AUGUST 15, 2024

APG – SMARTBOX INTEGRATION GUIDE DOCUMENT

Amwal Pay
payment gateway

Confidential

Table of Contents

Document Version History

Versions	#	Date	Description / Modifications
	v 1.0	15-08-2024	Initiation version
	v 1.1	15-11-2024	Added API Pay By Token
	v 1.2	10-12-2024	Added Response Integrity Validation
	v 1.3	31-12-2024	Updated Smartbox.js Config Parameters

1 Introduction

The **AMWAL Payment Gateway** offers merchants a secure and seamless way to accept payments online through a highly customizable and easy-to-integrate checkout solution. This document provides a step-by-step guide on how to integrate the **AMWAL Payment Gateway Checkout Page** into your website using simple JavaScript code.

This integration ensures that customers can complete their transactions with ease, using a wide range of payment methods including credit cards, debit cards, and e-wallets. By following this guide, merchants will be able to integrate the payment gateway efficiently, reducing development time and ensuring a smooth payment experience for their customers.

1.1 Purpose of Document

The purpose of this document is to provide merchants with clear and concise instructions on how to integrate the **AMWAL Payment Gateway Checkout Page** into their websites using JavaScript. It covers all the essential steps required to implement the integration, ensuring compliance with security standards and optimal functionality.

This guide aims to:

- Simplify the integration process by providing easy-to-understand examples of the JavaScript code required for the payment gateway
- Ensure that merchants can implement the checkout process with minimal effort while maintaining a secure and efficient payment environment.
- Help developers troubleshoot common integration issues.

By the end of this guide, merchants will be able to:

- Set up the **AMWAL Checkout Page** on their websites.
- Accept payments from customers securely.
- Customize the checkout flow to suit their business needs.

1.2 Scope

This document is intended for **merchants and developers** who wish to integrate the **AMWAL Payment Gateway Checkout Page** into their online store or web application using JavaScript. It covers the following areas:

- The basic structure and requirements of integrating the **AMWAL Payment Gateway Checkout Page.**
- Detailed steps to implement the checkout functionality using JavaScript
- Code examples for initializing the payment gateway, handling payment responses, and error handling
- Best practices for ensuring a secure integration

2 Prerequisites

Before beginning the integration process, ensure that you have the following:

- **AMWAL Merchant Account**: You need to have an active merchant account with AMWAL Payment Gateway. If you do not have one, please contact AMWAL's support team using support@amwal-pay.com
- **API Credentials**: You will be provided with the credentials that include your Merchant ID, API Key, and any other information required to generate the secure hash.
- **Access to Smartbox.js**: The `Smartbox.js` file should be included or linked in your web application to initialize the AMWAL Checkout Page.
  Production: https://checkout.amwalpg.com/js/SmartBox.js?v=1.1
  SIT environment: https://test.amwalpg.com:19443/js/SmartBox.js?v=1.1
  UAT environment: **https://test.amwalpg.com:7443/js/SmartBox.js?v=1.1**

  SSL Certificate: Your website must be hosted over HTTPS to ensure secure communication between your site and the payment gateway.

3 Integration Steps

The integration process consists of embedding the **Smartbox.js** file, configuring the required parameters, and ensuring secure communication between your website and AMWAL Payment Gateway using a secure hash. The following steps will guide you through the process:

3.1 Include Smartbox.js in Your Website

The first step is to include the `Smartbox.js` file in your website's HTML file. You can do this by adding the following line of code within your `<head>` or just before the closing `</body>` tag:

```
<script src="path-to-your-js-folder/Smartbox.js"></script>
```

Make sure the file path is correct based on your project structure, it is better to link to the published `Smartbox.js` mainly for production to keep your web site updated with the new updates that may happen on the JS file in the future .

3.2 Configure Payment Settings

You will need to configure the necessary parameters to initialize the payment gateway. These configurations include details such as the amount to be paid, the currency, and the merchant information.

