Amazon Pay Integration Guide

Web version 2.0

Document Version 1.0

Amazon pay
Web Integration Guide

Contents

Introduction

Amazon Pay provides buyers with a secure, trusted, and convenient way to log in and pay for their purchases by selecting a payment method stored in their Amazon account.

After completing the steps in this guide, Amazon pay will be enabled in your website, allowing buyers to sign in with their Amazon credentials to make a purchase.

Integration Overview

During a typical buyer experience, the buyer places items in their cart and at checkout website uses Amazon pay as a payment option.

During this process, there are several steps taking place:

1. When the buyer selects, Amazon pay option, your website will use Amazon pay SDK services to invoke an API hosted on your backend server to Sign and Encrypt the transaction payload using the shared key and Server Side SDK provided by Amazon pay post validation of the transaction information to be indeed valid.
2. Your website invokes the Amazon pay url generated above with the transaction payload and allows buyer to complete the payment.
3. Payment is processed with customer interaction and response is sent back to your Thank you page URL via HTTP GET.
4. Your website makes a request to your server side Backend to verify the signature of the response, Amount and other details.
5. Your server side Backend used the Amazon pay Server Side SDK to verify the signature and responds accordingly.

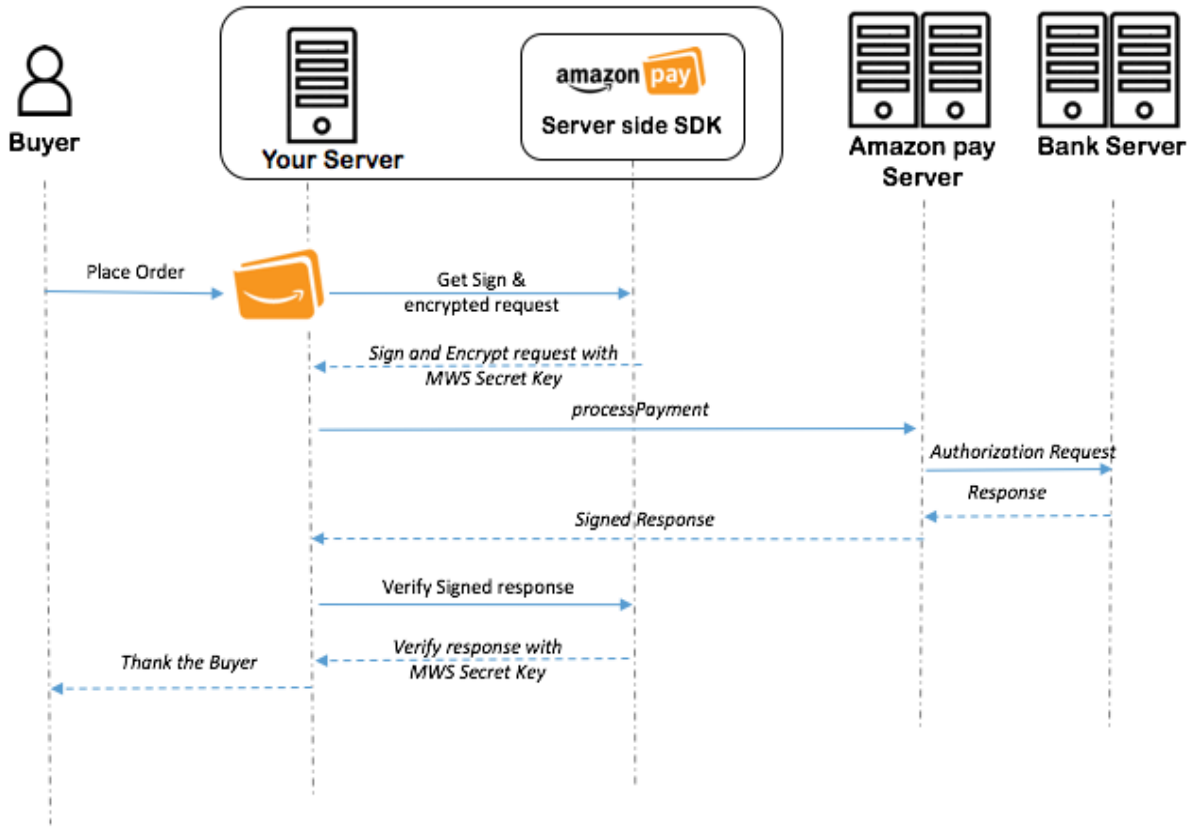


***Figure 1**. Overview of the payment workflow.*

Prerequisites

Sign up for an Amazon Payments User Account

You need a Seller ID and MWS Access keys to sign your requests.

If you have an existing merchant account, you can retrieve your Seller ID and MWS Access Keys on Seller Central. To retrieve the keys, sign in to your Seller Central account.

Integration Steps

The integration steps in this guide show you how to display an Amazon pay at checkout and accept payments.

Step 1. Specify the Parameter Values

Below are the mandatory and optional parameters used for making a transaction request to Amazon Pay

Mandatory Parameters

Parameters	Type	Description
orderTotalAmount	String	The amount in number. Eg.100, 10.5, 1.99
orderTotalCurrencyCode	String	The currency code. Value is INR.
sellerOrderId	String	Merchant specified identifier of this order. Maximum: 127 characters
startTime	String	Time in EPOCH

Optional Parameters

Parameters	Type	Description
isSandbox	String	This is set to “true” for testing in sandbox mode.
transactionTimeout	String	Merchant specified timeout in seconds
customInformation	String	Any additional metadata you want to pass in request
sellerNote	String	Any additional note you want to pass in request

sellerStoreName	String	Store name if that needs to be specified

Step 2. Generate a Signature for the Payment Request and Encrypt the request

If you are using Amazon Pay, you must include a signature in the request parameters so that Amazon can authenticate your payment requests. If you do not include a signature, or if the signature is incorrect, the buyer will be directed to an error page that will redirect to your specified returnURL for failure handling.

Using the Amazon Pay client libraries

By using an Amazon Pay client library for each of the Amazon pay operation, you save time and you know the request you send is correctly formatted. For example, the Amazon Pay client libraries perform the following tasks for you during process payment request:

- **startTime**- adds a timestamp on each request you submit. Each request must contain the timestamp of the request.
- **Signature** - creates a valid request HMAC-SHA signature. Each request must have a valid signature or the request is rejected. A request signature is calculated using your Secret Access Key, which is a shared secret, given to you when you registered, and known only to you and Amazon MWS.
- **Encrypting the request**- Amazon Pay client libraries performs encryption on the request using the public key bundled into the library by AES-GCM Algorithm.

Links to SDKs and Samples: PHP, Java

If you create your own client library

You can create your own client library for use with Amazon Pay. Your code should construct and sign a request in the format expected by Amazon MWS, and then encrypt the payload using AES-GCM No Padding algorithm.

You can create request to Amazon Pay by following the below steps:

To generate the signature, complete the following tasks:

1. Construct the string to sign.
2. Sign the string with your MWS secret access key.
3. Add the signature to the parameters to be encrypted for generating the payload.

To create a valid signature, you need to construct the string to sign according to the Amazon MWS V2 signature spec. The string consists of the following elements, with each section separated by a new line:

- The HTTP action. For a Pay with Amazon request it is always **POST**.
- The request domain. For a Pay with Amazon request it is always a forward slash **(/)**.
- Sorted parameters in query string format, with the URL encoded parameter name and value.

Task 1: Create String to Sign (MWS Signature Version 2)

1. The HTTP Action: Start with the request method, followed by a newline character. For an Amazon Pay request it is always POST. For human readability, the newline character is represented as \n.

```
POST\n
```

2. Add the HTTP host header (endpoint) in lowercase, followed by a newline character.

```
amazonpay.amazon.in\n
```

3. The request domain: For an Amazon Pay request it is always a forward slash (/).

```
/\n
```

4.

a. Add the query string components, as UTF-8 characters which are URL encoded (hexadecimal characters must be uppercase).
b. Sort the query string components on basis of string comparison.
c. Separate parameter names from their values with the equal sign character (=), even if the value is empty. Separate parameter and value pairs with the ampersand character (&). Concatenate the parameters and their values to make one long string with no spaces. Spaces within a parameter value are allowed, but must be URL encoded as %20. In the concatenated string, period characters (.) are not escaped. RFC 3986 considers the period character an unreserved character, so it is not URL encoded.

