E-Commerce Transaction With Google Pay™

Documentation For Merchant

Contents

Overview

This guide provides step-by-step instructions for integrating the Google Pay™ API into your e-commerce website. By following this guide, you will successfully render the Google Pay button, prompt the customer for their saved payment methods, and securely retrieve an encrypted Payment Token (ECv2).

Security Notice: This integration utilizes Network Tokenization. Your frontend application will never have access to the customer's raw card or pan number. The token you receive is encrypted and can only be decrypted by the Payment Gateway.

Prerequisites

Before beginning the frontend implementation, ensure you have the following:

- **Gateway Identifier:** The exact string identifying your gateway to Google (e.g., `citybank`).
- **Gateway Merchant ID:** Your unique merchant identifier (id) provided by the Payment Gateway.
- **HTTPS Environment:** Google Pay requires a secure HTTPS connection (even in local development, `localhost` is permitted).
- **Mandatory Policy Compliance:** When using our integration or a hosted checkout, you must adhere to the Google Pay and Wallet API's Acceptable Use Policy and accept the terms defined in the Google Pay API Terms of Service.

Hosted Checkout Integration Requirements

If you are embedding our hosted checkout page on your web domain using an `<iframe>` element, you must configure the iframe to allow the Payment Request API.

- Ensure you add `allow="payment *"` to your `<iframe>` tag.
- If your iframe utilizes the sandbox attribute, you must add the `allow-popups` token to permit the Google Pay payment sheet to display in a new window.

Step 1: Load the Google Pay API

Add the official Google Pay JavaScript library to your checkout page. It is highly recommended to load this script asynchronously.

HTML

```
<script async src="https://pay.google.com/gp/p/js/pay.js"
onload="onGooglePayLoaded()"></script>
<div id="google-pay-container"></div>
```

Step 2: Define Payment Parameters

Define the core objects that tell Google which payment methods you accept and how to encrypt the resulting token.

JavaScript

```
/* ============================   Google Pay
Configuration   ============================ */
let paymentsClient = null;

/* Base request */
const baseRequest = {
    apiVersion: 2,
    apiVersionMinor: 0
};

/* Allowed payment methods */
const allowedPaymentMethods = [{
    type: 'CARD',
    parameters: {
        allowedAuthMethods: ['PAN_ONLY', 'CRYPTOGRAM_3DS'],
        allowedCardNetworks: ['AMEX', 'VISA', 'MASTERCARD']
    },
    tokenizationSpecification: {
        type: 'PAYMENT_GATEWAY',
        parameters: {
```

```
            gateway: 'citybank',                // Your Gateway ID
            gatewayMerchantId: $('[name="MerchantId"]').val() // Your
Merchant ID
        }
    }
}];
```

Step 3: Initialize the Client & Check Readiness

When the script loads, initialize the **`PaymentsClient`**. You must check if the user's device and browser are capable of making a Google Pay transaction before rendering the button.

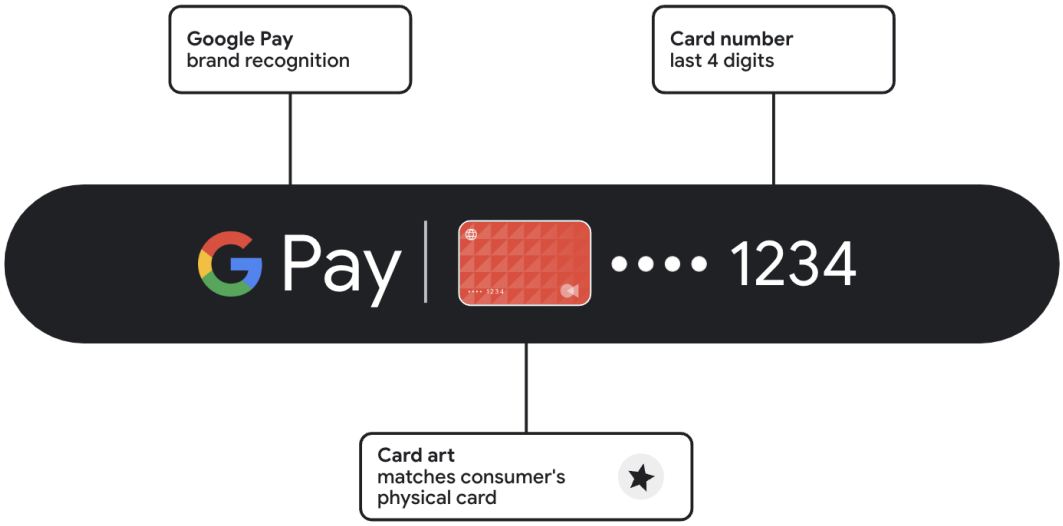
JavaScript

```
/* Check if Google Pay is available */
const isReadyToPayRequest = Object.assign({}, baseRequest);
isReadyToPayRequest.allowedPaymentMethods = allowedPaymentMethods;

/* Transaction details */
function getTransactionInfo() {
    return {
        totalPriceStatus: 'FINAL',
        totalPrice: $('[name="Amount"]').val(),
        currencyCode: 'BDT',
        countryCode: 'BD'
    };
}

/* Merchant info */
const merchantInfo = {
    merchantId: 'BCR2DN4TU6BMF3CE',
    merchantName: 'City Bank PLC'
};

/* Payment data request */
function getPaymentDataRequest() {
    const paymentDataRequest = Object.assign({}, baseRequest);
    paymentDataRequest.allowedPaymentMethods = allowedPaymentMethods;
    paymentDataRequest.transactionInfo = getTransactionInfo();
    paymentDataRequest.merchantInfo = merchantInfo;
    return paymentDataRequest;
}
```