Below is an example configuration that you can modify based on your specific business needs:

```
SmartBox.Checkout.configure = {
    MID: /*your merchant Id*/,
    TID: /*Your Terminal Id*/,
    CurrencyId: /*only 512 is supported*/,
    AmountTrxn: /*Amount to be paid*/,
```

```
        MerchantReference: /*Merchant Reference code if exists*/,
        LanguageId: /*display language either 'ar' for Arabic or 'en' English
*/,
        PaymentViewType: /*either 1 for Popup or 2 for Full Screen*/,
        TrxDateTime: /*transaction date time*/,
        SessionToken: /* if the merchant is enabled for recurring payment, he can
        request a session token from an API (Check Merchant API Integration Doc):
        You can check this section for further info Get Smartbox Session Token */
        SecureHash: /*secure hash value for all the configuration above,
        you can check this section for further info Calculate Secure Hash */,

        completeCallback: function (data) {
          console.log("completeCallback Received Data", data);
        },
        errorCallback: function (data) {
          console.log("errorCallback Received Data", data);
        },
        cancelCallback: function () {
          console.log("cancelCallback Received Data", data);
        },
  };
```

Make sure to replace the placeholder values (`your_merchant_id`, `order12345`, etc.) with the actual values specific to your website and order.

4 Generate the Secure Hash

In order to ensure the integrity of the data being transmitted, AMWAL Payment Gateway uses **SHA-256** hashing with a secret key (Merchant Secure Key). Merchants need to calculate a secure hash on their end and send it along with the payment request. Our system will validate this hash to ensure the request is authentic and has not been tampered with.

Secure hash calculation shouldn't be handled at the client side (ex: JS side). For security reasons it should be handled at the backend and send to front end for further steps.

Follow these steps to calculate the secure hash:

1- **Prepare the Data**
   - Collect all the required parameters for the payment request.
   - Ensure the data is properly validated and sanitized to prevent potential attacks such as injection and length extension attacks.

2- **Merchant Secure Key**

- Use the Merchant Secure Key provided by AMWAL Payment Gateway. This key is unique to each merchant and serves as a salt to protect against rainbow table attacks.
- The key should be at least 64 bits (8 bytes) in length and kept confidential.

3- Concatenate the Data and the Secret Key

- Sort the data parameters alphabetically by key.
- Concatenate the sorted key-value pairs into a string in the format `key=value`.
- Concatenate this string with the Merchant Secure Key.

4- Generate the SHA-256 Hash

- Use the **SHA-256** algorithm to hash the concatenated string of key-value pairs and the Merchant Secure Key.
- The resulting hash will be the secure hash for the transaction.

5- Transmit the Data and Hash

- Send both the original data and the calculated SHA-256 hash to AMWAL Payment Gateway.

6- Verification by AMWAL

- AMWAL Payment Gateway will receive the transmitted data and hash.
- On the server-side, AMWAL Payment Gateway will concatenate the same data with the merchant's secret key and generate a SHA-256 hash.
- The system will compare the calculated hash with the hash received from the merchant. If they match, the transaction is considered secure and untampered.

4.1 Example Code

Here are some sample code snippets that demonstrate how to calculate the secure hash in different programming languages.

