****

|  |
| --- |
| Technical Process Document for API Integration |
| Corporate Payment APIs with XML Payload |
| Version 1.0 |
| **Snehal Ghosalkar** |
| **18/04/2022** |

Document Control

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Name :** | HDFCBank\_CorporatePayments\_API Integration\_TechnicalProcessDoc\_XMLSignatures(HIGH) | **Version :** | 1.0 |
| **Template Owner :** | Information Technology | **Prepared By :** | Snehal Ghosalkar |
| **Date :** | 18/04/2022 | **Approved By :** | Vinod Rasquinha |

Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version Number** | **Date** | **Author(s)** | **Brief Description of Change** | **Reviewed by** |
| 1.0 | 18/04/2022 | Snehal Ghosalkar | Initial Version | Vinod Rasquinha |

Table of Contents

[1. Introduction 4](#_Toc101808268)

[1.1 Purpose 4](#_Toc101808269)

[1.2 Intended Audience 4](#_Toc101808270)

[2. Partner API Integration Development Process Workflow Overview 5](#_Toc101808271)

[3. How to consume the service 6](#_Toc101808272)

[3.1 Request: 6](#_Toc101808273)

[3.1.1 XML Request Payload - Generic Sample 6](#_Toc101808274)

[3.1.2 Standard JSON Request Payload Structure: 8](#_Toc101808275)

[3.1.1 Detailed description of the Standard JSON Request Payload fields: 9](#_Toc101808276)

[3.2 Response: 15](#_Toc101808277)

[3.2.1 Standard JSON Response Payload Structure: 15](#_Toc101808278)

[3.2.2 Receiving response payload will look like: 15](#_Toc101808279)

[3.2.3 Detailed description of the Standard JSON Response Payload fields: 16](#_Toc101808280)

[4. Certificate Prerequisites 20](#_Toc101808281)

[5. Error/Status Codes Details 20](#_Toc101808282)

[ HDFC Bank’s Public API Gateway Error Codes:- 20](#_Toc101808283)

[6. Appendix 21](#_Toc101808284)

[ Other Prerequisites required from External partner: 21](#_Toc101808285)

[ API Key Details:- 21](#_Toc101808286)

[ Sample Code Snippets for Client-Side API integration 23](#_Toc101808287)

[1.1.1. Code Snippet for Client-Side Encryption for the request payload field RequestSignatureEncryptedValue: 23](#_Toc101808288)

[1.1.2. Code Snippet for Client-Side Encryption for the request payload field SymmetricEncryptedValue 24](#_Toc101808289)

[ Glossary: 25](#_Toc101808290)

# Introduction

## 1.1 Purpose

The purpose of this document is to provide external partners of HDFC Bank with technical details/steps that are required to integrate with an HIGH Criticality API in HDFC Bank’s API Journey.

HDFC Bank’s HIGH Criticality API requires the request payload to be digitally signed and encrypted at partner’s end.

This document elaborates on the process that need to be followed by participating external API partners for development of digital signature, encryption and OAuth Token in order to implement required security as outlined in the HDFC Bank’s Partner API Governance Guidelines.

The APIs are available on public domain at [https://api-uat.hdfcbank.com/\*](https://api-uat.hdfcbank.com/*)

The API follows a REST JSON format.

## 1.2 Intended Audience

This document is intended for use by system designers and developers of various partners/consumers of HDFC Bank APIs to understand technical aspects of API Integration in detail.

# Partner API Integration Development Process Workflow Overview

A screenshot of a computer

Description automatically generated with medium confidence

# How to consume the service

## **3.1 Request:**

This section has utilized a generic sample XML request payload (given in section 3.1.1 below), in order to guide partners, by providing detailed steps to be followed during development of the API request.

However, partners need to use the original request payload provided in HDFC Bank API User Manual under section 3.1.1, instead of the generic sample request payload.

### 3.1.1 XML Request Payload - Generic Sample

<faxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="CO\_BD2.xsd">

<header>

<extsysname>COAPI</extsysname>

<datpost>2017-11-15</datpost>

<batchnumext>405734</batchnumext>

<idtxn>CO\_BD2</idtxn>

<codcurr>INR</codcurr>

<iduser>SL711045249778</iduser>

<idcust>33188831</idcust>

<groupid>BAJAJFIN</groupid>

<reqdatetime>2017-11-15T06:00:07</reqdatetime>

</header>

<summary>

<orgcountdr>1</orgcountdr>

<orgcountcr>3</orgcountcr>

<orgsumdr>12000.00</orgsumdr>

<orgsumcr>12000.00</orgsumcr>

</summary>

<debit>

<stanext>1</stanext>

<accountno>11362560000112</accountno>

<orgamount>12000.00</orgamount>

<txndesc>Salary Oct 2017</txndesc>

<referenceno>711045249778</referenceno>

</debit>

<creditlist>

<credit>

<stanext>2</stanext>

<accountno>50100093870971</accountno>

<amount>7000.00</amount>

<txndesc>Salary Oct 2017</txndesc>

<referenceno>711045249833</referenceno>

<beneid></beneid>

<BeneName>SRINIVAS MOTORS</BeneName>

<BeneAddress\_1/>

<BeneAddress\_2/>

<BeneAddress\_3/>

<BeneAddress\_4/>

<emailid>a@abc.com, b@abc.com,c@abc.com</emailid>

<advice1></advice1>

<advice2></advice2> <!--This is advice field -->

<advice3></advice3>

<advice4></advice4>

<advice5></advice5>

<advice6></advice6>

<advice7></advice7>

<advice8></advice8>

<advice9></advice9>

<advice10></advice10>

<addnlfield1></addnlfield1>

<addnlfield2></addnlfield2>

<addnlfield3></addnlfield3>

<addnlfield4></addnlfield4>

<addnlfield5></addnlfield5>

</credit>

<credit>

<stanext>3</stanext>

<accountno>50100093870971</accountno>

<amount>3000.00</amount>

<txndesc>Salary Oct 2017</txndesc>

<referenceno>711045249834</referenceno>

<beneid>AMIRK</beneid>

<BeneName>ABCD</BeneName>

<BeneAddress\_1/>

<BeneAddress\_2/>

<BeneAddress\_3/>

<BeneAddress\_4/>

<emailid>d@abc.com,e@abc.com</emailid>

<advice1>Payment for month of March</advice1>

<advice2>Amount transferred to the account registered in the system</advice2>

<advice3>You may check the salary statement from 3rd of this month</advice3>

<advice4></advice4>

<advice5></advice5>

<advice6></advice6>

<advice7></advice7>

<advice8></advice8>

<advice9></advice9>

<advice10></advice10>

<addnlfield1></addnlfield1>

<addnlfield2></addnlfield2>

<addnlfield3></addnlfield3>

<addnlfield4></addnlfield4>

<addnlfield5></addnlfield5>

</credit>

<credit>

<stanext>4</stanext>

<accountno>50100093870971</accountno>

<amount>2000.00</amount>

<txndesc>Salary Oct 2017</txndesc>

<referenceno>711045249835</referenceno>

<beneid>RAJMOTORS</beneid>

<BeneName>RAJ MOTORS</BeneName>

<BeneAddress\_1/>

<BeneAddress\_2/>

<BeneAddress\_3/>

<BeneAddress\_4/>

<emailid>d@abc.com,e@abc.com</emailid>

<advice1>Payment for month of March</advice1>

<advice2>Amount transferred to the account registered in the system</advice2>

<advice3>You may check the salary statement from 3rd of this month</advice3>

<advice4></advice4>

<advice5></advice5>

<advice6></advice6>

<advice7></advice7>

<advice8></advice8>

<advice9></advice9>

<advice10></advice10>

<addnlfield1></addnlfield1>

<addnlfield2></addnlfield2>

<addnlfield3></addnlfield3>

<addnlfield4></addnlfield4>

<addnlfield5></addnlfield5>

</credit>

</creditlist>

</faxml>

The digitally signed and encrypted equivalent values of the Original REST-XML Request Sample needs to be passed as values of Standard JSON Request Payload fields to HDFC Bank’s REST API.

