July 24, 2026

Google Pay™

INTEGRATION GUIDE

VERSION 1.0

Statement of Confidentiality

Table of Contents

1. Google Pay™ Merchant Hosted (Seamless) Integration

Merchant Side Activities

Step 1 : Integration of merchant website with Google Pay™

Merchant needs to integrate the Google Pay™ Web Checkout solution to their website and host Google Pay™ button. To implement this merchants can refer the official documentation that Google Pay™ has provided :

- Google Pay Web developer documentation
- Google Pay Web integration checklist
- Google Pay Web Brand Guidelines

Step 2: Before proceeding with integration, please review these Google Pay™ documents:

You must additionally complete the following essential steps to enable Google Pay™ functionality:

- Terms of Service
- Acceptable Use Policy

Step 3: Gateway and GatewayMerchantID :

When you integrate with the Google Pay™ API as a merchant, ensure to set the following values in the TokenizationSpecification object:
gateway : **'bankmuscat'**
gatewayMerchantID : '**Bank Muscat provided SmartPay Registration ID**'

Here's an API configuration example to show merchants, if your gateway value is set to Google and your gatewayMerchantID value is found in your merchant's account with you:

```
const tokenizationSpecification = {
        type: 'PAYMENT_GATEWAY',
        parameters: {
                'gateway': 'bankmuscat',
                'gatewayMerchantId': 'Bank Muscat provided SmartPay Registration ID'
                }
        };
```

Step 4 : Authorization Methods

3D Secure (3DS) enhances payment security and may be required depending on the card type used with Google Pay™.

Google Pay™ provides two different authorization methods: PAN_ONLY and CRYPTOGRAM_3DS,

Our integration with Google Pay™ supports both types :

- **PAN_ONLY :** Physical card details stored in Google Pay™
  3DS Required : Yes, standard 3DS flow applies
  Implementation : Handle 3DS challenge flow when indicated by the API
  Supported country : Oman
- **CRYPTOGRAM_3DS :** Tokenized virtual card stored on device
  3DS Required : No, authentication is performed by Google Pay™
  Implementation : No additional steps needed
  Supported country : Oman

Handling 3DS for PAN_ONLY Cards :
When processing a PAN_ONLY card through Google Pay™, you may need to handle 3DS authentication:

Bank Hosted Payment Page: 3DS is automatically handled
Merchant Hosted Payment Page: 3DS is automatically handled, merchant just need to pass encrypted payload to Bank Muscat SmartRoute Application request as mentioned in Request Post parameter.

Step 5 : Configuration allowed Cards and Payment methods :

we support VISA, Mastercard and American Express networks with the Google Pay™ API.
You can define these values in the allowedCardNetworks property and find the appropriate values in Google Pay's web developer documentation.

Here's an API configuration example to show merchants,

```
const allowedCardNetworks = ["AMEX", "MASTERCARD", "VISA"];
const allowedCardAuthMethods = ["PAN_ONLY", "CRYPTOGRAM_3DS"];
```

Step 6 : Button Implementation for Google Pay™

```
// Load Google Pay API
const googlePayClient = new google.payments.api.PaymentsClient({environment: 'PRODUCTION' // or
                                    'TEST' for sandbox});

// Create button
const button = googlePayClient.createButton({
        buttonType: 'pay',
        buttonColor: ‘black’,
        onClick: onGooglePayButtonClicked,
        allowedPaymentMethods: [baseCardPaymentMethod]
        });
document.getElementById('google-pay-container').appendChild(button);
```

Step 7 : Create PaymentDataRequest object includes details

- Build a JavaScript object that describes your site's support for the Google Pay™API :

```
const paymentDataRequest = Object.assign({}, baseRequest);
```

- Add the payment methods supported by your app, such as any configuration of additional data that's expected in the response :

```
paymentDataRequest.allowedPaymentMethods = [cardPaymentMethod];
```

- Define a total price and currency for a shopper to authorize :

```
paymentDataRequest.transactionInfo = {
        totalPriceStatus: 'FINAL',
        totalPrice: '123.45',
        currencyCode: 'OMR',
        countryCode: 'OM'
    };
```

- Provide a user-visible merchant name, and merchantId value when in merchantInfo :

```
paymentDataRequest.merchantInfo = {
    merchantName: 'Example Merchant'
    merchantId: '12345678901234567890'
};
```

Step 8 : Billing Address parameters details

if a billing address is required for Google Pay™, ensure that you include required parameters. For more details, see `BillingAddressParameters`.