4.1.1 Typescript Example

```
let paramsObj = {
  Amount: "10",
  CurrencyId: "512",
  MerchantId: "48804",
  MerchantReference: '',
  TrxDateTime: " 2024-12-31T15:27:10.361969Z",
  TerminalId: "113176",
  /* if the merchant is not enabled for recurring payment, Use Empty Session token
But if Merchant is enabled for recurring payment he can
request a session token from an API (Check Merchant API Integration Doc):
You can check this section for further info Get Smartbox Session Token */
  SessionToken: `eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..w__uUsJEv6GYqWSAbSHl4w.g44u9CS3h
Ye3DWaixF3rITecIognMFIAgg7eQUBLl7EUR76acc2km4xwfKKLJuFtZbR4A4I7JgJX3jFEqxgfERYEvbS-1UKQYIZUme
                        ch2K0y21LsUyQXz3tcKZWoBKHJh4NRvQr1AkWwPQkc3xhOzevGDQux4EGEeRRkchayvLBn39ONH
                        _ZmuyGuz3OStoGLgKrQaN9L2ga5g54u-2SsQP_Kf2A0RRvdgioXdAxsH5j2Z0xjkc1HE1t10cyQ
                        e3JBvwCsE8ohap5TtcXZFT4B88wql4Ut4kZRgK0VOHpl4QgNALeJBIrLN21AEc-K7FexWYCw28
                        itGzmPODSxuBpMyL80BR9CR_1.SL9xnd_0bIDYI0RLCPtVPg`
}
```

```
function calcHash(obj: any, secret: string) {
  try {
    let objSorted = Object.keys(obj)
      .sort()
      .reduce(
        (acc, key) => ({
          ...acc,
          [key]: obj[key],
        }),
        {},
      );
    let dataPreparedForHashing = Object.entries(objSorted)
      .map(([key, value]) => `${key}=${value}`)
      .join('&');

    // console.log(dataPreparedForHashing) should be like this
    //      Amount=${YOUR_AMOUNT}&CurrencyId=512&
    //              MerchantId=${YOUR_MERCHANT_ID}&
    //              MerchantReference=${YOUR_MERCHANT_REFERENCE}&
    //              RequestDateTime=${YOUR_REQUEST_DATE_TIME_IN_ISO_FORMAT}&
    //              SessionToken=&TerminalId=${YOUR_TERMINAL_ID}`

    const hmac = crypto.createHmac('sha256', Buffer.from(secret, 'hex'));
    const hashValue = hmac.update(dataPreparedForHashing, 'utf-8').digest('hex');
    return hashValue.toUpperCase();
  } catch (error) {
    return '';
  }
}
let hash = calcHash(paramsObj, "YOUR_SECRET_KEY")
```

4.1.2 PHP Example

```
<?php
function encryptWithSHA256($input, $hexKey) {
    // Convert the hex key to binary
    $binaryKey = hex2bin($hexKey);
    // Calculate the SHA-256 hash using hash_hmac
    $hash = hash_hmac('sha256', $input, $binaryKey);
    return $hash;

    // Provided input text
    inputText = "Amount=36&CurrencyId=512&MerchantId=1369217&
    MerchantReference=26_23122645&RequestDateTime=2023-12-26T09:42:46Z&SessionToken=&
    TerminalId=6942344";
```

```
    // Provided hex key
    $hexKey = "9FFA1F36D6E8A136482DF921E856709226DE5A974DB2673F84DB79DA788F7E19";
    // Calculate SHA-256 hash
    $result = encryptWithSHA256($inputText, $hexKey);
?>
```

4.1.3 Example Secure Hash Calculation

Let's consider the following example:

- **Merchant Secure Key**
  64373939653761352D343730352D343666632D623264312D34363235323463616165564654
- **Data to be Hashed**
  "Amount=10&CurrencyId=512&MerchantId=48804&MerchantReference=&RequestDateTime=121123103839&SessionToken=&TerminalId=113176"

After performing the steps above, the resulting secure hash will be:

8A8E9F1BC2979D6D89A947008831199E76331689D5B28D41395FA1DA65FFDE7B

This secure hash should be sent along with the request to AMWAL Payment Gateway, and our system will validate it to ensure the transaction's integrity.

5 Submit the Payment Request

Once the configuration and secure hash are set, submit the payment request by initializing the checkout session using `Smartbox.js`. When the user clicks the "Pay" button, this request is sent to the AMWAL Payment Gateway for validation:

```
<button id="payButton">Pay Now</button>
<script>
    document.getElementById("payButton").onclick = function() {
    Smartbox.submitPayment(); // Triggers the payment process
  };
</script>
```

Once the request is submitted, the AMWAL Payment Gateway will validate the request and open the checkout page for the user to complete the payment.