External partners should not send the Original Request Sample in an as-is manner, the sample request above needs to be passed as part of the standard JSON format listed below.

### 3.1.2 Standard JSON Request Payload Structure:

{

"RequestSignatureEncryptedValue": "",

"SymmetricKeyEncryptedValue": "",

"Scope": "",

"TransactionId": "",

"OAuthTokenValue": ""

}

The individual fields of the Standard JSON Request Payload are briefly described below-

1. RequestSignatureEncryptedValue: This will contain the base64 encoded encrypted value of IV + digitally signed XML request sample.

2. SymmetricKeyEncryptedValue: This will contain the base64 encoded encrypted value of a 32 byte symmetric key used for encrypting in 1.

3. Scope: This field should be the exact value which was set on the HDFC Bank’s API portal while registering the consumer application and indicates which external partner is invoking the API.

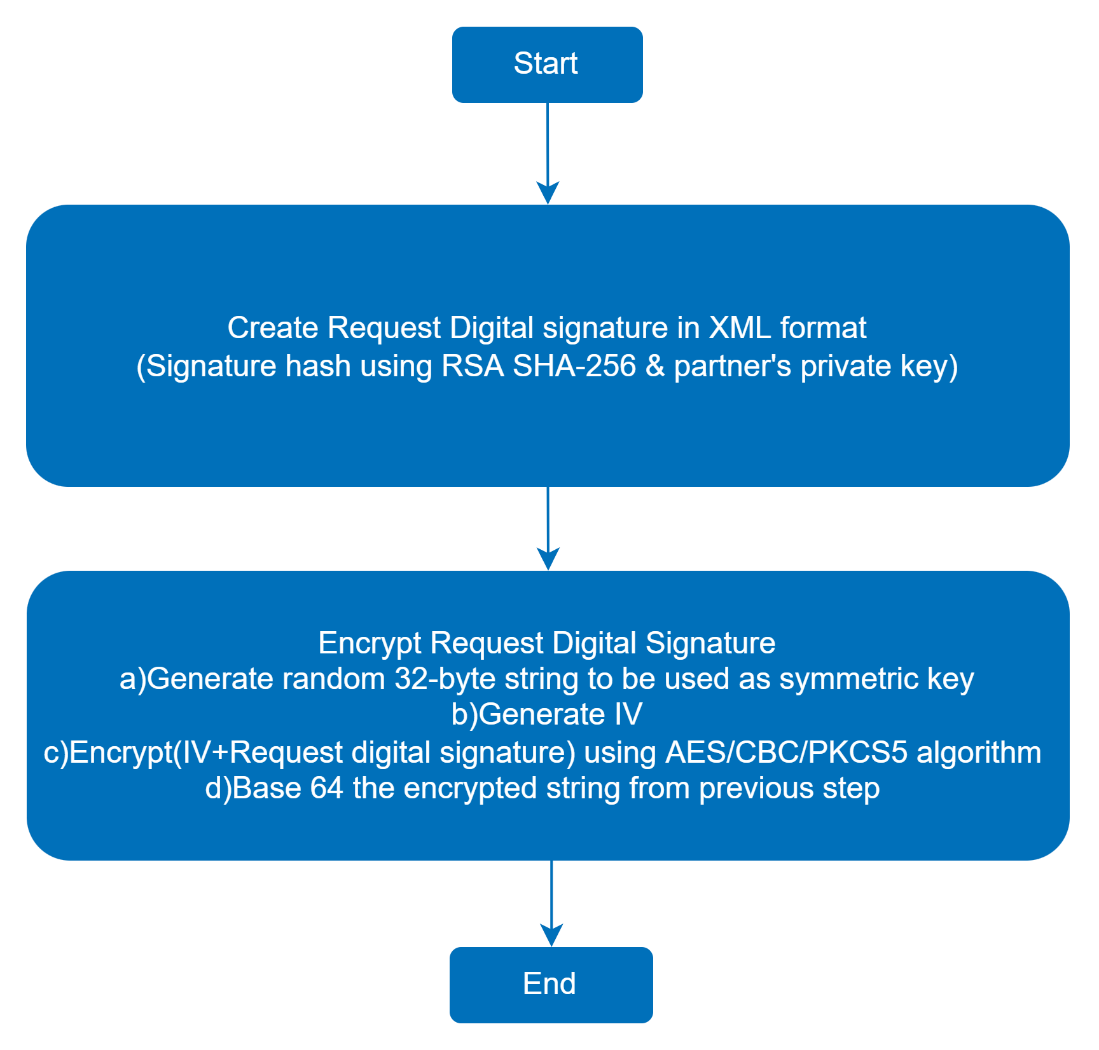
4. TransactionId: External partners need to set a transaction ID to uniquely identify every request, in order to retrieve it from an audit trail at a later date.

5. OAuthTokenValue: This value can be obtained by invoking HDFC Bank token generator service.

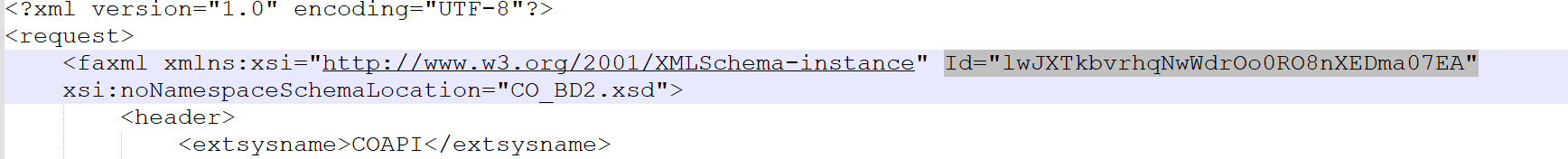
### 

### Detailed description of the Standard JSON Request Payload fields:

1. **RequestSignatureEncryptedValue**: This will contain the base64 encoded encrypted value of IV + digitally signed original XML request sample.



1. Steps for creating digital signature in XML Web Signature (XML) format:
2. To the request XML payload add **Id** attribute to the faml or faxml tags

Note: Id should be 32 byte alphanumeric String.

1. Do the XML Digital Signature the xml request payload with partners privatekey with RSA-SHA-256 algorithm and add certificate to the signature under X509Data tag

Below is the sample signature

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>

<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>

<Reference URI="#lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA">

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>

<DigestValue>Y4rjatbbUazQqA7Au2jySVuplYUowunlm5ulVrDahoo=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>hDp/hbC8eAlvM8KbnjWQBBf8pTLnsR7y+iq771uf4OKW35//jSup01vTP2Covfyhd99Wfc+StQyN&#xD; gTvCHNTDmAfq8SqHmuLstO9KIrTAjiJkF/leD4XTjsJzmz0WwTJPlKDqagj/EIx/zG2kyG9HrqnD&#xD; kocfWZGdcAcNwLVhTbNznI6F8Zxzcq6e3XYjlddqeUsA0stOkTxLJOcUdLZt5eRcf+FbmqzogH9A&#xD; H2anrpZ9zzyVVaSAPiezWrpo08O/tcIM1xcAWDvmnv0E8tsZTj/KqkX8o2s52BWsDE4fFkSkHIsN&#xD; R1JqiyRV/MFRpqBZHjIYI4K+ZVcQfC6qa5iJvQ==</SignatureValue>

<KeyInfo>

<X509Data>

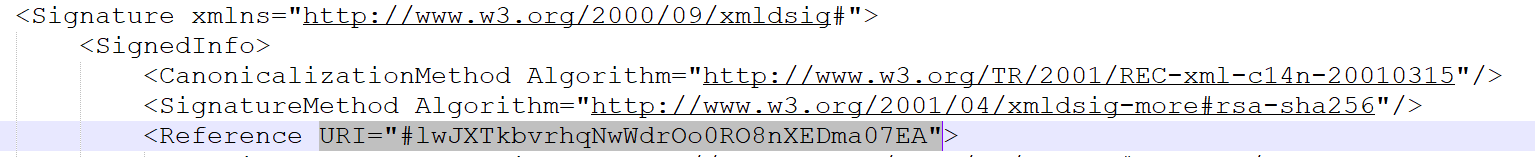
<X509SubjectName>CN=vinod</X509SubjectName>