The Bank Muscat Smartroute application does not require billing address information for Google Pay™ transactions; therefore, these parameters are not sent in the Google Pay™ request parameters.

Step 9 : Payment Processing for Google Pay™

```
paymentsClient.loadPaymentData(paymentDataRequest).then(function(paymentData){

    const token = paymentData.paymentMethodData.tokenizationData.token;
    const cardNetwork = paymentData.paymentMethodData.info.cardNetwork;
    const cardFundingSource = paymentData.paymentMethodData.info.cardFundingSource;

}).catch(function(err){
    console.error(err);
});
```

Add above values as follows in the request to Bank Muscat,

token = googlePayEncData,
cardNetwork = cardName,
cardFundingSource = cardType.

Step 10 : Google Pay™ payload

Once Google Pay™ button is clicked, user will receive notification to approve the transaction request from Google Pay™ wallet. once user accepts the request then merchant will get JSON payload from Google Pay™, sample of JSON payload is as below :


```
{
  "signature":"MEUCIAqB**********nBU",
  "intermediateSigningKey":{
    "signedKey":{
      "keyValue":"Mfkw**********SqWTTxncA",
      "keyExpiration":"1755****16205"
    },
    "signatures":["MEUCIQDmhh86OF8**********XbbNk"]
  },
  "protocolVersion":"Ecv2",
  "signedMessage":{
    "encryptedMessage":"9Nmjz**********ppMdywimqq5mqt",
    "ephemeralPublicKey":"BcwXKYNt**********LjsarRAyao",
    "tag":"2Kw3nFKYJx**********i5DnfD5l8"
  }
}
```


Merchant needs to send that payload in below mentioned parameters along with other parameters listed in below table.

Step 11 : Submitting the Transaction Request to Bank Muscat (Sending Encrypted data to SmartPay application)

Once you have followed the above steps and have the Google Pay™ button hosted on your page, you are ready to integrate with SmartPay application for performing the Google Pay™ transaction.

You need to make a call to SmartPay application on below URL along with encRequest and access_code parameters.

You need to use AES encryption method to encrypt the below listed parameters and include them in the **encRequest** parameter.

Transaction Request formation:

Each request parameter name and its value should be delimited using ampersand (&). Following is sample for request string creation with mandatory parameters:

- ### STEP 1: String Formation

merchant_id=12345&**order_id**=TR261A173&**currency**=OMR&**amount**=1.123&**redirect_url**=&**payment_option**=OPTGPAY&**cardName**=Visa&**cardType**=Credit&**googlePayEncData**={"signature":"MEUCIAqB**********nBU","intermediateSigningKey":{"signedKey":{"keyValue":"Mfkw**********SqWTTxncA","keyExpiration":"1755****16205"},"signatures":["MEUCIQDmhh86OF8**********XbbNk"]},"protocolVersion":"Ecv2","signedMessage":{"encryptedMessage":"9Nmjz**********ppMdywimqq5mqt","ephemeralPublicKey":"BcwXKYNt**********LjsarRAyao","tag":"2Kw3nFKYJx**********i5DnfD5l8"}}

- ### STEP 2 : String Encryption

encRequest = AES encryption of string generated in STEP 1 using working key shared against the Registration ID (reg_id) post onboarding. AES Encryption and Decryption detail is mentioned in Appendices.

- ### STEP 3 : Payment Initiation

Upon successful request encryption, following two parameters should be POSTED to Smart Pay application:

1) access_code
2) encRequest

Perform the test transactions to make sure everything works. As soon as the complete testing is done, merchant can move to production environment.

