay-transactions/{transactiondID}

Request Header	
HTTP Method	GET
Content type	
Authorization	"Bearer " + token received from step one

Request Parameters:

Path Variables	**Description**
transactionId	Received from previous step

Response Body Parameters:

Parameters	**Description**
transactionAmount	The price of product or service that merchant wants to sell
merchantId	Merchant code as described in section 3.1
transactionCurrency	Currently it needs constant value “MYR”, but it will change in next releases
merchantOrderNo	An optional unique value to keep track of transaction
exchangeOrderNo	Extra optional unique value to keep track of transaction
ProductDescription	Description about product or service that merchant wants to sell
fpxBank	Currently it needs constant value 1, but it will change in next releases
emailAddress	Optional email address of customer
transactionStatus	The last status of transaction.
model	B2B1 or B2C (Only applicable for FPX channel)

4.2.4 Calling Merchant Callback URL

At final step, M1Pay will send the result of transaction back to the merchant. So merchant has to provide a callback API to be called by M1Pay and inform it’s address to M1Pay support team by email. This API should have below specifications:

Request Header	
HTTP Method	POST
Content type	X-www-form-urlencoded

Request Body Parameters:

Parameters	**Description**
transactionAmount	The price of product or service that merchant wants to sell
fpxTxnId	A unique identifier generated and returned by FPX
merchantOrderNo	Optional unique value to keep track of transaction
status	Status of transaction. (Status values are: REQUEST, APPROVED, ROLLBACK, UNSUCCESSFUL, PENDING, CANCELLED, FAILED, CAPTURED, SUCCESSFUL, COMPLETED, CANCEL, COMPLETED_ACK) The followings are the success values of different channels: APPROVED: FPX CAPTURED: CARD_PAYMENT SUCCESSFUL: EMONEI, ALIPAY COMPLETED: BOOST SUCCESSFUL: UMOBILE
sellerOrderNo	Unique identifier generated by M1Pay
description	In case of any error during these steps, it contains additional description.
signedData	For verifying data at merchant side, it is needed to concat data as below and then verify it with signedData: transactionAmount + "|" + fpxTxnId + "|" + sellerOrderNo + "|" + status + "|" + merchantOrderNo

Appendix 1: Signing data in merchant host

There are two steps in signing data. The first one is concatenation data and the second one is signing data by using private key that M1Pay provides for merchant.

Merchant needs to merge following information to be signed:

```
StringBuilder sb = new StringBuilder();
    sb.append(this.productDescription).append("|");
    sb.append(this.transactionAmount).append("|");
    sb.append(this.exchangeOrderNo).append("|");
    sb.append(this.merchantOrderNo).append("|");
    sb.append(this.transactionCurrency).append("|");
    sb.append(this.emailAddress).append("|");
    sb.append(this.getMerchantId());
```

transactionAmount must have precision. For example if transactionAmount is 10, it should be concatenated as 10.00

Here is a sample code for signing data in java language. But, merchant could implement it using any programming language:

```
static char[] hexChar = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
    'A', 'B', 'C', 'D', 'E', 'F'};


  public static String byteArrayToHexString(byte b[]) {
    StringBuffer sb = new StringBuffer(b.length * 2);
    for (int i = 0; i < b.length; i++) {
      sb.append(hexChar[(b[i] & 0xf0) >>> 4]);
      sb.append(hexChar[b[i] & 0x0f]);
    }
    return sb.toString();
  }
public static String signData(String pvtKeyFileName, String dataToSign)  {
    PEMReader pemReader = new PEMReader(new StringReader(pvtKeyFileName));
```

```
        KeyPair pair = (KeyPair) pemReader.readObject();
        PrivateKey privateKey = pair.getPrivate();
        Signature signature = Signature.getInstance("SHA1withRSA", "BC");
        signature.initSign(privateKey);
        signature.update(dataToSign.getBytes());
        byte[] signatureBytes = signature.sign();
        return byteArrayToHexString(signatureBytes);
    }
```

The first parameter of this method is private key file name and path and the second one is concatenation data produced in previous step.