<X509Certificate>MIIC5TCCAc2gAwIBAgIJAKTgujyYltyIMA0GCSqGSIb3DQEBDAUAMBAxDjAMBgNVBAMTBXZpbm9k&#xD; MB4XDTE4MDUwNzA5MzU0MVoXDTIzMDUwNjA5MzU0MVowEDEOMAwGA1UEAxMFdmlub2QwggEiMA0G&#xD; CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCPMNQ/jgdSkelCikA0c9ZUvnikKtZ/t7H2SoduwvAJ&#xD; 9NQfnP6xTqd3owHGnj2jvy7Um+MQv5dFR9OvmDQcxI8wENQQl/Svt+dKKxeDphYFlBlAcbMLeXcH&#xD; iwcVV2F0Kuie/F2Ui8P9enQ/wq2YS95XkxBrisDsTyNrqLob57eXV8122f3fuc1rPtXmrrr3OJzU&#xD; EwlZqGDRfmh3QlByCMvEPps/r2Zm6mdhyDooqOHXsVz9Z5H5HcrywZhJ9csWeTh/GtyujhClPaG0&#xD; Yg4pdy90X3fhEpoEJGHY6nclePo3r/1jDHlCH+YMmlA0VJYtONZhvLlFboEJZ4WLkBxSk0QZAgMB&#xD; AAGjQjBAMB0GA1UdDgQWBBRz+4a+K9Y4GCMCFc9+6CeisCFC4zAfBgNVHSMEGDAWgBRz+4a+K9Y4&#xD; GCMCFc9+6CeisCFC4zANBgkqhkiG9w0BAQwFAAOCAQEAM/tVb54iv+jEnE+mlZnZuzZ9rfbjSIri&#xD; OAi/mso2G2ydCvlTecRFTDwo8pjlwTr4a4Bm8dGPywemOX8aMM+8984KP/taeMKoGB+6miXmE+5Y&#xD; Mv5uy7LjZ8AzjuX1NSPX2MjOxNJJSe1uhsNph2j1OcRMyiNO/xI0zysyVJK8E6dtmdS8iddge2Qr&#xD; ev6Wb4JM9mW8tTwLwHgLxd32CAwLQngCcX0+RASWQ22V2reVjV0s1TwpEgGcUFgkTSIPeWtGqoUm&#xD; Y9rfR4oQ+DHLS27Hgh6gsT6EGLzHXVqSHJKXgZMavzIbO6jGNfxb6zYNPZ6FCnxXy7wpVgIhUxl+&#xD; MiqmjA==</X509Certificate>

</X509Data>

</KeyInfo>

</Signature>

Note: In Signature add HASH + ID (generate in previous step a) to the Reference URI tag

1. Add a <request> tag over XML Signed payload

Below is the final signed XML Payload

*<?xml version="1.0" encoding="UTF-8"?>*

*<request>*

*<faxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Id="lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA" xsi:noNamespaceSchemaLocation="CO\_BD2.xsd">*

*<header>*

*<extsysname>COAPI</extsysname>*

*<datpost>2017-11-15</datpost>*

*<batchnumext>4057341</batchnumext>*

*<idtxn>CO\_BD2</idtxn>*

*<codcurr>INR</codcurr>*

*<iduser>SL711045249778</iduser>*

*<idcust>33188831</idcust>*

*<groupid>BAJAJFIN</groupid>*

*<reqdatetime>2017-11-15T06:00:07</reqdatetime>*

*</header>*

*<summary>*

*<orgcountdr>1</orgcountdr>*

*<orgcountcr>3</orgcountcr>*

*<orgsumdr>12000.00</orgsumdr>*

*<orgsumcr>12000.00</orgsumcr>*

*</summary>*

*<debit>*

*<stanext>1</stanext>*

*<accountno>11362560000112</accountno>*

*<orgamount>12000.00</orgamount>*

*<txndesc>Salary Oct 2017</txndesc>*

*<referenceno>711045249778</referenceno>*

*</debit>*

*</faxml>*

*<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">*

*<SignedInfo>*

*<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>*

*<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>*

*<Reference URI="#lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA">*

*<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>*

*<DigestValue>Y4rjatbbUazQqA7Au2jySVuplYUowunlm5ulVrDahoo=</DigestValue>*

*</Reference>*

*</SignedInfo>*

*<SignatureValue>hDp/hbC8eAlvM8KbnjWQBBf8pTLnsR7y+iq771uf4OKW35//jSup01vTP2Covfyhd99Wfc+StQyN&#xD; gTvCHNTDmAfq8SqHmuLstO9KIrTAjiJkF/leD4XTjsJzmz0WwTJPlKDqagj/EIx/zG2kyG9HrqnD&#xD; kocfWZGdcAcNwLVhTbNznI6F8Zxzcq6e3XYjlddqeUsA0stOkTxLJOcUdLZt5eRcf+FbmqzogH9A&#xD; H2anrpZ9zzyVVaSAPiezWrpo08O/tcIM1xcAWDvmnv0E8tsZTj/KqkX8o2s52BWsDE4fFkSkHIsN&#xD; R1JqiyRV/MFRpqBZHjIYI4K+ZVcQfC6qa5iJvQ==</SignatureValue>*

*<KeyInfo>*

*<X509Data>*

*<X509SubjectName>CN=vinod</X509SubjectName>*

*<X509Certificate>MIIC5TCCAc2gAwIBAgIJAKTgujyYltyIMA0GCSqGSIb3DQEBDAUAMBAxDjAMBgNVBAMTBXZpbm9k&#xD; MB4XDTE4MDUwNzA5MzU0MVoXDTIzMDUwNjA5MzU0MVowEDEOMAwGA1UEAxMFdmlub2QwggEiMA0G&#xD; CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCPMNQ/jgdSkelCikA0c9ZUvnikKtZ/t7H2SoduwvAJ&#xD; 9NQfnP6xTqd3owHGnj2jvy7Um+MQv5dFR9OvmDQcxI8wENQQl/Svt+dKKxeDphYFlBlAcbMLeXcH&#xD; iwcVV2F0Kuie/F2Ui8P9enQ/wq2YS95XkxBrisDsTyNrqLob57eXV8122f3fuc1rPtXmrrr3OJzU&#xD; EwlZqGDRfmh3QlByCMvEPps/r2Zm6mdhyDooqOHXsVz9Z5H5HcrywZhJ9csWeTh/GtyujhClPaG0&#xD; Yg4pdy90X3fhEpoEJGHY6nclePo3r/1jDHlCH+YMmlA0VJYtONZhvLlFboEJZ4WLkBxSk0QZAgMB&#xD; AAGjQjBAMB0GA1UdDgQWBBRz+4a+K9Y4GCMCFc9+6CeisCFC4zAfBgNVHSMEGDAWgBRz+4a+K9Y4&#xD; GCMCFc9+6CeisCFC4zANBgkqhkiG9w0BAQwFAAOCAQEAM/tVb54iv+jEnE+mlZnZuzZ9rfbjSIri&#xD; OAi/mso2G2ydCvlTecRFTDwo8pjlwTr4a4Bm8dGPywemOX8aMM+8984KP/taeMKoGB+6miXmE+5Y&#xD; Mv5uy7LjZ8AzjuX1NSPX2MjOxNJJSe1uhsNph2j1OcRMyiNO/xI0zysyVJK8E6dtmdS8iddge2Qr&#xD; ev6Wb4JM9mW8tTwLwHgLxd32CAwLQngCcX0+RASWQ22V2reVjV0s1TwpEgGcUFgkTSIPeWtGqoUm&#xD; Y9rfR4oQ+DHLS27Hgh6gsT6EGLzHXVqSHJKXgZMavzIbO6jGNfxb6zYNPZ6FCnxXy7wpVgIhUxl+&#xD; MiqmjA==</X509Certificate>*

*</X509Data>*

*</KeyInfo>*

*</Signature>*

*</request>*

1. Steps for encrypting the digital signature value obtained from 1 A):
2. Generate a random string of 32 bytes (256 bits) using below character range

(A-Z&a-z&0-9) which will be utilized as a Symmetric key for encryption of original request payload.