> **Note**: RFC 3986 does not specify what happens with ASCII control characters, extended UTF-8 characters, and other characters reserved by RFC 1738. Since any

values may be passed into a string value, these other characters should be percent encoded as %XY where X and Y are uppercase hex characters. Extended UTF-8 characters take the form %XY%ZA... (this handles multibytes).

Note: While generating a signature, include **only the parameters that must be signed**. The **mandatory** parameters for payment request are:

- AWSAccessKeyId
- orderTotalAmount
- orderTotalCurrencyCode
- sellerId
- sellerOrderId
- startTime

You should include the following parameters when generating a signature only if you designated a value other than the default:

- transactionTimeout
- isSandbox
- sellerNote
- customInformation

Use "**isSandbox=true**" only for sandbox mode. Incase if this parameter is not present then it defaults to "false" enabling production mode.

The following example shows what your string to sign might look like, with the parameters concatenated with the ampersand character (&), and sorted on basis of string comparison. Note that you must replace the value of each field according to the parameter values that you want to use.

```
AWSAccessKeyId=AKIAJFIEPUA4TOSN4BPA
&isSandbox=true
&orderTotalAmount=1.00
&orderTotalCurrencyCode=INR
&sellerId= SELLERID
&sellerOrderId= SELLERORDERID
&startTime = 1486454855
&transactionTimeout=1000000
```

Note

- To construct the finished canonical request, combine all the components from each step. As shown, each component ends with a newline character.

```
POST\n
amazonpay.amazon.in\n
/\n
AWSAccessKeyId=AKIAJFIEPUA4TOSN4BP
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&isSandbox=true
&orderTotalAmount=1.00
&orderTotalCurrencyCode=INR
&sellerId= SELLERID
&sellerOrderId= YOURORDERID
&startTime = 1486454855
&transactionTimeout=1000000
```

Task 2: Calculate the Signature

After you've created the canonical string as described in create the string to sign, calculate the signature by creating a hash-based message authentication code (HMAC) that uses HMAC-SHA256 protocols.

The signature is calculated with the canonical string and secret key as inputs to a keyed hash function.

```
POST\n
amazonpay.amazon.in\n
/\n
AWSAccessKeyId=AKIAJFIEPUA4TOSN4BP
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&isSandbox=true
&orderTotalAmount=1.00
&orderTotalCurrencyCode=INR
&sellerId= SELLERID
&sellerOrderId= YOURORDERID
&startTime = 1486454855
&transactionTimeout=1000000
```

The resulting signature must be base-64 encoded.

```
OWaj6KqS30yVa%252Bi5mCCOAyiMigWGpAOtFQ0KVR%252BxACw%253D
```

Task 3: Encryption

Once the signature is generated, you will need to create a URI encoded plain data string representation of the AWSAccessKeyId, sorted parameters and Signature. Signature will be the last parameter. The sample string representation will look like this

Sample Payload:

```
AWSAccessKeyId=AKIAJFIEPUA4TOSN4BP&isSandbox=true&orderTotalAmount=10.00&orderTotalCurrencyCode=INR&sellerId=A376KVFLEFT6BB&sellerNote=test&sellerOrderId=test&startTime=1486454855&transactionTimeout=100000&Signature=OWaj6KqS30yVa%252Bi5mCCOAyiMigWGpAOtFQ0KVR%252BxACw%253D
```

Note :

Most common application languages include cryptographic libraries that allow you to perform encryption in your applications. Two commonly available open source tools are Bouncy Castle and OpenSSL.

Steps for Encryption

- Generate a random session key of length 16 bytes and initialization vector.
- Encrypt payload using a cipher created with the session key and initialization vector by AES/GCM/NoPadding algorithm.
- Encrypt that session key from the master public RSA key available at https://amazonpay.amazon.in/getDynamicConfig?key=serverSideSDKJava using RSA/ECB/OAEPWithSHA-1AndMGF1Padding algorithm.
- Pass the encrypted envelope key and initialization vector with the encrypted payload. Both the encrypted envelope key and initialization vector must be base64 encoded.

The following example demonstrates steps for encryption using java.

```
package com.amazonpay.encryption.examples;

import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Security;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.HashMap;
import java.util.Map;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Base64;

public class EnvelopeEncryption {
```

```
	private static final SecureRandom secureRandom = new SecureRandom();
	private static final String stringToBeEncrypted =
"AWSAccessKeyId=AKIAJIJMTWVP5TOL7Y7Q&isSandbox=true&orderTotalAmount=10.00&orderTotalCurr
encyCode=INR&sellerId=A1T5DRWMBFNP17&sellerNote=test&sellerOrderId=test&startTime=1489490
390&transactionTimeout=100000&Signature=nWYMwpsppgFophKTfnOe7TqqOXQF%2BUeVKFev71urez8%3D"
;


	public static void main(String[] args) throws NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException,
			InvalidAlgorithmParameterException, IOException,
IllegalBlockSizeException, BadPaddingException,
			NoSuchProviderException {
		Map<String, String> encryptedResult =
EnvelopeEncryption.encrypt(stringToBeEncrypted);
		// encryptedResult contains the key,iv,payload parameters required for
		// making payment request
		System.out.println(encryptedResult.toString());
	}


	public static Map<String, String> encrypt(String stringToBeEncrypted) {
		byte[] rawKey = new byte[16];
		secureRandom.nextBytes(rawKey);
		Map<String, String> result = new HashMap<>(3);
		try {
			// Bouncycastle is preferred provider, Sun JCE can also be used
			Security.addProvider(new BouncyCastleProvider());
			Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding",
BouncyCastleProvider.PROVIDER_NAME);
			// 1. Generate the session Key which will be used for encrypting the
			// payload
			SecretKey key = new SecretKeySpec(rawKey, "AES");
			// 2. Generate the initialization vector
			byte[] iv = new byte[cipher.getBlockSize()];
			// 3. Generate the AES-GCM encrypted payload
			byte[] encryptedData = null;
			secureRandom.nextBytes(iv);
			AlgorithmParameterSpec ivSpec = new IvParameterSpec(iv);
			cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
			encryptedData = cipher.doFinal(stringToBeEncrypted.getBytes());
			// 4. RSA encrypt the session key
			byte[] cipherText = null;
			PublicKey publicKey;
			// In a production implementation, load this key from
```

```
                        //
https://amazonpay.amazon.in/getDynamicConfig?key=serverSideSDKJava
                        String encodedKey =
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDcW7r1yKMtr+P3ZU3IaC6raMCbg7Wo1jEP4CTa36ByIjeI4eNA
miozmwqwwx3W5k+gPH/KI4lZojWYZPswfDz7LNF2q8jnSSzNHrdgw2vtyc63XL+h1kZ9hdBc+zv7QEnnEr1OxxN/E
WGOR3PXGypBRT83udG6eHbG5vNlGAhlXwIDAQAB";
                        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
                        X509EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(Base64.decode(encodedKey));
                        publicKey = keyFactory.generatePublic(publicKeySpec);
                        Cipher rsaCipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
1AndMGF1Padding","BC");
                        rsaCipher.init(Cipher.ENCRYPT_MODE, publicKey);
                        cipherText = rsaCipher.doFinal(rawKey);
                        result.put("IV", new String(Base64.encode(iv)));
                        result.put("KEY", new String(Base64.encode(cipherText)));
                        result.put("PAYLOAD", new String(Base64.encode(encryptedData)));
                        return result;
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return result;
        }

}
```

Task 4: Generate the Amazon Pay processPayment url and redirect the buyer

The encrypted data, initialization vector , encrypted session key and redirectUrl is appended to the url given by Amazon pay

URL:

Redirect the user to Amazon Pay using the following URI.

1 https://amazonpay.amazon.in/initiatePayment

Query Parameters:

Use the following parameters in the redirect. All of the parameter values must be url-encoded.

- **payload**: This is the base64 and URI encoded encrypted data

- **key**: This is the base64 and URI encoded encrypted session key
- **iv**: This is the base64 and URI encoded SecureRandom session key of 128 bit fixed length
- **redirect_url** : This is the return URL or "Thank you" page on the merchant website. The redirectUrl will have be either http or https and needs to be whitelisted by Amazon Pay. This redirect URL should be a static one as Amazon Pay needs the complete URL to be whitelited and only whitelisting the domain name will not work. Although you can get multiple URLs whitelisted. Please contact your Integration specialist for whitelisting your production URL.