PHP sample code for signing data:

```
<?php
$data = ‘concat data’;
$file_name = '/home/rasoul/merchantXXXXXXXX.key'; //Path of private key
$signature = '';            //Signed data will be store in this parameter
try {
        $myfile = fopen($file_name, "r") or die("Unable to open file!");
        $priv_key = fread($myfile,filesize($file_name));
        fclose($myfile);
        $pkeyid = openssl_get_privatekey($priv_key);
        openssl_sign($data, $signature, $pkeyid, "sha1WithRSAEncryption");
        echo strToHex($signature);
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}
function strToHex($string){
    $hex = '';
    for ($i=0; $i<strlen($string); $i++){
        $ord = ord($string[$i]);
        $hexCode = dechex($ord);
        $hex .= substr('0'.$hexCode, -2);
    }
    return strToUpper($hex);
}
?>
```

.Net C# sample:

```
static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("merchantXXXXXX.key");
            PemReader pr = new PemReader(sr);
            AsymmetricCipherKeyPair KeyPair = (AsymmetricCipherKeyPair)pr.ReadObject();
            ISigner sig = SignerUtilities.GetSigner("SHA1withRSA");
            sig.Init(true, KeyPair.Private);
            var bytes = Encoding.UTF8.GetBytes("ConcatenatedData");
            sig.BlockUpdate(bytes, 0, bytes.Length);
            byte[] signature = sig.GenerateSignature();
            Console.WriteLine(ByteArrayToString(signature));
}

public static string ByteArrayToString(byte[] ba)
        {
                StringBuilder hex = new StringBuilder(ba.Length * 2);
                foreach (byte b in ba)
                        hex.AppendFormat("{0:x2}", b);
                return hex.ToString();
        }
```

Appendix 2: Verifying data

```
public static Boolean verifyData(String pubKeyFileName, String calcCheckSum, String
checkSumFromMsg)  {
        log.info("Entering verifyData method");
        boolean result;
        try {
            InputStream inStream = new FileInputStream(pubKeyFileName);
            log.debug("inStream: {}", inStream.toString());
            CertificateFactory certFactory = CertificateFactory
                .getInstance("X.509");
            log.debug("certFactory: {}", certFactory.toString());
            X509Certificate cert = (X509Certificate) certFactory
                .generateCertificate(inStream);
            log.debug("cert: {}", certFactory.toString());
            inStream.close();
            PublicKey pubKey = (RSAPublicKey) cert.getPublicKey();
            Signature verifier = Signature.getInstance("SHA1withRSA", "BC");
            verifier.initVerify(pubKey);
            verifier.update(calcCheckSum.getBytes());
            result = verifier.verify(HexStringToByteArray(checkSumFromMsg));
            log.debug("result of verifier.verify [{}]", result);
            if (result)
                return true;
            else {
                log.error("Your Data cannot be verified against the Signature. ErrorCode :[09]");
                return false;
            }
        } catch (Exception e) {
            log.error("ErrorCode : [03]" + e.getMessage());
            return false;
        } finally {
            cerExpiryCount = 0;
        }
    }
```

PHP sample code for verifying data:

```
<?php
$data = ‘concat data’;
$file_name = '/home/rasoul/merchantXXXXXXXX.crt';     //Path of public key
$signature = ‘signed data’;
$signature = hexToStr($signature);
try {


        $myfile = fopen($file_name, "r") or die("Unable to open file!");
        $pub_key = fread($myfile,filesize($file_name));
        fclose($myfile);


        $pubkeyid = openssl_get_publickey($pub_key);


        $r = openssl_verify($data, $signature, $pubkeyid, "sha1WithRSAEncryption");
        var_dump($r);
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}
function hexToStr($hex){
    $string='';
    for ($i=0; $i < strlen($hex)-1; $i+=2){
        $string .= chr(hexdec($hex[$i].$hex[$i+1]));
    }
    return $string;
}
?>
```

.Net C# sample:

```
static void Main(string[] args)
        {
                FileStream fileStream = new FileStream("merchantXXXXXXXX.crt", FileMode.Open);
                X509Certificate cert = new X509CertificateParser().ReadCertificate(fileStream);
            ISigner signerVerify = SignerUtilities.GetSigner("SHA1withRSA");
            signerVerify.Init(false, cert.GetPublicKey());
            byte[] signBytes = FromHex("Signed Data");
            byte[] dataBytes = Encoding.UTF8.GetBytes("concat data");
            signerVerify.BlockUpdate(dataBytes, 0, dataBytes.Length);
            Console.WriteLine(signerVerify.VerifySignature(signBytes));
        }

public static byte[] FromHex(string hex)
        {
                hex = hex.Replace("-", "");
                byte[] raw = new byte[hex.Length / 2];
                for (int i = 0; i < raw.Length; i++)
                {
                        raw[i] = Convert.ToByte(hex.Substring(i * 2, 2), 16);
                }
                return raw;
        }
```

Appendix 3: Sample web integration SDK

There are a sample web SDK contains an html and a JS. JS address is:

https://m1pay.com.my/integration-sdk/js/m1paywebsdk.js

and html address is:

https://m1pay.com.my/integration-sdk/