Character Range:

((character <= Z && character >= A) || (character <= z && character >= a)   
|| (character <= NINE && character >= ZERO))

The 32 bytes random string will look like this:

OfpBWSiDeP6kIjbyYFDGu3TnBqxTEpLM

Note: Special characters are not allowed

This will later be used in step 2 for obtaining ‘SymmetricKeyEncryptedValue’.

1. Generate random IV of 16 character length and only alphanumeric are allowed
2. Encrypt IV + request digital signature using the 32 bytes random generated symmetric key

*Encrypt (<<IV Value>>+<<request XML digital signature>>)*

Algorithm: AES/CBC/PKCS5Padding.

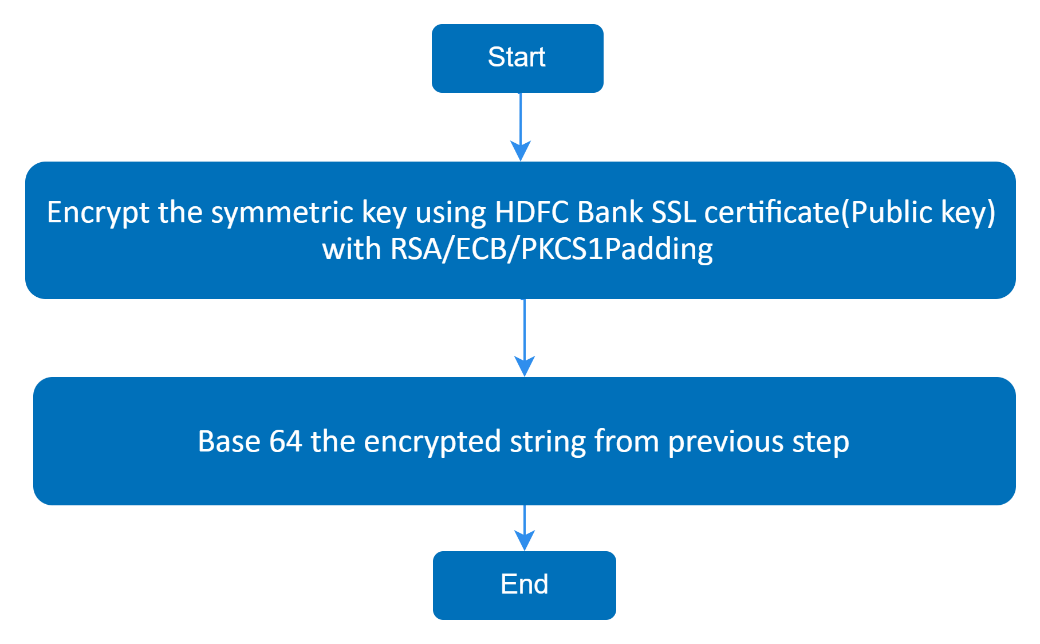
\*\*Please note IV value being used for AES encryption, should be sent along with the request payload at the start. IV value should be of 16 characters length and only alphanumeric are allowed.

1. Base64 encode the Encrypted value from point c)

This will look like:



1. The encrypted value above is to be sent within the tag for RequestSignatureEncryptedValue
2. **SymmetricKeyEncryptedValue**: This will contain the base64 encoded encrypted value of a 32 byte symmetric key used for encrypting in Step1.



Steps for encryption:

1. External partner needs to use the TLS/SSL certificate of the HDFC Bank’s Public API Gateway shared as a public key to perform the asymmetric key encryption of the symmetric key generated in step 1 B a).

Algorithm: RSA/ECB/PKCS1Padding

1. Base 64 encode the above result.

Result will look like this:

gTwOshZ8xb04zfjF0r17hyPwnMRM71Hp0Ej4e2U9/fXGHn4zw70eNzUTTUMBIYg9Ut/3Un0ApQvI+0YphcVl3rdmp8IekjYV7EGT3nNIBVltCZJ4ydThMU6h/6afCdg6jklzasHXaRmXXR2k03ufs4zuvKLjncRKnXjhlJyjxzRTuNVG0nRKS9iolQRTw5BkCRjRleZaOCSyLzc2d4IDLwBCqVfstl2W2fRZvmiDoCHDswRcZK0AJFZsg71lh8TXbFhp9VUGy1rOM3PSbHRXQdrtZbBt5c7j+rzmxe3Y/+8mgb1TIQPN6+nzMeESYjnMWQFaTN5R+INs0uXDrl8zgg==

1. The encrypted value to be send within the tag for SymmetricKeyEncryptedValue.

1. **Scope**: This contains the scope value (which was set on the HDFC Bank’s API portal while registering the consumer application). It will get validated against the corresponding apikey (which can be obtained from the HDFC Bank’s API portal after successfully registering the consumer application) sent in the request header.
2. **TransactionId**: Set a unique transaction ID every-time for identifying the request, in order to retrieve it from an audit trail. The value of this field should be alphanumeric without any special characters.
3. **OAuthTokenValue**: This value can be obtained by invoking HDFC token generator service.

OAUTH service Endpoint:-

<https://api-uat.hdfcbank.com/auth/oauth/v2/token>

The parameters for token generator service are as follows-

1. Authorization Headers-
2. API Key (Client ID): This field contains your client ID obtained from the HDFC Bank API portal.

This will look like-

l7392addb99ed44e93937d914c3dc4d4c6

This value needs to be passed as ‘Username’ in Authorization Headers.

1. Shared Secret (Client Secret): This field contains your client Secret obtained from the HDFC Bank API portal.

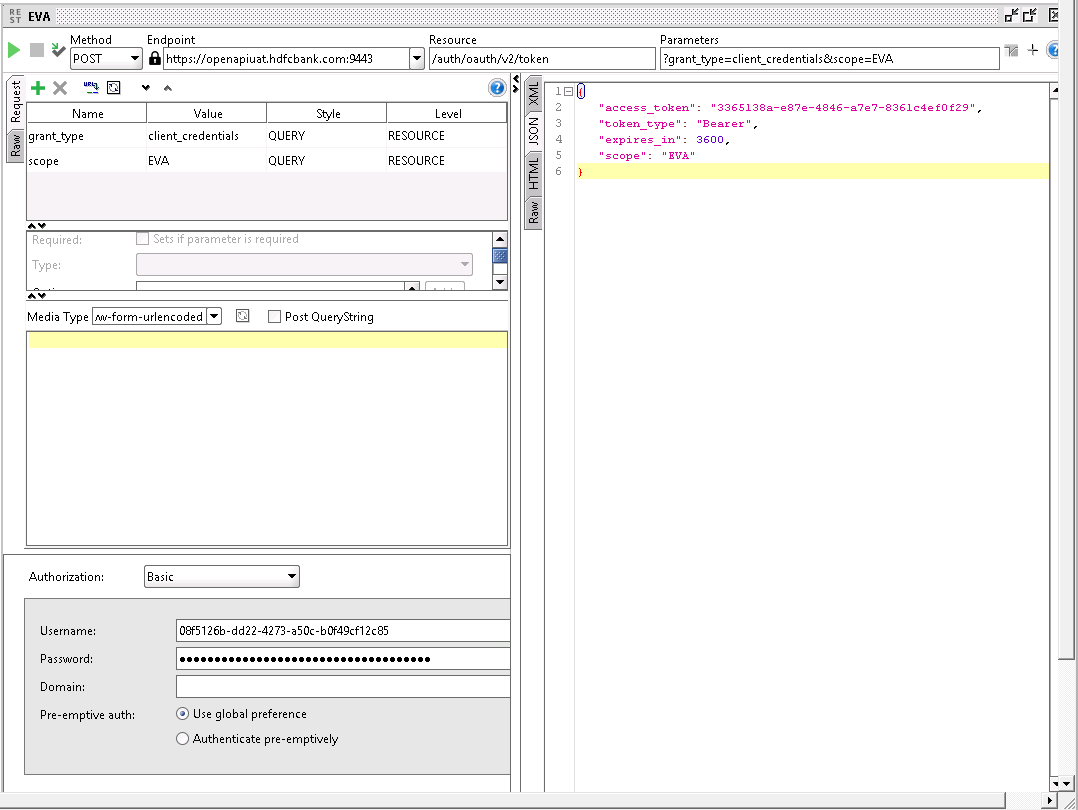
This will look like-

dc728bdfbbd64c04ba3675082a08ce8f

This value needs to be passed as ‘Password’ in Authorization Headers.