Sample Request:

```
https://amazonpay.amazon.in/initiatePayment
?payload=ylcdc3mdQq3wDqxw3MrFLEwx8nhSf1VdCSZmuDNyb8VTjAt9OHuYXurXgOEhOkboWdakZ0f
8DN%2Fn8v8h8RFOec4FlYPyNo1BkL9zNksJjQKaEArgOI1KyIYFhDhILFecixP%2Fmyvcoc%2Bpvbst%2Fb7
7q7d1Uw2avHN6fObV%2Bhon%2F%2BtVtB1NDl%2Fg4jr%2BnqSLS46iX5yfEj5w%2BIJo5W7FLClzc%2B
12qUQiekE78T65wOPTfONnGGYNn1YawDKtPN57sz4Dhv8Fe3GIoFeZA6MSX%2BLh%2Bk%2BKyBWSF
uxO6oRd6OtUTfLOhWenXw7osZn9ARPVNMYxYAbJ&key=sI7%2F3CzdpEo87Ud5LttysDhA742gFRAKs6f
yrD3GB9o%2FvJuUGA4pwkT4%2BAF4poNXNChOsERs%2FeMksGCAdrrEfZ9oHVQ%2FvXAhK%2FUudbs
%2B26aqTDPIfVg%2F1Flan2r1ndwgdsNXg8SL6us6ohtBFZT%2FjKbYo64ptX4SAvMAyVb695E%3D&iv=C
kcBbgXVRhuw14szVmFrGQ%3D%3D&redirectUrl=http%3A%2F%2Fyour-
domain.com%2Fthankyou.php
```

After user completes payment, the user is redirected to the redirect_url that you provided with status and other values as GET parameters.

Sample Response:

```
http://your-domain.com/thankyou.php? amazonOrderId=S04-5003075-4211618&description=Txn
Success&orderTotalAmount=1.00&orderTotalCurrencyCode=INR&reasonCode=001&sellerOrderId=Y
OUR_SELLER_ORDER_ID&signature=oiK4hbAFYh%2BtBZmCmzyoRyqDFPDyKxT0lWF4WUvFSAI%3D&s
tatus=SUCCESS&transactionDate=1488169624101
```

Step 3. Integrating redirect URL parameters

Whether the request was successful or failed, Amazon pay returns buyer to a URL containing parameters that you can integrate into your order management system. For example, if the request was successful, at the conclusion of the checkout flow the buyer will be redirected back to the URL that you specified in the request, along with parameters describing the successful transaction.

Important: Before fulfilling orders on a successful response, you should verify the signature, amount, sellerOrderId, and currencyCode parameters to ensure that the response was sent by Amazon Pay and that it has been properly processed.

Note: Integrating return URL parameters is one way to monitor the status of transactions on your web page, but we recommend that you also use Instant Payment Notification (IPN) to monitor transactions. IPN is a HTML POST notification that is sent when a transaction either completes successfully or fails. You can specify the default URL to handle IPN in your Amazon Payments account settings. To view transactions in Seller Central, sign in and go to Orders, and then select Manage Transactions.

The transaction response will contain the following parameters

Parameters	Type	Description
amazonOrderId	String	This is the unique order id generated by Amazon for the particular transaction
sellerOrderId	String	Merchant generated order id that was sent to Amazon
reasonCode	String	Reason code will provide the reasons why the status can end up in a state. See Reason codes for possible values.
description	String	Description of transaction status
signature	String	The signature sent by Amazon for verification.
status	String	The transaction status. The status values will be SUCCESS or a FAILURE.
amount	String	The amount charged. Eg.100, 10.5, 1.99
currencyCode	String	The currency code. Value is INR.
transactionDate	String	Time in EPOCH

Optional Parameter

Parameters	Type	Description
customInformation	String	Any meta data that was sent to Amazon. This is added in response only if customInformation is added as part of request.

Step 4. Validate the Signature

After an order is placed, you should validate the signature in the return URL to ensure that it came from Amazon.

```
http://your-domain.com/thankyou.php?amazonOrderId=S04-5003075-4211618&description=Txn Success&orderTotalAmount=1.00&orderTotalCurrencyCode=INR&reasonCode=001&sellerOrderId=YOUR_SELLER_ORDER_ID&signature=oiK4hbAFYh%2BtBZmCmzyoRyqDFPDyKxT0lWF4WUvFSAI%3D&status=SUCCESS&transactionDate=1488169624101
```

To validate the returned signature, complete the following tasks:

1. URL Decode the response parameters.
2. Construct the string to sign (refer Task 1).
3. The required parameters to sign are:

    - AWSAccessKeyId
    - SignatureMethod
    - SignatureVersion
    - amazonOrderId
    - description
    - orderTotalAmount
    - orderTotalCurrencyCode
    - reasonCode
    - status
    - transactionDate

4. Sign the string with your Amazon MWS secret access key.

```
POST\n
amazonpay.amazon.in\n
/\n
AWSAccessKeyId=YOUR_ AWSAccessKeyId
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&amazonOrderId=P04-7840151-7793053
&description=Txn Success
&orderTotalAmount=2.00
&orderTotalCurrencyCode=INR
&reasonCode=001
&status=SUCCESS
&transactionDate=1488953370248
```

5. Generate the signature on your server and compare with the returned signature. The signature is valid if generated signature and returned signature are same.

Step 5. Test Your Integration in the Sandbox Environment and Then Switch to Production

The Amazon pay Sandbox environment enables you to thoroughly test your Amazon pay integration before going live. When you test your implementation in Sandbox mode, you can simulate the buyer experience as your buyers navigate through the Amazon Pay on your website.

In Sandbox mode, you can also test your API calls to Amazon to ensure that the calls are configured correctly and that the responses include all the payment parameters that you need to track the entire order. Simulating error conditions helps you to better manage your buyer's experience in the event that something goes wrong during the checkout experience. In addition to testing approval scenarios, be sure to test decline workflows thoroughly to avoid the risk of filling an order when a buyer's card has been declined.

Note

- To enable Sandbox requests make sure that **isSandbox=true** parameter is added to the string to sign.

Sandbox Simulations

The following tables outline how certain responses can be simulated using Payment instruments provided by default in sandbox mode for every buyer.

Sandbox Payment Instruments

Cards

Issuer	3DSecure?	EMI?	Tail	ReasonCode
Visa	Y	N	0001	001 - SUCCESS
MasterCard	N	N	1001	001 - SUCCESS
MasterCard	Y	N	211	211 - Failure Received from Bank
Visa	Y	Y	1111	Json Specified
Visa	N	N	2222	Json Specified

Debit Card

Bank	3DSecure?	ReasonCode
CitiBank	Y	001 - SUCCESS

Net Banking

Bank	3DSecure?	ReasonCode
Axis Bank	Y	001 - SUCCESS
HDFC Bank	Y	211 - Failure received from bank
ICICI Bank	Y	229 - 3D Secure Verification failed
State Bank of India	Y	230 - Bank response Timed out
Andhra Bank	Y	101 - Transaction Timed out before sending to Bank
Bank of Bahrain and Kuwait	Y	100 - Amazon has classified the transaction as suspicious.

Bank of India	Y	103 - Merchant Not Active
Bank of Maharashtra	Y	104 - Merchant does not exist.
Canara Bank	Y	310 - Merchant time out exceeded

Taking Your Integration Live

You have been developing your integration in the Sandbox environment. To move your integration to a production environment you will need to perform the following steps.

Update the following URLs from Sandbox to Production:

Production Environment
https://mws.amazonservices.in/OffAmazonPayments/2013-01-01/
Seller central Integration settings URL
https://sellercentral.amazon.in/gp/pyop/seller/account/settings/user-settings-view.html/ref=py_pyopiset_dnav_home
APP and website Return URL whitelisting
Please get the URL whitelisted from your Integration specialist at Amazon.You can also reach out directly to apay-integration@amazon.com.

Important Prerequisites

Update your bank information on https://sellercentral.amazon.in/gp/on-board/configuration/single-section/global-bank.html where the settlement would happen

Request a Refund

You can issue full or partial refunds against previously successful payment.

After a refund is successfully processed, the Refund object is in a **Completed** state, and a Refund Notification email is sent to the buyer.

If the Refund is declined by Amazon, the Refund object is in a **Declined** state. See Declined Refunds.

Procedure

Request a refund by performing the following steps:

1. Make a call to the RefundPayment API.
2. Listen for the Refund IPN message returned by Amazon.

   Refund requests are not processed in real time and the initial **RefundStatus** is always **Pending**.