6 Configuration

This section describes the parameters you need to configure before submitting the payment request. Below is a table of all the required parameters:

Field Name	Mandatory	Field Type	Constraints	Description	Sample value
MID	Yes	Numeric	Length:1-19	AMWAL Payment Gateway Merchant Id	8305
TID	Yes	Numeric	Length:1-19	AMWAL Payment Gateway TerminalID	189903
TrxnAmount	Yes	Numeric	Length:1-10	Amount	105.755
MerchantReference	No	Numeric	Length:1-36	Merchant's System Reference	50049
CurrencyId	No	Numeric	Length:3	ID of the Currency – "only 512 is supported"	512
LanguageType	No	String	Length:2	Smartbox Display Language	"en" Or "ar"
RequestDateTime	Yes	String	DateTime	Request Date Time	"2023-11-12T10:38:39.92000Z"
SessionToken	No	String	Length:Max 256		eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUh TMjU2In0..w__uUsJEv6GYqWSAbSHl4w. g44u9CS3hFS-Ye3DWaixF3rlTecIognMFIAgg 7eQUBLl7EUR76acc2km4xwfKKLJuFtZbR4 A4I7JgJX3jFEqxgfERYEvbS- 1UKQYIZUmech2K0y21LsUyQXz3tcKZ WoBKHJh4NRvQr1AkWwPQkc3xhOzevG

					DQux4EGEeRRkchayvLBn39ONKZAT7ODJ f5DiUogoZJy_ZmuyGuz3OStoGLgKrQaN9 L2ga5g54u-2SsQP_Kf2A0RRvdgioXdAxsH 5j2Z0xjkc1HE1t10cyCymX8uK_Ds4-e3JBvwCsE8ohap5TtcXZFT4B88wql 4Ut4kZRgK0VOHpl4QgNALeJBIrLN21 AEc-K7FexWYCw28-itGzmPODSxuBpMyL80BR9CR_1.SL9xnd_0bIDYI0RLCPtVPg
PaymentViewType	No	Numeric	Length:1	Checkout page Display Type either Popup or Fullscreen	1 Or 2 1=Popup 2=Fullscreen
SecureHash	Yes	String	Generate The Secure Hash	Secure Hash Value	"84EB3BF8F62EF25717D1E9E13C3CFB719A890980BBF2631AFD8965182ADE1754"

7 Sample iFrame Implementation

Once you've completed the required configurations for `SmartBox.js`, you can easily integrate the payment page into your website using an iframe. Follow the example below to display the payment page within an iframe element in your DOM.

JavaScript Integration Example:

This code snippet demonstrates how to fetch the SmartBox URL and embed it into an iframe with the ID `#framePaymentPage`.

```
// Fetch the payment URL from SmartBox.js
var url = SmartBox.Checkout.getSmartBoxUrl()

// Target the iframe element in your DOM
var $iframe = $("#framePaymentPage")

// Set the payment page URL into the iframe
$iframe.attr("src", url);
```

```
// Show the containing div (in case it's hidden by default)
$("#frameDiv").show();
```

In this implementation:

- `SmartBox.Checkout.getSmartBoxUrl()` retrieves the payment URL after you've successfully configured SmartBox.js.
- The iframe is targeted using its ID `#framePaymentPage` and the URL is assigned to it.
- The surrounding div `#frameDiv` is then shown to display the iframe.

Custom Styling for iFrame:

You can define custom styles for the iframe and its container to control how it appears on your website. Below is an example of how you can style the iframe for a full-screen, centered payment page:

```
<style>
      #frameDiv {
        width: auto;
        height: auto;
        position: absolute;
        top: 0;
        bottom: 0;
        left: 0;
        right: 0;
        background: rgba(0, 0, 0, 0.35);
        opacity: 1;
        z-index: 1000;
        pointer-events: auto;
        -webkit-tap-highlight-color: transparent;
        transition: opacity 400ms cubic-bezier(0.25, 0.8, 0.25, 1);
      }
      #framePaymentPage {
        position: absolute !important;
        right: 0 !important;
        top: 0 !important;
        height: 100%;
        width: 100%;
        border: none !important;
        overflow: hidden !important;
```

```
        z-index: 999999 !important;
        filter: none !important;
        padding: 0 !important;
        margin: 0 !important;
    }
</style>
```

In this styling:

- The `#frameDiv` is styled to cover the full screen with a semi-transparent background, ensuring the iframe appears centered with a focused overlay.
- The `#framePaymentPage` iframe itself is positioned absolutely within the container to ensure it occupies the full height and width of the div without borders or scrollbars.