Test Environment Pointing URL:
https://spayuattrns.bmtest.om/transaction.do?command=initiateTransaction

PROD Environment Pointing URL:
https://smartpaytrns.bankmuscat.com/transaction.do?command=initiateTransaction

Complete URL structure to be sent to Bank Muscat Samrtroute Application :

https://smartpaytrns.bankmuscat.com/transaction.do?command=initiateTransaction&access_code=ANDS*****KXSQ&encRequest=31E4C8727B93A8B258C91308513C……..23EE151D175BF98795EEF460D453967DCFD64F983B9CC2AC62AD5CF07B6FEF4962093365766CC1D552FB

Request Parameter Details:

Merchant must send the following parameters for processing an order.

Parameter Name	Description	Type
merchant_id	Merchant ID is a unique identifier generated by SmartPay application for each activated merchant.	NUMERIC
order_id	This ID is used by merchants to identify the unique order. Ensure that you send a unique id with each request. SmartPay application will not check the uniqueness of the order id as it generates a unique payment reference number for each order sent by the merchant.	ALPHANUMERIC (30) Characters allowed: Alphabet (A-Z), (a-z), Numbers,- (hyphen), / (slash), ,_ (underscore)
currency	Currency for processing the transaction. OMR – Omani Riyal	Alphabets (3) OMR
amount	Order amount	NUMERIC (10,3)
redirect_url	SmartPay will post the status of the order along with the other response parameters to this URL. If you do not send this value, order status will be sent back to the URL configured in dynamic event notifications module in your SmartPay MARS account. If there is no URL configured in SmartPay MARS account, PG will display the status of the order on the SmartPay confirmation page.	ALPHANUMERIC (100)
payment_option	Set value as OPTGPAY	Static Value (08) OPTGPAY
cardName	paymentData.paymentMethodData.info.cardNetwork :- Card Name received from Google Pay™ payments data Sample :- Mastercard / Visa	varchar(100)
cardType	paymentData.paymentMethodData.info.cardFundingSource :- Type of card used by the customer recived from Google Pay™ payments data. **Sample :-** Credit and Debit	Alphabets (10)

googlePayEncData	paymentMethodData.tokenizationData.token :-
	Token received from Google Pay™ payment data
	Sample data mentioned below:-
	{
	"signature":"MEUCIAqB**********
	**********nBU",
	"intermediateSigningKey":{
	"signedKey":{
	"keyValue":"Mfkw**********SqWTTxncA",
	"keyExpiration":"1755****16205"
	},	

	"signatures": ["MEUCIQDmhh86OF8******************XbbNk"] }, "protocolVersion":"Ecv2", "signedMessage":{ "encryptedMessage":"9Nmjz******************************ppMdywimqq5mqt", "ephemeralPublicKey":"BcwXKYNt**********LjsarRAyao", "tag":"2Kw3nFKYJx**********i5DnfD5l8" } }	

Note: **googlePayEncData** this parameter should be encoded by URLEncoder (UTF-8) before sending encrypted request to SmartPay.

Billing Details -Merchant can send any of the following parameters in addition to the required parameters listed below to the Bank Muscat SmartPay Application.

Required Parameters:

Parameter Name	Description	Type
billing_name	Name of the customer	Alphabets (60) Characters allowed: Alphabet (A-Z), (a-z). Space in between words.
billing_address	Customer's billing address	Alphanumeric (150) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen) Space in between words.
billing_city	Customer's billing city	Alphabets (30) Characters allowed: Alphabet (A-Z), (a-z). Space in between words.
billing_state	Customer's billing state	Alphabets (30) Characters allowed: Alphabet (A-Z), (a-z). Space in between words.
billing_zip	Customer's billing zip code	Alphanumeric (15) Characters allowed: Alphabet (A-Z), (a-z). Numbers
billing_country	Customer's billing country	Alphabets (50) Characters allowed: Alphabet (A-Z), (a-z).