   Processing time varies and can be several hours. Amazon will notify you of the processing status via an IPN message.

3. Get Refund Details

   You can also query the details of a Refund request by calling the GetRefundDetails API using the **AmazonRefundId** that was returned with RefundPayment API response.

Full API Reference

ListOrderReference

Call the ListOrderReference API to query and check the details of a previously placed order.

This operation has a maximum request quota of 10 and a restore rate of one request every second in the production environment. It has a maximum request quota of two and a restore rate of one request every two seconds in the sandbox environment. For definitions of throttling terminology and for a complete explanation of throttling, see Throttling: Limits to how often you can submit requests in the *Amazon MWS Developer Guide*.

Request Parameters

For more information about the request parameters that are required for all Amazon MWS operations, see Required request parameters in the *Amazon MWS Developer Guide*.

Parameter Name	Required	Type	Description
PaymentDomain	Yes	xs:string	In our case the string constant IN_INR
QueryId	Yes	xs:string	Merchant generated order id that was sent to Amazon
QueryIdType	Yes	xs:string	In our case the string constant SellerOrderId

CreatedTimeRange.StartTime	No	xs:string	Start time in EPOCH for search
CreatedTimeRange.EndTime	No	xs:string	End time in EPOCH for search

Example Response

```
<ListOrderReferenceResponse
xmlns="http://mws.amazonservices.com/schema/OffAmazonPayments/2013-01-01">
 <ListOrderReferenceResult>
  <OrderReferenceList>
   <OrderReference>
    <ReleaseEnvironment>Live</ReleaseEnvironment>
    <OrderReferenceStatus>
     <ReasonDescription>Txn Success</ReasonDescription>
     <LastUpdateTimestamp>2017-11-29T04:29:08.599Z</LastUpdateTimestamp>
     <ReasonCode>UpfrontChargeSuccess</ReasonCode>
     <State>Closed</State>
    </OrderReferenceStatus>
    <AmazonOrderReferenceId>P04-2422439-6338041</AmazonOrderReferenceId>
    <CreationTimestamp>2017-11-29T04:27:46.232Z</CreationTimestamp>
    <SellerOrderAttributes>
     <SellerOrderId>test</SellerOrderId>
    </SellerOrderAttributes>
    <OrderTotal>
     <CurrencyCode>INR</CurrencyCode>
     <Amount>1.00</Amount>
    </OrderTotal>
   </OrderReference>
  </OrderReferenceList>
 </ListOrderReferenceResult>
 <ResponseMetadata>
  <RequestId>ae31bae0-718a-4b48-a0fb-afb6ba12f678</RequestId>
 </ResponseMetadata>
</ListOrderReferenceResponse>
```

RefundPayment

Call the RefundPayment API to refund a previously captured amount.

You call the GetRefundDetails operation to query the status of a refund.

This operation has a maximum request quota of 10 and a restore rate of one request every second in the production environment. It has a maximum request quota of two and a restore rate of one request every two seconds in the sandbox environment. For definitions of throttling terminology and for a complete explanation of throttling, see Throttling: Limits to how often you can submit requests in the *Amazon MWS Developer Guide*.

Request Parameters

For more information about the request parameters that are required for all Amazon MWS operations, see Required request parameters in the *Amazon MWS Developer Guide*.

Parameter Name	Required	Type	Description
AmazonTransactionIdType	Yes	xs:string	"OrderReferenceId"
AmazonTransactionId	Yes	xs:string	Id of the Amazon Order ID for which refund is to be initiated
RefundReferenceId	Yes	xs:string	The identifier for this refund transaction that you specify. This identifier must be unique for all your refund transactions. Amazon recommends that you use only the following characters: • lowercase a-z • uppercase A-Z • numbers 0-9 • dash (-) • underscore (_) Maximum: 32 characters
RefundAmount	Yes	Price	The amount to refund.
SellerRefundNote	No	xs:string	A description for the refund that is displayed in emails to the buyer. Maximum: 255 characters
SoftDescriptor	No	xs:string	The description to be shown on the buyer's payment instrument statement. The soft descriptor sent to

			the payment processor is: “AMZ* <soft descriptor specified here>”. Default: “AMZ*<SELLER_NAME> amzn.com/pmts WA” Maximum: 16 characters

Response Elements

Element Name	Description
RefundDetails	Encapsulates details about the Refund object and its status. Type: RefundDetails

RefundDetails

Encapsulates details about a Refund object and its status.

Datatype: content

Element Name	Description
AmazonRefundId	The Amazon-generated identifier for this refund transaction. Type: xs:string
RefundReferenceId	The identifier for this refund transaction that you specify. Maximum: 32 characters Type: xs:string
SellerRefundNote	A description for the refund that is displayed in emails to the buyer. Maximum: 255 characters Type: xs:string
RefundType	Indicates the refund type. Allowed values:

	• SellerInitiated Type: xs:string
RefundAmount	The amount requested for the refund. Type: Price
FeeRefunded	The capture fee that has been refunded. Type: Price
CreationTimestamp	The time at which the refund was created. In ISO 8601 format. Type: xs:dateTime
RefundStatus	Represents the status of the refund request. **Note:** The Refund operation always returns the **State** as *Pending*. The Refund object remains in this state until it is processed by Amazon. The refund processing time varies and can be several hours. After processing is complete, Amazon will notify you of the final processing status. For more information about the **State** and **ReasonCode** response elements, see Refund States and Reason Codes. Type: Status
SoftDescriptor	The description to be shown on the buyer's payment instrument statement. Maximum: 16 characters Type: xs:string

Status

Indicates the current status of a Refund object.

Datatype: content

Element Name	Description
State	Indicates the state that the Refund object. For more information, see Refund States and Reason Codes.

	Allowed values: *Pending* *Declined* *Completed* Type: xs:string
LastUpdateTimestamp	A timestamp that indicates the time when the authorization, capture, or refund state was last updated. In ISO 8601 format. Type: xs:dateTime
ReasonCode	The reason that the Refund object is in the current state. Refund States and Reason Codes. Type: xs:string
ReasonDescription	An optional description of the refund status. Maximum: 255 characters Type: xs:string

Example Query Request

End point to hit:

```
https://mws.amazonservices.in/
```

Request

```
GET /OffAmazonPayments/2013-01-
01?AWSAccessKeyId=AKI********L2YA&Action=RefundPayment&AmazonTransactionId=P04-
0591809-
2423722&AmazonTransactionIdType=OrderReferenceId&ContentType=XML&RefundAmount.Amount
=5&RefundAmount.CurrencyCode=INR&RefundReferenceId=YOUR_REFUND_ID&SellerId=A28RUGPV
UTQXU1&SellerRefundNote=Note&Signature=FbbcdJ%2BpV97gPrl93392KJnsv16ZWO7L%2BsohQZ2Lr
E0%3D&SignatureMethod=HmacSHA256&SignatureVersion=2&SoftDescriptor=softDescriptor&Times
tamp=2017-01-31T13%3A55%3A42.307Z []
```

Example Response

```
<RefundPaymentResponse
xmlns="http://mws.amazonservices.com/schema/OffAmazonPayments/2013-01-01">
  <RefundPaymentResult>
```

```
  <RefundDetails>
   <RefundDetail>
    <RefundReferenceId>test</RefundReferenceId>
    <RefundType>SellerInitiated</RefundType>
    <SellerRefundNote/>
    <CreationTimestamp>2017-04-17T06:54:47.837Z</CreationTimestamp>
    <RefundStatus>
     <LastUpdateTimestamp>2017-04-17T06:54:47.837Z</LastUpdateTimestamp>
     <State>Pending</State>
    </RefundStatus>
    <SoftDescriptor>AMZ*Test</SoftDescriptor>
    <FeeRefunded>
     <CurrencyCode>INR</CurrencyCode>
     <Amount>0.00</Amount>
    </FeeRefunded>
    <AmazonRefundId>S04-2665653-4222901-R066827</AmazonRefundId>
    <RefundAmount>
     <CurrencyCode>INR</CurrencyCode>
     <Amount>1.00</Amount>
    </RefundAmount>
   </RefundDetail>
  </RefundDetails>
 </RefundPaymentResult>
 <ResponseMetadata>
  <RequestId>ebe60e12-a824-4002-bfe4-1c783bd95c12</RequestId>
 </ResponseMetadata>
</RefundPaymentResponse>
```

GetRefundDetails

Call the GetRefundDetails operation to query the status of a particular refund. If you received a **Pending** status when you called the Refund operation, you can call this operation to get the current status.