1. Query Parameters-
2. ‘grant\_type’- Please set the ‘grant\_type’ as “client\_credentials”.
3. ‘Scope’ - Please set the ‘Scope’ as the **scope** value which was set on the HDFC Bank API portal while registering the consumer application.

\*\*Please refer to the Sample screenshot for OAuth Token request from a SOAP-UI test tool below:-



**In the end the whole JSON Request will look like this**:

{

"RequestSignatureEncryptedValue": "",

"SymmetricKeyEncryptedValue": "gTwOshZ8xb04zfjF0r17hyPwnMRM71Hp0Ej4e2U9/fXGHn4zw70eNzUTTUMBIYg9Ut/3Un0ApQvI+0YphcVl3rdmp8IekjYV7EGT3nNIBVltCZJ4ydThMU6h/6afCdg6jklzasHXaRmXXR2k03ufs4zuvKLjncRKnXjhlJyjxzRTuNVG0nRKS9iolQRTw5BkCRjRleZaOCSyLzc2d4IDLwBCqVfstl2W2fRZvmiDoCHDswRcZK0AJFZsg71lh8TXbFhp9VUGy1rOM3PSbHRXQdrtZbBt5c7j+rzmxe3Y/+8mgb1TIQPN6+nzMeESYjnMWQFaTN5R+INs0uXDrl8zgg==",

"Scope": "ExampleOrg",

"TransactionId": "REF1234",

"OAuthTokenValue": "9aaed862-477f-4274-8436-ba9bbee4fc47"

}

## Response:

### Standard JSON Response Payload Structure:

{

"ResponseSignatureEncryptedValue": "",

"GWSymmetricKeyEncryptedValue": "",

"Scope": "",

"TransactionId": "",

"Status": ""

}

### Receiving response payload will look like:

{

"ResponseSignatureEncryptedValue": "",

"GWSymmetricKeyEncryptedValue": "LA0DVfqMBLb0k8ujpoHg97r/ff5tfnKo+TBn5EaJxqwGeBZENTvXSDz8b9aX4hm7hNAeqH9+Lr9/T62mK1AUC3tka1nKpzF2L6Ciju4PBak1n2G0IfL3MKpo/dkyK3txbypQ5eREcDkLqjJbDjot4Hb17UnisPOoDjMROb67RYxkOVgHn0PtsG9T+X8nqdU8mug+1OHOY+HH6+5TGUYDRvOwC5YiE6N6dfRMeupwZo4jPBZZNE5uLzk5bWcrrmu/8is2Q1F8AJvq+rfDBK2jwbYrm1HXbQZL1STld078vGOxc4rMZJ2cbfYlFOiBoSUdEGq1nKuLG1cB5IE7hZ5SXg==",

"Scope": "ExampleOrg",

"TransactionId": "REF1234",

"Status": "SUCCESS"

}

The individual fields of the Standard JSON response Payload are briefly described below:

7. ResponseSignatureEncryptedValue: This will contain the base64 encoded encrypted value of IV + digitally signed original XML response.

8. GWSymmetricKeyEncryptedValue: This will contain base64 encoded encrypted value of the 32 byte Symmetric Key which will be used for decrypting in 7.

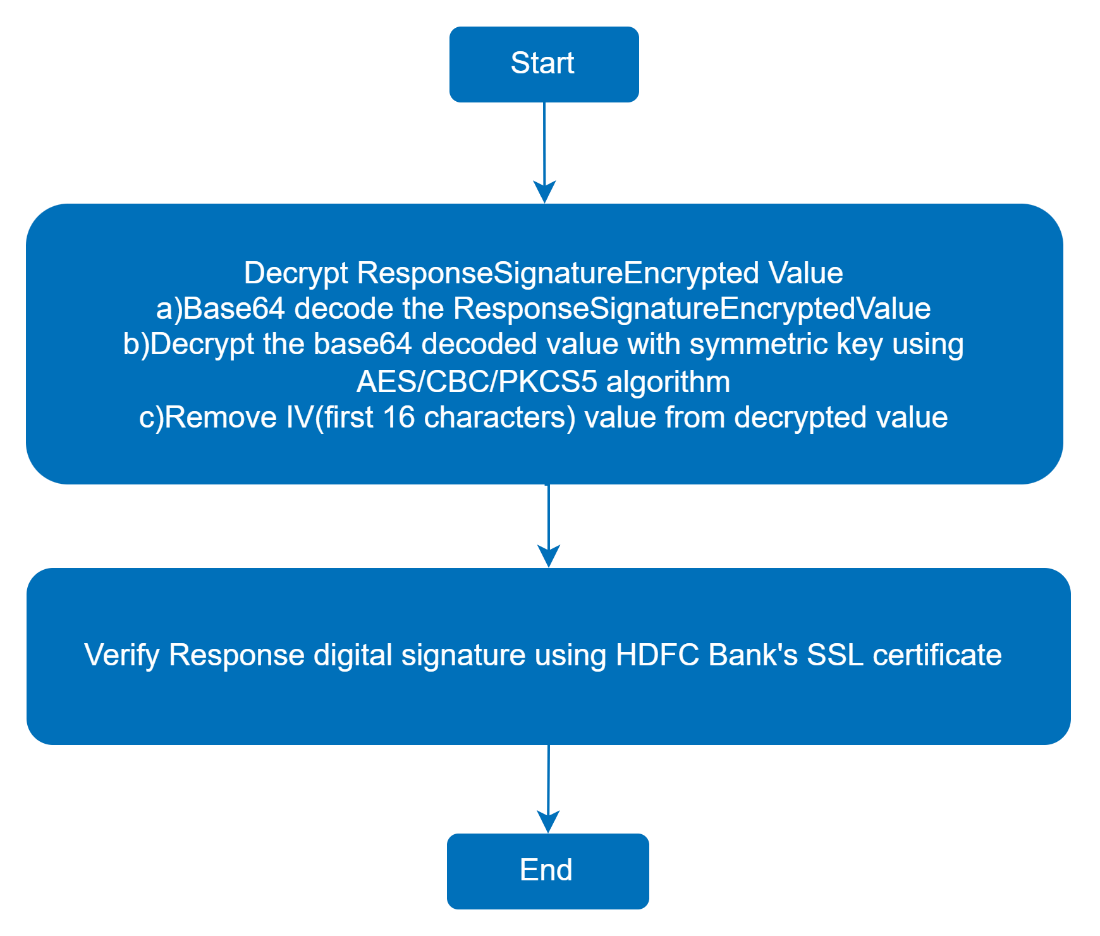
9. Scope: This field will contain the Scope value which was received in the request at HDFC Bank’s Public API Gateway.

10. TransactionId: This field will contain the transaction ID which was received in request at HDFC Bank’s Public API Gateway.

11. Status: This field will return the status of response with fail or success message, or specific error codes.

### Detailed description of the Standard JSON Response Payload fields:

1. **ResponseSignatureEncryptedValue**: This will contain the base64 encoded encrypted value of IV + digitally signed original XML response.



1. Steps for decryption:
   1. Key to be used is decrypted key retrieved from the second field in response named GWSymmetricKeyEncryptedValue. Follow steps for decryption as listed in step 8

After decryption, we get the 32 bytes key, which looks like this:

OfpBWSiDeP6kIjbyYFDGu3TnBqxTEpLM

This value is to be used as the Symmetric key for decryption of Response as given in step 7 c).