		Space in between words.
billing_tel	Customer's phone number	Numeric (20)
billing_email	Customer's email address	Alphanumeric (70) Characters allowed: Alphabet (A-Z), (a-z). Numbers @ (at), dot,_ (underscore)
merchant_param1	Additional merchant defined reference number, if any	Alphanumeric(100) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen)
merchant_param2	Additional merchant defined reference number, if any	Alphanumeric(100) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen)
merchant_param3	Additional merchant defined reference number, if any	Alphanumeric(100) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen)
merchant_param4	Additional merchant defined reference number, if any	Alphanumeric(100) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen)
merchant_param5	Additional merchant defined reference number, if any	Alphanumeric(100) Characters allowed: Alphabet (A-Z), (a-z). Numbers # (hash), Comma, circular brackets, /(slash), dot, - (hyphen)
promo_code	This parameter is used to send the promotion code created in SmartPay MARS., through which the merchant can offer specific discounts to customers using particular payment options.	Alphanumeric(15)

2. Bank Hosted Google Pay™ Transaction Flow

A. Following is brief description of Google Pay™ transaction flow in 'Bank Muscat SmartRoute application':

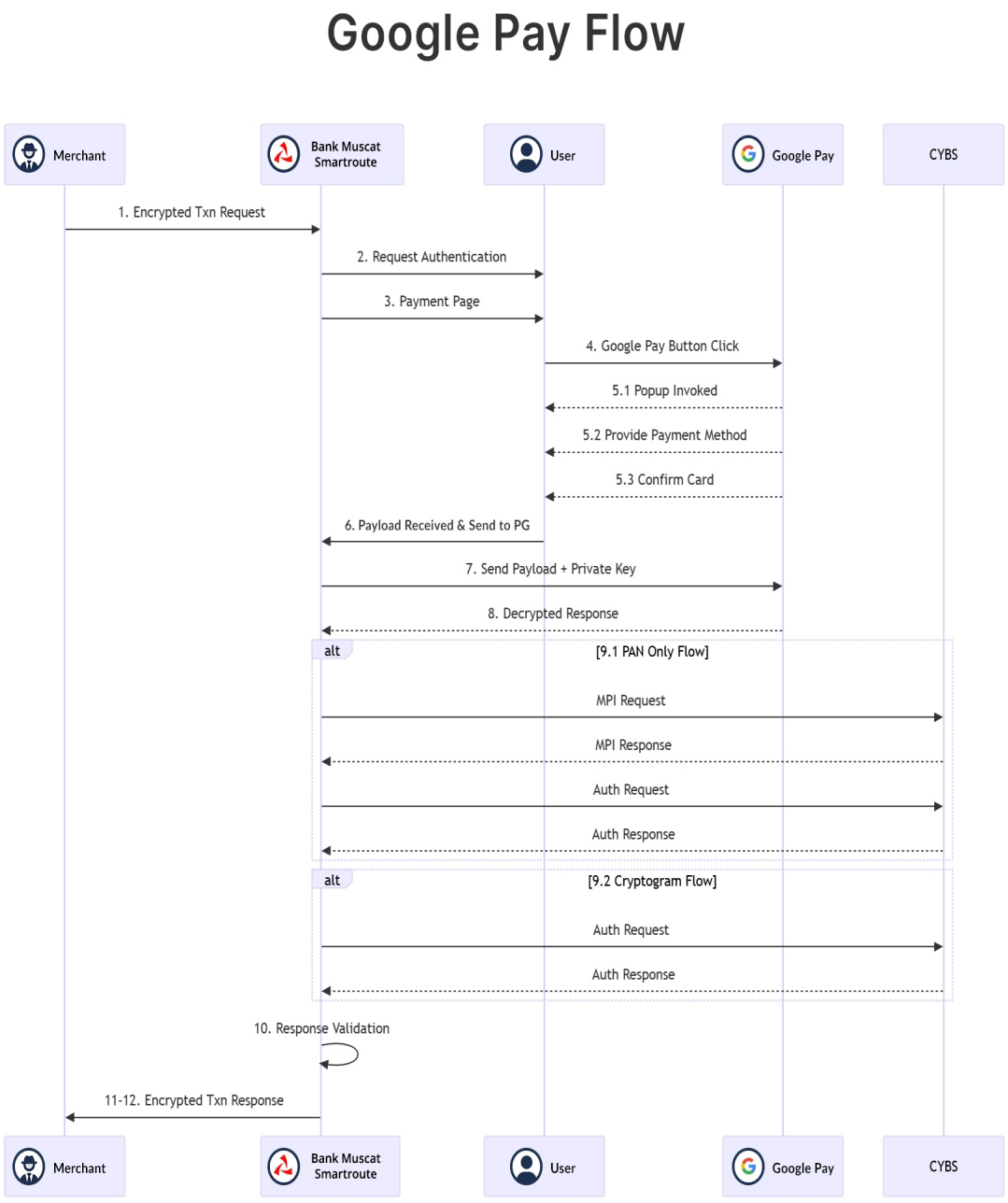
- Merchant sends an encrypted transaction request along with access code to the 'Bank Muscat SmartRoute application'.
- 'Bank Muscat SmartRoute application' authenticates the request on basis of access code and decrypts the transaction request followed by parameter level validation.
- Post successful validation, payment page is displayed w*here the* Google Pay™ *button hosted user has to click that button.*
- After clicking on Google Pay™ Button, Google populate the Google Pay™ payment sheet with the user saved cards and additional details.