This operation has a maximum request quota of 20 and a restore rate of two requests every second in the production environment. It has a maximum request quota of five and a restore rate of one request every second in the sandbox environment. For definitions of throttling terminology and for a complete explanation of throttling, see Throttling: Limits to how often you can submit requests in the *Amazon MWS Developer Guide*.

Request Parameters

For more information about the request parameters that are required for all Amazon MWS operations, see Required request parameters in the *Amazon MWS Developer Guide*.

Parameter Name	Required	Type	Description
AmazonRefundId	Yes	xs:string	The Amazon-generated identifier for this refund transaction.

Response Elements

Element Name	Description
RefundDetails	Encapsulates details about the Refund object and its status. Type: RefundDetails

Declined Refunds

Handling Refund Declines

If a Refund request has a status of **Declined**, you will see one of the following reason codes in the response:

1. AmazonRejected

   Amazon has declined the refund because the maximum amount has already been refunded on the capture. You need to work with the buyer to issue a refund in an alternate way, such as issuing a gift card or a store credit.

2. ProcessingFailure

   Amazon could not process the transaction due to one of the following:

   - There was an internal processing error.

     If you receive a **ProcessingFailure** reason code check the A-to-z claim and chargeback status for your order in Seller Central, by clicking **Performance**.

     If no A-to-z claim or chargeback refund has been issued to the buyer and the payment is still in the **Completed** state, retry your request in one to two minutes. If the retry attempt does not succeed, contact the buyer and issue the refund in an alternative way.

Handling Errors from Amazon Pay API Calls

When you get an error from the Amazon Pay API section operations, you may be able to retry that operation depending on the nature of the error. Properly formed operation requests are idempotent — that is, if you call the same operation on an already successful request, you will not create a duplicate transaction.

For refund requests, the idempotency key is the **RefundReferenceId**. For example, you cannot refund against the same payment more than once if you provide the same **RefundReferenceId**. This functionality allows you to safely retry any operation call if the error meets the conditions described in the table below.

If you want to retry an operation call after you receive an error, you can immediately retry after the first error response. If you want to retry multiple times, Amazon recommends that you implement an "exponential backoff" approach up to some limit. Then, log the error and proceed with a manual follow-up and investigation. For example, you can time your retries in the following time spacing: 1s, 2s, 4s, 10s, and 30s. The actual backoff times and limit will depend upon your business processes.

Error Code	**HTTP Status Code**	**Is Retriable (Y/N)**	**Description**
MarketplacePaymentDuplicateReferenceIdException	400	N	When referenceId provide in request is redundant
MarketplacePaymentCaptureNotRefundableException	405	N	When the refund can not be initiated on the capture of the order
MarketplacePaymentInvalidParameterValueException	400	N	Invalid parameter value passed in the request
MarketplacePaymentInvalidSellerIdException	400	N	Invalid sellerId is passed
MarketplacePaymentInvalidAccountStatusException	403	N	Seller account status is invalid
MarketplacePaymentInvalidTransactionIdException	404	N	Invalid transaction id is provided
MarketplacePaymentServiceException	500	N	Internal server error occurred while processing the request
MarketplacePaymentServiceUnavailableException	503	Y	Service is unavailable
MarketplacePaymentTransactionAmountExceededException	400	N	Refund amount has exceeded transaction amount.

MarketplacePaymentTransactionCountExceededException	400	N	Number of refund transactions on an order has been exhausted.
MarketplacePaymentInvalidSandboxSimulationException	400	N	Invalid sandbox simulation is provided in the request

The following table describes re-triable errors:

HTTP Code	HTTP Status	Description and Corrective Action
500	InternalServerError	The server encountered an unexpected condition which prevented it from fulfilling the request. This error is safe to retry. This error can occur if you send a request for a sandbox session to the production end points or vice versa.
502	Bad Gateway	A server between the client and the destination server was acting as a gateway or proxy and rejected the request. In many cases this is an intermittent issue and is safe to retry. If the issue persists, please contact your network administrator.
503	ServiceUnavailable RequestThrottled	There was an unexpected problem on the server side or the server has throttled you for exceeding your transaction quota. For a complete explanation of throttling, see Throttling: Limits to how often you can submit requests in the *MWS Developer Guide*. The client should retry the request or reduce the frequency of requests.
504	Gateway Timeout	A server between the client and the destination server was acting as a gateway or proxy and timed out on the request. In many cases this is an intermittent issue and is safe to retry.

Instant Payment Notification (IPN)

Handling Instant Payment Notification (IPN) messages

After Amazon processes the payment/refund request, the status of the corresponding payment/refund objects will be updated. While you might check for any payment/refund request periodically requesting it from our API, you can also set up a listener and receive notifications via Instant Payment Notification (IPN) messages.

IPN messages are sent by Amazon Payments without any action required on your part and can be used to update your internal order management system and process orders.

The following is a list of notifications that are available from Amazon:

- ChargeNotification
- RefundNotification

For more information about the format of the Instant Notification, see HTTP/HTTPS Notification JSON Format in the Amazon SNS Getting Started Guide, which is available from the Amazon AWS Documentation portal.

The contents of the **NotificationData** member that is returned in the Instant Notification are described by the following publicly available XSD:
https://s3-us-west-2.amazonaws.com/amazonpay/ipn/payments_ipn.xsd.

Setting up to receive IPN messages

An IPN message is an HTTPS POST request containing the XML-based notification data in its body. Before you can receive IPN messages you must do the following:

1. Set up endpoints in Seller Central.
    a. To set up endpoints, log in in to Seller Central and select **Integration Settings** from the **Settings** drop-down box.
    b. Click **Edit** under the **Instant Notifications Settings** section, and enter your **Merchant URL** on your endpoint where you want to receive instant notifications. Note that you need to use HTTPS for production, but for the Sandbox environment HTTP is acceptable as an alternative to HTTPS. However, if you specify an HTTPS endpoint, the certificate must be valid and have an intact certificate chain up to a root certificate.
    c. If you need to receive the IPNs for the same event on two distinct endpoints you can provide the URL in the **Integrator URL**.
2. Check your endpoint and SSL certificates to ensure they are working properly.
3. Verify that you are running a web service that can receive HTTPS POST requests made to your endpoint and process the notifications. Remove potential access protections, such as .htaccess rules requiring a user login.
4. Verify that your HTTPS uses valid SSL certificates from a trusted certificate provider. For more information see SSL Certification.

Sample notifications

The following is a list of notifications available from Amazon:

- **ChargeNotification**
- **RefundNotification**

ChargeNotification

The following example shows the **ChargeNotification**:

```
POST /SPN_project2/iopn HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 4227aa54-ccf8-5a2a-8038-fb740d9f65d6
x-amz-sns-topic-arn: arn:aws:sns:eu-west-1:598607868003:A18VPDB9UTK24DA3GEDG4FJC14BQ
x-amz-sns-subscription-arn: arn:aws:sns:eu-west-1:598607868003:A18VPDB9UTK24DA3GEDG4FJC14BQ:993a0851-1b8d-4e0c-a48a-c4b2cbd17036
Content-Length: 2301
Content-Type: text/plain; charset=UTF-8
Host: ded73b97.ngrok.io
User-Agent: Amazon Simple Notification Service Agent
Accept-Encoding: gzip,deflate
X-Forwarded-Proto: https
X-Forwarded-For: 54.240.197.7

{
  "Type" : "Notification",
  "MessageId" : "4227aa54-ccf8-5a2a-8038-fb740d9f65d6",
  "TopicArn" : "arn:aws:sns:eu-west-1:598607868003:A18VPDB9UTK24DA3GEDG4FJC14BQ",
  "Message" : "{\"ReleaseEnvironment\":\"Live\",\"MarketplaceID\":\"220451\",\"Version\":\"2013-01-01\",\"NotificationType\":\"OrderReferenceNotification\",\"SellerId\":\"A3GEDG4FJC14BQ\",\"NotificationReferenceId\":\"f80ab4f0-82ca-42c8-a0d1-9b07f5b3fa30\",\"Timestamp\":\"2017-02-17T09:15:18.679Z\",\"NotificationData\":\"<?xml version=\\\"1.0\\\" encoding=\\\"UTF-8\\\"?><ChargeTransactionNotification
```

xmlns=\\\"https://mws.amazonservices.com/ipn/OffAmazonPayments/2013-01-01\\\">\\n
<ChargeTransactionDetails>\\n <OrderID>P04-5366666-6431174<\\/OrderID>\\n
<SellerReferenceId>test<\\/SellerReferenceId>\\n <Amount>\\n
<Amount>10.0<\\/Amount>\\n <CurrencyCode>INR<\\/CurrencyCode>\\n <\\/Amount>\\n
<TotalFee>\\n <Amount>0.0<\\/Amount>\\n <CurrencyCode>INR<\\/CurrencyCode>\\n
<\\/TotalFee>\\n <PaymentModes/>\\n <FeeBreakup/>\\n <CreationTimestamp>2017-
02-17T09:00:13.592Z<\\/CreationTimestamp>\\n <Status>\\n
<State>Declined<\\/State>\\n <LastUpdateTimestamp>2017-02-
17T09:15:13.879Z<\\/LastUpdateTimestamp>\\n
<ReasonCode>SessionExpired<\\/ReasonCode>\\n <ReasonDescription>Session
Expired<\\/ReasonDescription>\\n <\\/Status>\\n
<\\/ChargeTransactionDetails>\\n<\\/ChargeTransactionNotification>\"}",