Step 4: Render the Button

Once readiness is confirmed, create the button and attach it to the DOM container.

Example Image:



JavaScript

```
/* ============================   Load Google Pay
Button   ============================ */
function onGooglePayLoaded() {
    paymentsClient = new google.payments.api.PaymentsClient({
        environment: 'PRODUCTION' // Change to 'PRODUCTION' for live
    });

    paymentsClient.isReadyToPay(isReadyToPayRequest)
        .then(function(response) {
            if (response.result) {
                addGooglePayButton();
```

```
            } else {
                alert('Google Pay is not available on this
device/browser.');
            }
        })
        .catch(function(err) {
            console.error(err);
        });
}

function addGooglePayButton() {
    const button = paymentsClient.createButton({
        onClick: onGooglePayButtonClicked,
        buttonType: 'pay'
    });
    document.getElementById('google-pay-button').appendChild(button);
}
```

Step 5: Handle the Click Event & Retrieve the Token

When the user clicks the button, construct the final `PaymentDataRequest` (including the cart total) and launch the Google Pay UI.

JavaScript

```
/* ============================   Handle Payment   ============================
*/
function onGooglePayButtonClicked() {
    const paymentDataRequest = getPaymentDataRequest();

    paymentsClient.loadPaymentData(paymentDataRequest)
        .then(function(paymentData) {
            console.log('Payment Success:', paymentData);
            // Extract token to send backend
            const gPayEncryptedMessage =
paymentData.paymentMethodData.tokenizationData.token;

            // Send token to backend server
            getDecryptedToken(gPayEncryptedMessage);
        })
```

```
        .catch(function(err) {
            console.error('Payment failed', err);
        });
}
```

Step 6: Send the Token to Your Backend & Handle 3D Secure

Extract the encrypted token from the Google Pay response and forward it to your server for processing.

Important Note on 3D Secure (3DS):

Our integration supports two types of card credentials:

- **CRYPTOGRAM_3DS:** A tokenized virtual card stored on the device. Authentication is performed by Google Pay™, meaning standard 3DS is not required.
- **PAN_ONLY:** Physical card details stored in Google Pay™. **Standard 3DS applies.** When your server receives a `PAN_ONLY` credential in the encrypted payload, you are required to initiate 3DS verification. Your backend will execute the charge and, if 3DS is required, return an ACS URL. You must then formulate a POST request on the frontend to redirect the user to the ACS URL for standard 3DS verification, as demonstrated below:

JavaScript

```
function getDecryptedToken(gPayEncryptedMessage){
    $('#loadingImg').css('display','inline');
    var id = $('[name="InternalSequenceId"]').val();

    $.ajax({
        url: "/payment/gateway/gpay/transaction/payment",
        type: "POST",
        dataType: 'json',
        data: {id:id, gPayEncryptedMessage:gPayEncryptedMessage},
```

```
        success: function (data) {
            if(data.status === 'success') {
                // Initiating 3DS Verification for PAN_ONLY credentials
                var form = $('<form>', {
                    action: data.acsUrl,
                    method: 'POST'
                });
                form.append($('<input>', {
                    type: 'hidden',
                    name: 'creq',
                    value: data.creq
                }));
                $('body').append(form);
                $('#loadingImg').css('display','none');
                form.submit();
            }
            else {
                $('#loadingImg').css('display','none');
                window.location.href = data.redirectUrl;
            }
        },
        error: function (ex) {
            $('#loadingImg').css('display','none');
            alert(ex.error())
        }
    });
}
```

Next Steps

Once your frontend successfully logs the `paymentToken` and sends it to your backend, the frontend integration is complete. Your backend server must now forward this token to the Payment Gateway to execute the charge and handle any required 3D Secure authentication.