Final Steps:

To integrate the payment page into your merchant site:

1- Place the provided JavaScript and HTML snippets into the appropriate sections of your page.
2- Ensure the iframe element (`#framePaymentPage`) is defined in your HTML, and that the containing `#frameDiv` is properly configured.
3- Customize the styles as needed to fit your website's design.

This setup will embed the AMWAL Payment Gateway's checkout page seamlessly into your website.

8 Acquiring Session Token

8.1 Overview

The `Customer/GetSmartboxDirectCallSessionToken` API provides a session token that merchants can use to access a customer's saved cards for recurring payments. Merchants should acquire and store the `customerId` when the customer first chooses to save their card during payment.

8.2 Endpoint Information

- **Method**: POST
- **Content Type**: application/json
- **Authentication**: Secure Hash (secureHashValue)
- **URLS**:
  - **Production**: https://webhook.amwalpg.com/Customer/GetSmartboxDirectCallSessionToken

- **SIT**: https://test.amwalpg.com:24443/Customer/GetSmartboxDirectCallSessionToken
- **UAT**: https://test.amwalpg.com:14443/Customer/GetSmartboxDirectCallSessionToken

8.3 Request Parameters

The following parameters are required in the JSON request body:

```
{
/*customer id is the value that merchant received with the response of
the first transaction execution and with request to enable save card
checkbox at the Payment Page.

The unique CustomerId should be received in the response so that
merchant can use it here to generate a private session token for the
next payment.

This session token should allow the customer to see his saved cards in
order to choose from them and proceed with payment*/
  "customerId":"7267684c-3600-403e-81c7-87d778496e28",
  "merchantId": 7921,
  "requestDateTime": "2023-11-12T10:38:39.92000Z",
  "secureHashValue":"84EB3BF8F62EF25717D1E9E13C3CFB719A890980BBF2631AFD
8965182ADE1754"
}
```

Field Name	Mandatory	Field Type	Constraints	Description	Sample value
customerId	Yes	String	Length:1-36	Unique identifier for the customer. Received when the customer checks "save card" option, The completeCallBack will hold the customer Id Value.	7267684c-3600-403e-81c7-87d778496e28
merchantId	Yes	Numeric	Length:1-30	AMWAL Payment Gateway MerchantID	189903

requestDateTime	Yes	String	DateTime	Request Date Time	"2023-11-12T10:38:39.92000Z"
secureHashValue	Yes	String	Look at the Section "Generate The Secure Hash"	Secure Hash Value	"84EB3BF8F62EF25717D1E9E13C3CFB719A890980BBF2631AFD8965182ADE1754"

NOTE: Ensure that the `requestDateTime` matches the server time closely to avoid security validation errors.

8.4 Sample Request

```
POST /Customer/GetSmartboxDirectCallSessionToken

Content-Type: application/json
```

```
{
/*customer id is the value that merchant received with the response of
the first transaction execution and with request to enable save card
checkbox at the Payment Page, the complete call back will hold the
customer Id value.

The unique CustomerId should be received in the response so that
merchant can save and use it here to generate a private session token
for the next payment.

This session token should allow the customer to see his saved cards in
order to choose from them and proceed with payment*/

  "customerId":"7267684c-3600-403e-81c7-87d778496e28",
  "merchantId": 7921,
  "requestDateTime": "2023-11-12T10:38:39.92000Z",
  "secureHashValue":"84EB3BF8F62EF25717D1E9E13C3CFB719A890980BBF2631AFD
8965182ADE1754"
}
```