* 1. Base64 decode the ResponseSignatureEncryptedValue
  2. Decrypt the decoded value with symmetric key using Algorithm: AES/CBC/PKCS5 Padding Result will be (IV + XML digital signature)

<?xml version="1.0" encoding="UTF-8"?>

1234567890123456<response>

<faxml xsi:noNamespaceSchemaLocation="CO\_BD2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Id="lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA">

<header>

<extsysname>COAPI</extsysname>

<datpost>2021-11-10</datpost>

<batchnumext>000050</batchnumext>

<idtxn>CO\_BD2</idtxn>

<codcurr>INR</codcurr>

<iduser>APIUser@APITEST</iduser>

<idcust>10054238</idcust>

<groupid>FCAT\_APITEST</groupid>

<reqdatetime>2021-11-10T17:52:49</reqdatetime>

<batchnum>2021314069153743</batchnum>

<codstatus>0</codstatus>

<txtstatus>ACCEPTED</txtstatus>

</header>

<summary>

<orgcountdr>1</orgcountdr>

<orgcountcr>1</orgcountcr>

<orgsumdr>1</orgsumdr>

<orgsumcr>1</orgsumcr>

</summary>

<debit>

<stanext>1</stanext>

<accountno>00010110000182</accountno>

<orgamount>1</orgamount>

<txndesc>API A2A PAYMENT\_DEBIT</txndesc>

<referenceno>API3456789</referenceno>

</debit>

<creditlist>

<credit>

<stanext>2</stanext>

<accountno>26090350000028</accountno>

<amount>1</amount>

<txndesc>API A2A PAYMENT\_CREDIT</txndesc>

<referenceno>GPI3456789</referenceno>

<beneid/>

<BeneName>HDFC API</BeneName>

<BeneAddress\_1/>

<BeneAddress\_2/>

<BeneAddress\_3/>

<BeneAddress\_4/>

<emailid>vrushali.bhagwat@hdfcbank.com</emailid>

</credit>

</creditlist>

</faxml>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">

<SignedInfo>

<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>

<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>

<Reference URI="#lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA">

<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>

<DigestValue>Y4rjatbbUazQqA7Au2jySVuplYUowunlm5ulVrDahoo=</DigestValue>

</Reference>

</SignedInfo>

<SignatureValue>hDp/hbC8eAlvM8KbnjWQBBf8pTLnsR7y+iq771uf4OKW35//jSup01vTP2Covfyhd99Wfc+StQyN&#xD; gTvCHNTDmAfq8SqHmuLstO9KIrTAjiJkF/leD4XTjsJzmz0WwTJPlKDqagj/EIx/zG2kyG9HrqnD&#xD; kocfWZGdcAcNwLVhTbNznI6F8Zxzcq6e3XYjlddqeUsA0stOkTxLJOcUdLZt5eRcf+FbmqzogH9A&#xD; H2anrpZ9zzyVVaSAPiezWrpo08O/tcIM1xcAWDvmnv0E8tsZTj/KqkX8o2s52BWsDE4fFkSkHIsN&#xD; R1JqiyRV/MFRpqBZHjIYI4K+ZVcQfC6qa5iJvQ==</SignatureValue>

<KeyInfo>

<X509Data>

<X509SubjectName>CN=vinod</X509SubjectName>

<X509Certificate>MIIC5TCCAc2gAwIBAgIJAKTgujyYltyIMA0GCSqGSIb3DQEBDAUAMBAxDjAMBgNVBAMTBXZpbm9k&#xD; MB4XDTE4MDUwNzA5MzU0MVoXDTIzMDUwNjA5MzU0MVowEDEOMAwGA1UEAxMFdmlub2QwggEiMA0G&#xD; CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCPMNQ/jgdSkelCikA0c9ZUvnikKtZ/t7H2SoduwvAJ&#xD; 9NQfnP6xTqd3owHGnj2jvy7Um+MQv5dFR9OvmDQcxI8wENQQl/Svt+dKKxeDphYFlBlAcbMLeXcH&#xD; iwcVV2F0Kuie/F2Ui8P9enQ/wq2YS95XkxBrisDsTyNrqLob57eXV8122f3fuc1rPtXmrrr3OJzU&#xD; EwlZqGDRfmh3QlByCMvEPps/r2Zm6mdhyDooqOHXsVz9Z5H5HcrywZhJ9csWeTh/GtyujhClPaG0&#xD; Yg4pdy90X3fhEpoEJGHY6nclePo3r/1jDHlCH+YMmlA0VJYtONZhvLlFboEJZ4WLkBxSk0QZAgMB&#xD; AAGjQjBAMB0GA1UdDgQWBBRz+4a+K9Y4GCMCFc9+6CeisCFC4zAfBgNVHSMEGDAWgBRz+4a+K9Y4&#xD; GCMCFc9+6CeisCFC4zANBgkqhkiG9w0BAQwFAAOCAQEAM/tVb54iv+jEnE+mlZnZuzZ9rfbjSIri&#xD; OAi/mso2G2ydCvlTecRFTDwo8pjlwTr4a4Bm8dGPywemOX8aMM+8984KP/taeMKoGB+6miXmE+5Y&#xD; Mv5uy7LjZ8AzjuX1NSPX2MjOxNJJSe1uhsNph2j1OcRMyiNO/xI0zysyVJK8E6dtmdS8iddge2Qr&#xD; ev6Wb4JM9mW8tTwLwHgLxd32CAwLQngCcX0+RASWQ22V2reVjV0s1TwpEgGcUFgkTSIPeWtGqoUm&#xD; Y9rfR4oQ+DHLS27Hgh6gsT6EGLzHXVqSHJKXgZMavzIbO6jGNfxb6zYNPZ6FCnxXy7wpVgIhUxl+&#xD; MiqmjA==</X509Certificate>

</X509Data>

</KeyInfo>

</Signature>

</response>

* 1. Remove the first 16 characters (i.e; IV) to get the XML digital Signature.

<response>

<faxml xsi:noNamespaceSchemaLocation="CO\_BD2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Id="lwJXTkbvrhqNwWdrOo0RO8nXEDma07EA">

..........................................................................................

</response>

1. Steps to verify the response digital signature-
2. First remove <response> tag from the XML digital signature.
3. Then validate the Response XML digital signature value obtained as a result from 7 A (d) using HDFC Banks’s public key.
4. After signature gets validated remove the <Signature> from the payload and then remove Id attribute from the <faxml> or <faml> tag.
5. Below is the final plaintext response.

<faxml xsi:noNamespaceSchemaLocation="CO\_BD2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<header>