"Timestamp" : "2017-02-17T09:15:19.922Z",

"SignatureVersion" : "1",

"Signature" :
"FIRgFXytZTrpt4axHOHqVto+hbXadKhCnP2gfGaII3+6Jnawz939iT/KW4Z8wVYed3s+EGtC+xM3JCBVNJ5
m7Ctf4bZZ9rFy+7Y7hAS/c18J1bNeEbEz2l0WQvpI4MDzH5/mmSVEWawfwX6zPE0R9U9kT81hac7a/NR
edbUnJpOQCytCbTHxCn/k1s4WQQpXwIPnOVyp0x3Dj7ofkhJNB7bZk2bQET22DaOpSg01I4/KTU5t1iFzY
VeoVRa3BcnB+X9d5GEdbmKjGg0SHhVSkzq0Qx3cpcipiyXzqv1IR62wxlpVC1yYkGXiw5uNU9k8QIweAoO
4TuzR1IwYakTO3g==",

"SigningCertURL" : "https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-
b95095beb82e8f6a046b3aafc7f4149a.pem",

"UnsubscribeURL" : "https://sns.eu-west-
1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:eu-west-
1:598607868003:A18VPDB9UTK24DA3GEDG4FJC14BQ:993a0851-1b8d-4e0c-a48a-c4b2cbd17036"

}

RefundNotification

The following example shows the **RefundNotification**:

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 5f43584c-1f96-5880-9c98-119f5EXAMPLE
x-amz-sns-topic-arn: arn:aws:sns:EXAMPLE:59860EXAMPLE:TestTopic
x-amz-sns-subscription-arn:
arn:aws:sns:EXAMPLE:59860EXAMPLE:TestTopic: EXAMPLE
Content-Length: 961
Content-Type: text/plain; charset=UTF-8
Host: ec2-EXAMPLE.compute-1.amazonaws.com
```

```
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
{
   "Type" : "Notification",
   "MessageId" : "1c53034e-051f-5fe2-a4e7-4d17c21104eb",
   "TopicArn" : "arn:aws:sns:us-east-1:291180941288:A3BXB0YN3XH17HA2K7NDRCTOTPW9",
   "Message" : "{"ReleaseEnvironment":"Live","MarketplaceID":"220451","Version":"2013-01-
01","NotificationType":"PaymentRefund","SellerId":"A28RUGPVUTQXU1","NotificationRef
erenceId":"8fc521ee-ba9d-40c9-a64d-be10117e3f58","Timestamp":"2017-01-
30T10:50:16.880Z","NotificationData":"<?xml version=\"1.0\" encoding=\"UTF-
8\"?><RefundTransactionNotificatio
n xmlns=\"https://mws.amazonservices.com/ipn/OffAmazonPayments/2013-01-01\">\n
<RefundTransactionDetails>\n      <RefundTransactionId>P04-3975959-6325632-
R067318<\/RefundTran
sactionId>\n      <RefundReferenceId>test<\/RefundReferenceId>\n
<RefundType>SellerInitiated<\/RefundType>\n      <PaymentModes>\n          <Id>NetBanking<\/Id>\
n      <\/PaymentModes>\n      <Amount>\n          <Amount>5.0<\/Amount>\n
<CurrencyCode>INR<\/CurrencyCode>\n      <\/Amount>\n      <Fee>\n          <Amou
nt>0.0<\/Amount>\n          <CurrencyCode>INR<\/CurrencyCode>\n      <\/Fee>\n
<CreationTimestamp>2017-01-30T10:47:45.506Z<\/CreationTimestamp>\n      <Status>\n
      <State>Completed<\/State>\n          <LastUpdateTimestamp>2017-01-
30T10:50:11.109Z<\/LastUpdateTimestamp>\n      <\/Status>\n
<\/RefundTransactionDetails>\n<\/Refund
TransactionNotification>"}",
   "Timestamp" : "2015-08-28T02:17:50.722Z",
   "SignatureVersion" : "1",
   "Signature" :
"gY8bJ8JWxw1jW28Pg3U9CDheqfPYSAYHpLPKyVZwNoc3epJ0YUvAyqs+QKslPAmRm2Dgeb5acDWgEG
U+WulcWyK1s7s5uz+Ngr12zyWaZLbvtOuEoimTYZQHsWQNsjtTyjv+Huj4Lm81QOlJs3DSX19N9SMYSiLb
hODVo6N+bkuELZBoT5QSx+y9+tb/AEK2zKdbwFwXcCKZryu+igj69hiyDSyKYRRFDs7Zxert7EWGWIQaJB
anHV3vggs+slAUDz7RrAgmJwQ2ktrH89/BQk4GGDT5Z9d6Xu7Mt7VCWkv1E3PU4C5dg7wSVD4+WAt5
23436llHrNVgD7N39cKl9g==",
   "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-
bb750dd426d95ee9390147a5624348ee.pem",
   "UnsubscribeURL" : "https://sns.us-east-
1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-
1:291180941288:A3BXB0YN3XH17HA2K7NDRCTOTPW9:05542723-375e-4609-98f1-8abcf427d95f"
}
```

Responding to an IPN message

With each notification, you receive, you should configure your endpoint to send Amazon a "200 OK" response immediately after receipt. If you do not send this response or if your server is down when the SNS message is sent, Amazon SNS will perform retries every hour for 14 days.

Note: Most of the time each notification will be delivered to your application exactly once but, due to the distributed nature of Amazon SNS and transient network conditions, there could be an occasional, duplicate message. Be sure to design your application so that if multiple IPN messages are received your payment workflow will not break.

Secure IPN processing

To prevent spoofing attacks you must validate the IPN signature to verify the authenticity of the message. For more information see Verifying the Signatures of Amazon SNS Messages.

If you are using an Amazon SDK this is validation is handled for you. If not, use the same validation steps as described in Verifying the Signatures of Amazon SNS Messages and as implemented in the SDKs.

TLS/SSL certification

Introduction to TLS/SSL

Amazon recommends that you always use a secure connection. However, there are instances when your server is required to have a valid TLS/SSL certificate, issued by a trusted Certificate Authority:

- **IPN messages** — IPN (Instant Payment Notification) messages can only be sent to a secure endpoint. Without a valid certificate Amazon cannot tell whether the server receiving the IPN messages actually belongs to the merchant or to somebody who is attempting to intercept the data.

What is TLS/SSL?

Transport Security Layer ("TLS") and Secure Sockets Layer ("SSL") are protocols designed to ensure that data can be securely transported between a web server and a browser, using cryptographic algorithms. TLS/SSL ensures that the data transmitted comes from the source it claims to be coming from, and that it has not been modified or read by a 3rd party during the transmission. For additional details on versions of TLS/SSL that we support, see TLS/SSL frequently asked questions.

HTTP versus HTTPS

When a URL address contains begins with the HTTPS protocol, the ‘S’ stands for secure, and it indicates that data is being transmitted securely. The difference between HTTP and HTTPS is that in HTTPS the data is transferred on top of TLS/SSL protocols, and therefore inherits all of its security.

TLS/SSL certificates

TLS/SSL uses certificates to secure and protect transmitted data. A certificate contains information about the owner of the certificate, such as the organization, country, duration of validity, website address, and the certificate ID of the person who certifies (signs) this information. It also contains the public key and a hash to ensure that the certificate has not been tampered with.

Here is a sample certificate:

```
Company Root CA 9
==================
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgITBmyfz5m/jAo54vB4iXxxababbmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468Sample
-----END CERTIFICATE-----
```

Certificate chains

Large global Certificate Authorities (CAs) certify other agencies to issue TLS/SSL certificates, which usually operate at a regional level. If a server's certificate was issued by an intermediate CA, the server must also host the intermediate CA's certificate which, in turn, can be verified against a trusted root certificate that is stored locally.

Here are the steps for verifying a chain:

Download the certificates from the server.

Check to see if the server certificate matches the website name and is signed by the intermediate certificate.

Check to see if the intermediate certificate is signed by one of the trusted root certificates stored locally.

Intermediate CAs can issue certificates to other intermediate CAs; this means that the certificate chain can be longer than 3 certificates.

Why are TLS/SSL certificates needed?

Here are some reasons for using a TLS/SSL certificate:

Security: The primary reason for using a TLS/SSL certificate is to keep the data that is exchanged between a buyer’s browser and your server secure. This prevents order and payment details and buyer data — such as the buyer’s username and password — from being exposed to the internet and intercepted.

Buyer trust: When you obtain a TLS/SSL certificate, the Certificate Authority will issue a seal to be displayed on your web page. This seal instills trust in your website because buyers know that their data is secure. The image below shows some sample seals:

Traffic: Search engines such as Google rank eCommerce sites that operate over insecure connections lower than sites that use secure connections.

TLS/SSL certificates and Amazon requirements

MD5 with RSA restriction

Amazon checks the bottom of the certificate chain (usually a domain such as example.com) and will not establish a TLS/SSL connection with a site using a Certificate Signature Algorithm that uses MD5 with RSA Encryption.

Examining your site's encryption algorithm

To examine your site's encryption algorithm, follow these steps.

In Firefox

1. Go to your site using the https:// secure protocol.
2. Click the site icon to the left of the domain name. This brings up an information dialog box about the host.
3. Click the **More Information** button to display the **Page Info** dialog box.
4. Click the **Security** icon, and then click the **View Certificate** button to display the **Certificate Viewer** dialog box.
5. Click the **Details** tab, and then, under the **Certificate Fields** list box, scroll down and click **Certificate Signature Algorithm** to display the Field Value. The **Field Value** box displays the certificate algorithm that is used.
6. If the Field Value is **MD5 With RSA Encryption**, the certificate is not valid for use with Amazon Payments transactions.

In Internet Explorer

1. Go to your site using the https:// secure protocol.

2. Click the security icon (a lock) to the right of the domain name. This brings up the **Website Identification** pop-up window.
3. Click **View Certificates** to display the **Certificate** dialog box.
4. Click the **Details** tab to display the Signature algorithm that is used.
5. If the Signature algorithm value is **md5RSA**, the certificate is not valid for use with Amazon Payments transactions.

In Safari

1. Go to your site using the https:// secure protocol.
2. Click the security icon (a lock) to the left of the domain name. This brings up a digital certificate pop-up window.
3. Click **Show Certificate**, and the window expands to display additional information.
4. Click the arrow to the left of **Details** and scroll down to view the Signature algorithm that is used.
5. If the Signature algorithm value is **md5RSA**, the certificate is not valid for use with Amazon Payments.

Common TLS/SSL errors

Missing intermediate certificate

A missing intermediate certificate can occur when the certificate is installed correctly, but the server does not store the intermediate certificate. When this happens, the chain of trust cannot be established. To prevent this problem, ensure that all of the certificates in the chain are stored locally.

Certificate name mismatch

This error occurs when the name on the installed certificate is different from the web site's address (the installed certificate belongs to a different website). To resolve this error, you need to purchase a new certificate for the website.

Purchasing a certificate

Certificates can be purchased over the internet from any number of hosting companies.

TLS/SSL certificates are issued and maintained by a network of Certificate Authorities (CAs). These are usually well-known companies from the IT sector that adhere to strict security standards.