a. User select a payment method and confirm the details.

b. Google Pay™ encrypt the payload using public key.

- Google Pay™ sends the encrypted payload to payment gateway with additional details.
- PG application sends this payload and private key to google for decrypt.
- Decrypted response from Google Pay™ sends to PG application.
- Google Pay™ transaction with additional details, transaction route via Cybersource.
- Response from PG connectors such as 'Cybersource' are validated and saved in 'Bank Muscat SmartRoute application'.
- 'Bank Muscat SmartRoute application' massages the response as per merchant integration and encrypts the same using the shared symmetric encryption key. Finally, the encrypted response is dispatched on merchant's Redirect URL.

B. Following diagram depicts flow of a Google Pay™ transaction :



3. AES Encryption and Decryption

Smart Pay Application uses AES 256 Encryption (AES-256-GCM) for request/response message exchange. To enable easy integration, Smart Pay provides encryption/decryption library files for following languages:

a) Java
b) PHP
c) Python

In case if merchant is using any other programming language, AES Encryption can be developed on basis of following information:

a) Algorithm Used: AES 256/GCM/No Padding
b) Initialization Vector (**IV**): 16 bytes
c) Tag Length: 16 bytes

Following is a sample step-wise output:

i. Assuming Working Key as WK = "secretKey12345**********12345"

ii. Create Initialization Vector (IV) of random 16 bytes.

iii. Generate secret key by converting Working Key (in step i) into bytes.

iv. Get Cipher instance of as AES 256 GCM with No Padding (RAW).
v. Generate encrypted value of Plain Text.

vi. Convert the encrypted value to Hex and ensure TAG value is incorporated. i.e. byteToHex(CIPHER+TAG). Some programming languages like Java by default append TAG value so, in such cases directly convert the cipher text to hex value.

vii. Append the Hex value of IV create on Step ii.

viii. Pass the encrypted value to Smart Pay Application. So, the encrypted value should be like following:

ENC_REQUEST = Hex(IV) + Hex(Cipher+Tag)

ix. Following is sample step wise output for help in programming

PLAIN_TEXT = "Hello World"
WORKING_KEY = "secretKey12345**********12345"
RANDOM_IV_HEX = "d99c5**********482c"

CIPHERTEXT (with Tag) HEX VALUE =

1d0476a6**********99899cd

ENC_REQUEST (IV HEX VALUE + CIPHERTEXT HEX VALUE):
d99c56f**********be9c4499899cd

Similarly, for decryption extract first 32 characters of encrypted string.

Convert the extracted string from Hex to Byte. This will give IV in bytes.

This IV can be used for initializing cipher block and carrying out decryption.