<extsysname>COAPI</extsysname>

<datpost>2021-11-10</datpost>

<batchnumext>000050</batchnumext>

<idtxn>CO\_BD2</idtxn>

<codcurr>INR</codcurr>

<iduser>APIUser@APITEST</iduser>

<idcust>10054238</idcust>

<groupid>FCAT\_APITEST</groupid>

<reqdatetime>2021-11-10T17:52:49</reqdatetime>

<batchnum>2021314069153743</batchnum>

<codstatus>0</codstatus>

<txtstatus>ACCEPTED</txtstatus>

</header>

<summary>

<orgcountdr>1</orgcountdr>

<orgcountcr>1</orgcountcr>

<orgsumdr>1</orgsumdr>

<orgsumcr>1</orgsumcr>

</summary>

<debit>

<stanext>1</stanext>

<accountno>00010110000182</accountno>

<orgamount>1</orgamount>

<txndesc>API A2A PAYMENT\_DEBIT</txndesc>

<referenceno>API3456789</referenceno>

</debit>

<creditlist>

<credit>

<stanext>2</stanext>

<accountno>26090350000028</accountno>

<amount>1</amount>

<txndesc>API A2A PAYMENT\_CREDIT</txndesc>

<referenceno>GPI3456789</referenceno>

<beneid/>

<BeneName>HDFC API</BeneName>

<BeneAddress\_1/>

<BeneAddress\_2/>

<BeneAddress\_3/>

<BeneAddress\_4/>

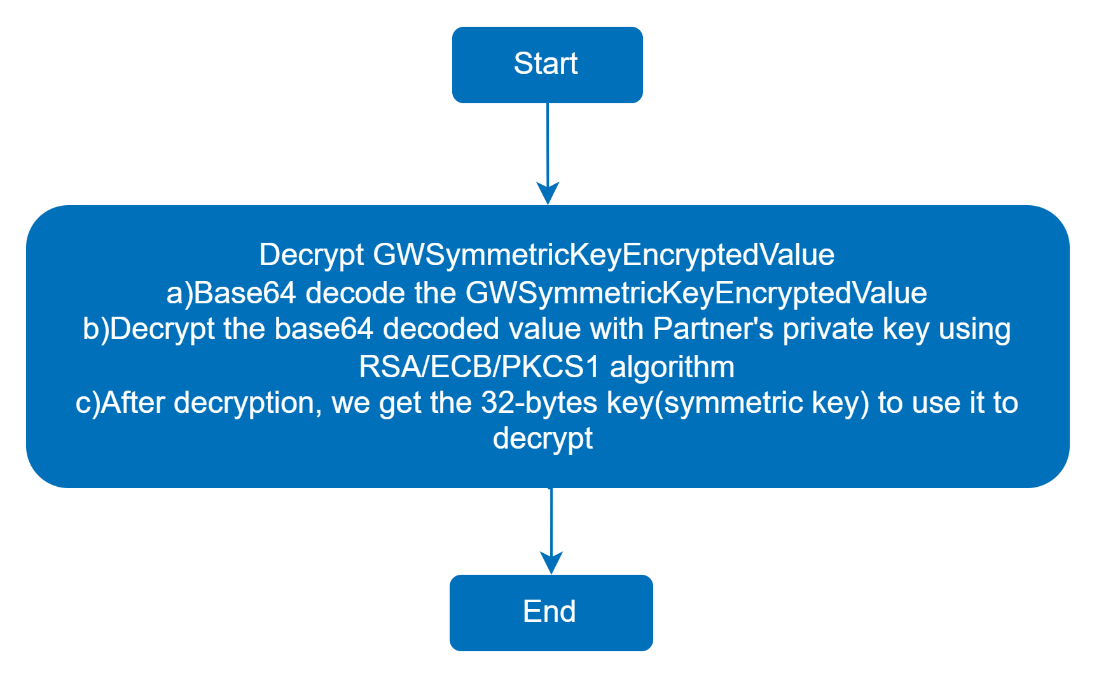
<emailid>vrushali.bhagwat@hdfcbank.com</emailid>

</credit>

</creditlist>

</faxml>

1. **GWSymmetricKeyEncryptedValue**: This will contain base64 encoded encrypted value of the 32 byte Symmetric Key which will be used for decryption of ResponseSignatureEncryptedValue.



1. Steps for decryption:
2. Base64 decode the Key GWSymmetricKeyEncryptedValue:
3. External partner needs to use the private key of their SSL certificate for decryption.
4. Decrypt the decode value with Partner’s Private key using Algorithm: RSA/ECB/PKCS1Padding

After decryption, we get the 32 bytes key, which looks like this:

OfpBWSiDeP6kIjbyYFDGu3TnBqxTEpLM

1. The above value needs to be used as key for decryption of the tag ResponseSignatureEncryptedValue in Step7

1. **Scope**: This field will contain the Scope value which was received in the request at HDFC Bank.
2. **TransactionId**: This field will contain the transaction ID which was received in request at HDFC Bank.
3. **Status**: This field will return the Status of response with fail or success message, or specific error codes.

NOTE:

1. Encrypted values and digital signature values will never repeat, for the same input text.
2. Only the Base64 Encoded value will remain same for the similar input.
3. Perform Xpath or JSON path extraction of the values for conversion.

NOTE:

1. Encrypted values and digital signature values will never repeat, for the same input text.
2. Only the Base64 Encoded value will remain same for the similar input.
3. Perform Xpath or JSON path extraction of the values for conversion.

# Certificate Prerequisites

1. All certificates used to access HDFC Bank's APIs need to be signed by a valid certifying authority. This is applicable even for UAT.

# Error/Status Codes Details

## HDFC Bank’s Public API Gateway Error Codes:-

|  |  |  |
| --- | --- | --- |
| **Error Codes** | **Error Message** | **Http Status Code** |
| TH99401 | Attempted login from unauthorized IP | 401 Unauthorized |
| TH99405 | Request Method Not Allowed For API Access | 405 Method Not Allowed |
| TH99429 | DDOS Attack Detected from Requestor | 429 Too Many Requests |
| TH99410 | Cross Site Forgery Detected in API Request Message Payload | 410 Gone |
| TH99400 | Code Injection Detected in API Request Message Payload | 400 Bad Request |
| TH99421 | Document Structure Threat in API Request Message Payload | 421 Misdirected Request |
| TH99422 | SQL Injection Detected in API Request Message Payload | 422 Unprocessable Entity (WebDAV) |
| TH99411 | Invalid Content Type in API Request Message Payload | 411 Length Required |
| TH99503 | API Back-end Service Not Available or Timed-Out | 503 Service Unavailable |
| TH99429 | API Access Quota Exceeded | 429 Too Many Requests |
| TH99423 | XML Entity Expansion Attack Detected from Requester | 423 Locked (WebDAV) |
| TH99500 | Backend Service Provided Unexpected Response | 500 Internal Server Error |
| TH99412 | Oauth Token Verification Failed | 412 Precondition Failed |
| TH99412 | ID Token Verification Failed | 412 Precondition Failed |
| TH99401 | Invalid Client Certificate | 401 Unauthorized |
| TH99401 | Scope Validation Failed | 401 Unauthorized |
| TH99401 | Invalid API Key | 401 Unauthorized |
| TH99401 | Signature verification Failed | 401 Unauthorized |
| TH99400 | Decryption Failed | 400 Bad Request |
| TH99400 | Bad Request | 400 Bad Request |

**In case the transaction fails at the HDFC Bank’s Public API Gateway, the error structure will be as follows:**

HTTP status code is returned with error payload in JSON format as below-

{

"ResponseSignatureEncryptedValue": "",

"GWSymmetricKeyEncryptedValue": "",

"Scope": "ExampleOrg",

"TransactionId": "REF1234",

"Status": "TH99401: Invalid Client Certificate ",

"Id-token-jwt": ""

}

# Appendix

## Other Prerequisites required from External partner:

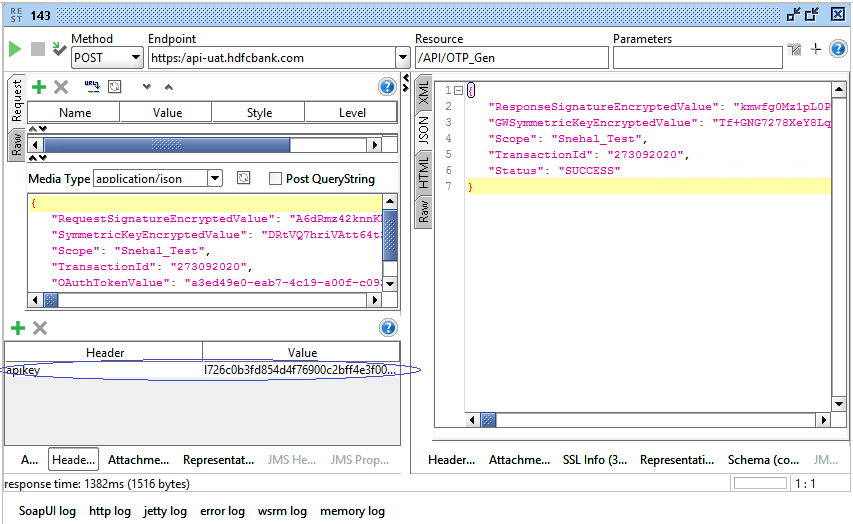
* SSL certificate (along with Root and Intermediate CA certificates) of the external partner/consumers to be shared for two-way mutual SSL authentication. SSL Leaf certificate to be uploaded on API Portal during onboarding process.
* List of source IPs of the external partner/consumers to be uploaded to API Portal during onboarding process.
* HTTPS requests to HDFC Bank should TLS 1.2.
* Sign-up at API Portal https://developer-uat.hdfcbank.com:8443/, by Representative of external partner. Following acceptance of Sign-up by HDFC Bank, external partner needs to click on link Activation mail, and complete a form on the Bank’s API Portal.