There are two types of certificates, standard and extended. The standard version meets the requirements for Login and Pay with Amazon, it is less expensive than an extended type, and it is issued within minutes of purchase.

Certificate prices vary. Note that many certificates come with a 30-day free trial, and can be revoked at no cost if you aren't happy.

Amazon-approved TLS/SSL certificates

Pay with Amazon currently accepts TLS/SSL certificates with root certificates from any of the Certificate Authorities (CAs) listed on the Certificate Authorities (CA) Recognized by Amazon SNS for HTTPS Endpoints page.

Contact

For any technical issues you can reach out directly to apay-integration@amazon.com.

Appendix

Registering with Amazon Payments

Style and button guidelines

To complete integrating Amazon Pay, you'll display Payment Button images or cascading style sheets (CSS) that your buyers will click to successfully place orders.

You can choose from several buttons styles based upon the size of the button or the color of the button itself or its background.

Colors

In color

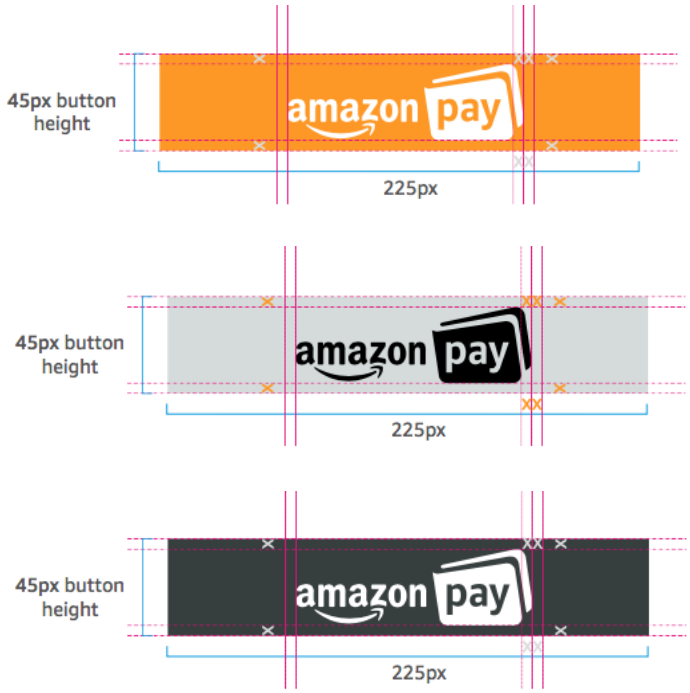
HEX #FF9900

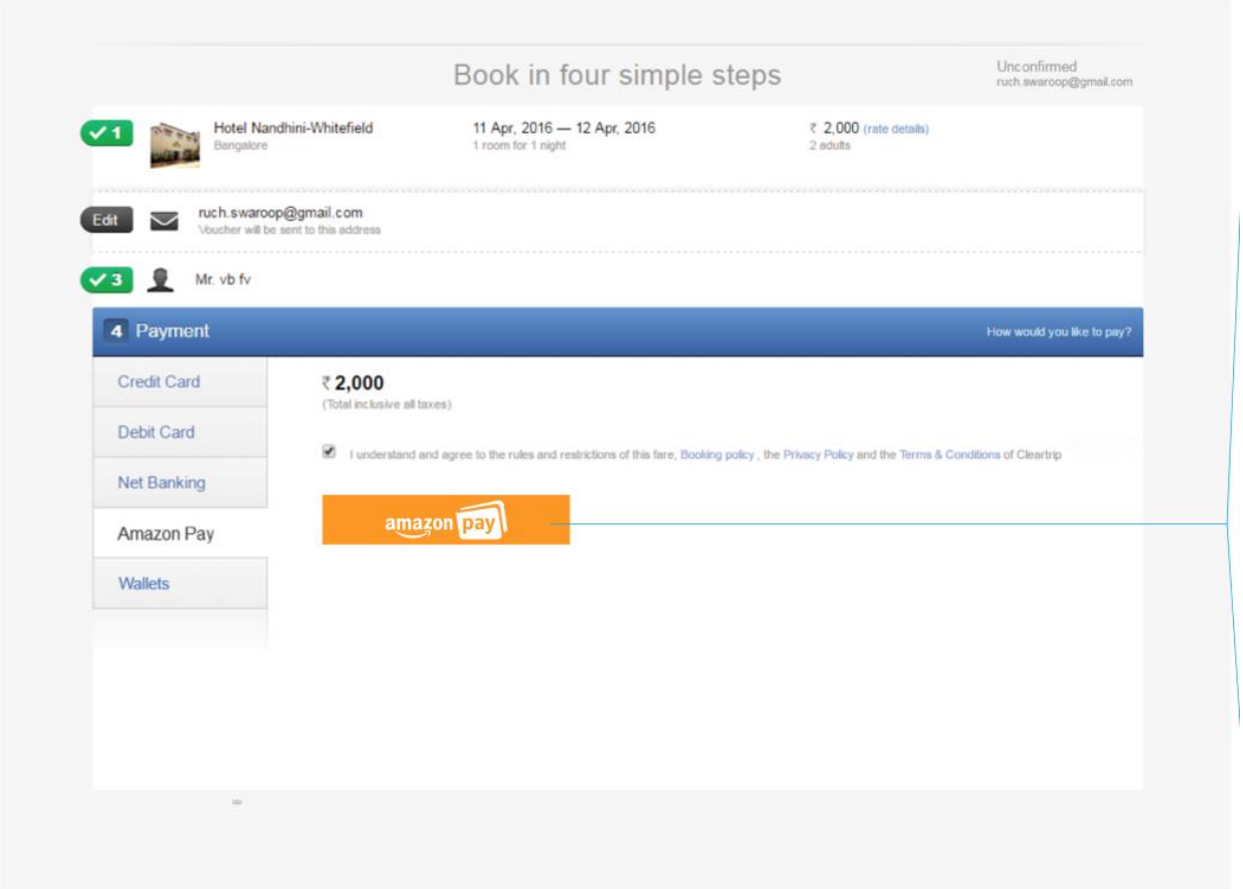
Monochrome

HEX #000000

Logo Usage (Background Color)

Button color & specification





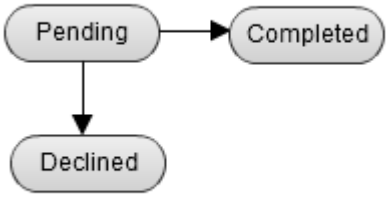
Reason Codes

reasonCode	Description
0	NONE
01	Pending
001	Success
229	3d Secure Verification Failed
211	Failure Received From Bank
100	Amazon has classified the transaction as suspicious
301	Simulation data provided is not valid
302	Simulation data provided is not valid for selected instrument
230	Bank response Timed out

231	User response timed out failure
101	Transaction Timed out before sending to bank
310	Merchant time out exceeded
103	Merchant Not Active
104	Merchant does not exist

Refund States and Reason Codes

The following diagram depicts the state transition of a Refund object:



REFUND State Transitions

The following table describes each refund state in detail, the allowed operations on a state, and the reasons why a Refund object can end up in a state.

State	Description	Allowed Operations	Reason Codes
Pending	A Refund object is in the **Pending** state until it is processed by Amazon.	GetRefundDetails	--
Declined	Amazon has declined the refund because the maximum amount has been refunded on the capture.	GetRefundDetails	**AmazonRejected**—Amazon has rejected the refund. You should issue a refund to the buyer in an alternate manner (for example, a gift card or store credit). **ProcessingFailure**—Amazon could not process the

			transaction due to an internal processing error or because the buyer has already received a refund from an A-to-z claim or a chargeback. You should only retry the refund if the Capture object is in the **Completed** state. Otherwise, you should refund the buyer in an alternative way (for example, a store credit or a check).
Completed	The refund request has been processed and funds will be returned to the buyer.	GetRefundDetails	--

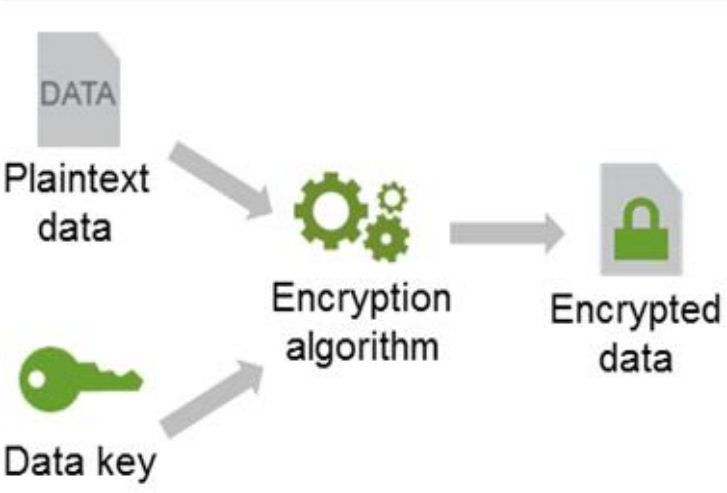
Encryption Concepts

Topics

- Encryption Basics
- Envelope Encryption

Encryption Basics

To encrypt data, you provide the raw data (*plaintext*) and a *data key* to an encryption algorithm. The algorithm uses those inputs to produce encrypted data (*ciphertext*).

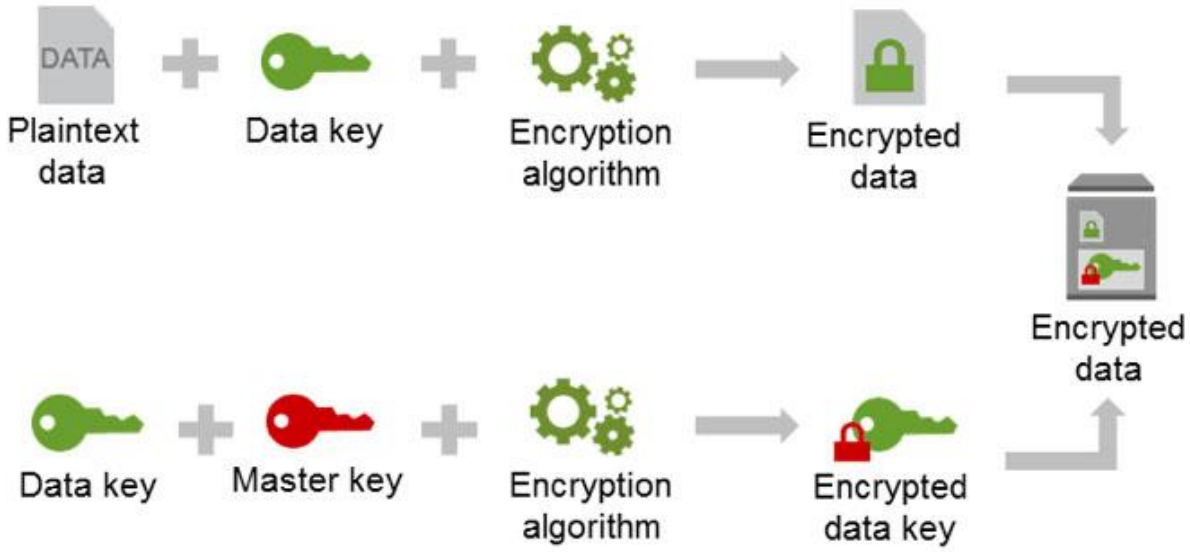


The security of your encrypted data depends on protecting the data key that can decrypt it. One accepted best practice for protecting the data key is to encrypt it. To encrypt the data key you need another encryption key called a *key encryption key* (KEK). This practice of using KEKs to encrypt data keys is called *envelope encryption*.

Envelope Encryption

Envelope encryption is the practice of encrypting plaintext data with a unique data key, and then encrypting the data key with a KEK. You might choose to encrypt the KEK with another KEK, and so on, but eventually you must have a *master key*. The master key is an unencrypted (plaintext) key with which you can decrypt one or more other keys.

The following image provides an overview of envelope encryption. In this scenario, the data key is encrypted with a single KEK, which is the master key.



Note

- Amazon Pay requires AES-GCM-No Padding algorithm for Plaintext data encryption and RSA/ECB/OAEPWithSHA-1AndMGF1Padding Algorithm for Data key encryption.

Dynamic config

The public key/master key for encryption can be dynamically fetched by making GET request to https://amazonpay.amazon.in/getDynamicConfig?key=serverSideSDKJava

Sample response:

```
{
  "publicKey":
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDcW7r1yKMtr+P3ZU3IaC6raMCbg7Wo1jEP4C
Ta36ByIjeI4eNAmiozmwqwwx3W5k+gPH/KI4lZojWYZPswfDz7LNF2q8jnSSzNHrdgw2vtyc63X
L+h1kZ9hdBc+zv7QEnnEr1OxxN/EWGOR3PXGypBRT83udG6eHbG5vNlGAhlXwIDAQAB",
  "metricsPublishingTimeInterval": "3600"
}
```