8.5 Sample Response

If the request is successful, the response will include a `sessionToken`, allowing the customer to view and select from their saved cards.

```
{
  "success": true,
  "responseCode": "00",
  "message": "Success",
  "data": {
    "sessionToken":
"eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..w__uUsJEv6GYqWSAbSHl4
w.g44u9CS3hFS-
Ye3DWaixF3rITecIognMFIAgg7eQUBLl7EUR76acc2km4xwfKKLJuFtZbR4A4I7JgJX3jFE
qxgfERYEvbS-
```

```
  },
  "errorList": []
}
```

8.6 Generating Secure Hash

To secure the API call, generate a hash value (`secureHashValue`) by concatenating and sorting the request parameters. Use a SHA-256 HMAC with your secret key to produce the hash.

8.6.1 Steps to generate the Secure Hash

Please check section 4

Example sorted string:

Amount=1&CurrencyId=512&MerchantId=7921&MerchantReference=1581&RequestDateTime=2024-10-29T14:16:19.2682Z&SessionToken=<your-session-token>&TerminalId=221143

Secure hash value output:

4C690E7CD330FB0CE5DE9E8219532DED1CC493F26D4693C824622EBD21D9BB8F

8.7 Response Integrity Validation

For Integrity concerns we included a secureHashValue that you can use to make sure the response is received as expected

```
{
    "success": true,
    "responseCode": "00",
    "message": "Success",
    "data": {
        "systemTraceNr": null,
        "message": "CAPTURED - ",
        "transactionId": "6b75efb6-84ab-46f2-8a32-351a23490f45",
        "isOtpRequired": false,
        "hostResponseData": {
            "TransactionId": "202434515895614",
            "Rrn": "434580000143",
            "TrackId": "6b75efb684ab46f28a32351a23490f45",
            "PaymentId": null,
```

```
            "Auth": "537465"
        },
        "terminalId": 221143,
        "transactionTypeId": 2,
        "transactionTypeDisplayName": "Purchase",
        "merchantId": 7921,
        "currency": null,
        "amount": 1,
        "currencyId": 512,
        "merchantName": "MahmoudGrocery",
        "transactionTime": "2024-12-10T15:56:37.1099636Z",
        "customerId": "82383bce-6e32-4f5b-b1ea-7e00d5c446ed",
        "customerTokenId": "aacd0817-2246-4521-a3df-9f3971c63a22",
        "secureHashValue":
"A1091C89D4C2D3E95630722DE0469DBDAB6FD01AF005E76A36ABFA2D45997B91", //
compare this value with your calculated value it should be the same
        "merchantReference": "201204"
    },
    "errorList": []
}
```

Calculate the secure hash for the following parameters with their correspondent values from the response

```
let integrityParametrs = {
    "amount" :  response.data.amount,
    "currencyId" : response.data.currencyId,
    "customerId" : response.data.customerId,
    "customerTokenId" : response.data.customerTokenId,
    "merchantId" : response.data.merchantId,
    "merchantReference" : response.data. merchantReference,
    "responseCode" : response.responseCode,
    "terminalId" : response.data.terminalId,
    "transactionId" :  response.data.transactionId,
    "transactionTime" : response.data.transactionTime,
};
```

Example sorted string:

amount=1&currencyId=512&customerId=82383bce-6e32-4f5b-b1ea-7e00d5c446ed&customerTokenId=aacd0817-2246-4521-a3df-9f3971c63a22&merchantId=7921&merchantReference=201204&responseCode=00&terminalId=221143&transactionId=6b75efb6-84ab-46f2-8a32-351a23490f45&transactionTime=2024-12-10T15:56:37.1099636Z

Secure hash value output:

4E21F6F06C188F38B0B6A3CD2EFD9DC93EE83E3D5284C708B1C8930D9BCF0D11

compare this secure hash result with the one received in the response
it should be the same