## API Key Details:-

The Bank's API Portal will display a value known as an apikey after access to the API has been granted. The API Key should be sent, by the external partner, as part of the request header of the API request made to HDFC Bank.

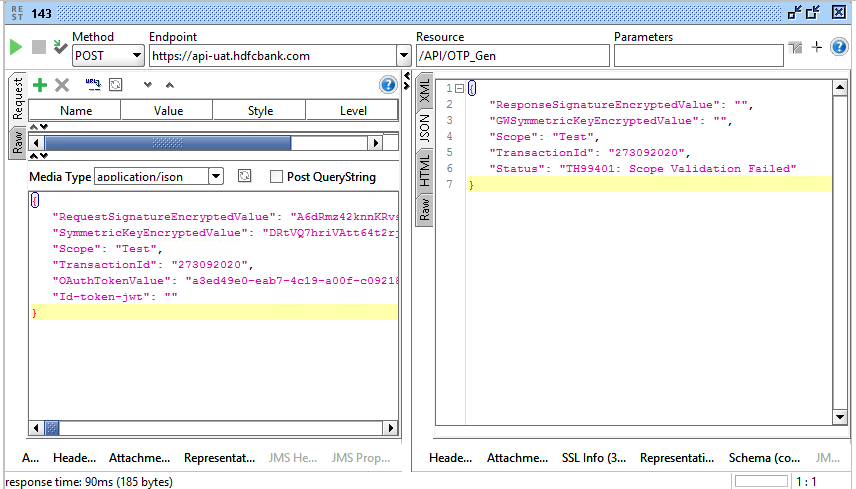
The Bank's Public API will be verifying the apikey value coming from the external partner.

A screen shot of the Bank’s main web service testing tool, SoapUI, shows the apikey in the header encircled in blue. (Sample screenshot below)



The Bank's API also expects the scope value in the “Scope” field of the request packet which should correspond to registered API Key (Client ID). This value will be verified by the Public API Platform for requests from the external entity.

In case invalid values are provided for either for Scope, an error message such as “Scope Validation Failed” should be expected with response sample as below-



## Sample Code Snippets for Client-Side API integration

### Code Snippet for Client-Side Encryption for the request payload field RequestSignatureEncryptedValue:

/\*\*

\* declaring instance variables here

\*

\*/

**private** **static** **final** **int** ***VAR1*** = 'a';

**private** **static** **final** **int** ***VAR2*** = 'z';

**private** **static** **final** **int** ***VAR3*** = 'A';

**private** **static** **final** **int** ***VAR4*** = 'Z';

**private** **static** **final** **int** ***ZERO*** = '0';

**private** **static** **final** **int** ***NINE*** = '9';

/\*\*

\* Declaring IV(16 Bytes)

\*/

**private** **static** **final** **byte**[] ***IVECTOR*** = "1234567890123456".getBytes();

/\*\*

\* generate 32 byte alphanumeric Symmetric key

\*

\*/

**public** String generateAlphaNumericKey(**final** **int** keySize) {

**final** Random random = **new** Random();

**final** StringBuilder key = **new** StringBuilder();

**while**(key.length() < keySize) {

**final** **int** character = random.nextInt(128);

//is within range

**if**((character <= ***VAR2*** && character >= ***VAR1***) || (character <= ***VAR4*** && character >= ***VAR3***) || (character <= ***NINE*** && character >= ***ZERO***)) {

key.append((**char**)character);

}

}

**return** key.toString();

}

/\*\*

\* Snippet for encrypting the payload using symmetric key

\* Algorithm: AES/CBC/PKCS5Padding

\* Here data is signed JWS with partner’s private key

\*/

**public** **byte**[] encrypt(**final** String data, **final** **byte**[] key) {

**byte**[] encryptedValue = **null**;

**try** {

**final** SecretKeySpec secretKeySpec = **new** SecretKeySpec(key, "AES");

**final** Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");

cipher.init(Cipher.***ENCRYPT\_MODE***, secretKeySpec, **new** IvParameterSpec(***IVECTOR***));

**final** **byte**[] dataArr = data.getBytes();

**final** **byte**[] ivAndData = **new** **byte**[***IVECTOR***.length + dataArr.length];

System.*arraycopy*(***IVECTOR***, 0, ivAndData, 0, ***IVECTOR***.length);

System.*arraycopy*(dataArr, 0, ivAndData, ***IVECTOR***.length, dataArr.length);

encryptedValue = cipher.doFinal(ivAndData);

} **catch** (NoSuchAlgorithmException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 1", exp);

} **catch** (NoSuchPaddingException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 2", exp);

} **catch** (IllegalBlockSizeException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 3", exp);

} **catch** (BadPaddingException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 4", exp);

} **catch** (InvalidKeyException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 5", exp);

} **catch** (InvalidAlgorithmParameterException exp) {

***LOGGER***.log(Level.***INFO***, "Exception occur 6", exp);

}

**return** encryptedValue;

}

/\*\*

\* Base 64 encoding of byte array

\* **@param** byte array to be encoded

\* **@return** encoded byte array

\*/

**public** **byte**[] encode(**final** **byte**[] value) {

**return** Base64.*getEncoder*().encode(value);

}

/\*\*

\* Main Function of RequestSignatureEncryptedValue

\*

\*/

**public** **static** **void** main(String[] args){

final String jwsToken = “***your signature***”;

final String symmetricKeyValue =generateAlphaNumericKey(32);

final byte[] aesEncryptionData =encrypt(jwsToken, symmetricKeyValue.getBytes());

final String requestSignatureEncryptedValue = new String(encode(aesEncryptionData));

}

### Code Snippet for Client-Side Encryption for the request payload field SymmetricEncryptedValue

/\*\*

\* Snippet for fetching public Key

\*

\*/

**public** **static** PublicKey getPublicKey(**final** String base64PublicKey) {

PublicKey publicKey = **null**;

**try** {

**final** X509EncodedKeySpec keySpec = **new** X509EncodedKeySpec(

Base64.*getDecoder*().decode(base64PublicKey.getBytes()));

**final** KeyFactory keyFactory = KeyFactory.*getInstance*("RSA");

publicKey = keyFactory.generatePublic(keySpec);

} **catch** (NoSuchAlgorithmException e) {

***LOGGER***.log(Level.***INFO***, "Exception occur", e);

} **catch** (InvalidKeySpecException e) {

***LOGGER***.log(Level.***INFO***, "Exception occur", e);

}

**return** publicKey;

}

/\*\*

\* Snippet for encrypting the 32 bytes symmetric key using HDFC Bank’s Public Key

\*

\*/

**public** **static** **byte**[] encrypt(**final** String data, **final** String publicKey) **throws** BadPaddingException,

IllegalBlockSizeException, InvalidKeyException, NoSuchPaddingException, NoSuchAlgorithmException {

**final** Cipher cipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.***ENCRYPT\_MODE***, *getPublicKey*(publicKey));

**return** cipher.doFinal(data.getBytes());

}

/\*\*

\* Main Function of SymmetricEncryptedValue

\*/

**public** **static** **void** main(String[] args) {

**final** String publicKey = “***HDFC Bank publickey***”;

**final** String symmetricKeyValue = “***32 bytes symmetrickey***”;

**final** **byte**[] symmetricKeyEncryptedValue =encrypt(symmetricKeyValue, publicKey);

**final** String symmetricKeyBase64EncodedValue = Base64.*getEncoder*()

.encodeToString(symmetricKeyEncryptedValue);

}

## Glossary:

Definition

The following are definitions of terms, abbreviations and acronyms used in this document.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| REST | (REpresentational State Transfer) is an architectural style for developing web services |
| JSON | JavaScript Object Notation, a lightweight data-interchange format |
| Base64 | Base64 is an encoding and decoding technique used to convert binary data to an American Standard for Information Interchange (ASCII) text format, and vice versa |