Qfix Payment Gateway Integration Document

V0.3.2

Last modified date: 18-03-2024

Introduction

This document helps the merchant to understand the payment gateway integration enabling merchants to collect payments through Debit Card, Credit Card, Net Banking.

Merchant Onboarding Flow

- The merchant should be on-boarded with Qfix and must have access to the admin panel.
- The merchant will be provided with a private key to sign the transaction before it is initiated.
- The merchant should share the URL(s) on which the redirection will be done for the transaction initiated from Qfix Checkout.
- Any loss to the above registered private key will lead to a new key generation.
- Payments through Qfix Checkout will only be enabled only for approved merchants.

Modes of payments

Payment modes listed below are configurable:

1. FSS
2. Net Banking (CCAvenue)
3. PayZapp
4. UPI
5. BQR
6. MPGS

To enable UPI and BQR payments, register a callback URL with us. This callback URL is then used to push the updates/status of the transaction.

Security

Integrations done, must be secured where no data tampering could be allowed by man in the middle. Transactions once submitted to https:///www.eduqfix.com are fully encrypted and secured with SSL and RSA Encryptions.

In your app, make sure the parameters like customer and payment details are handled in one of the below listed ways.

1. Put all of the data, and the amount of transaction in the session of server. While redirecting the user pick the details from the session.
2. Encrypt (not encode) the details at the frontend before sending it to the server or redirecting to the PG.

The details must not be encoded as it can be decoded easily. The details must be encrypted, and the provided keys can be used in RSA, SHA256 algorithms as well.

Checkout Page Integration

All transactions can be demonstrated from the demo page
https://www.eduqfix.com/checkout/integration/demo

Demo page helps in generating the request body and displays the code snippets for integration.

A transaction can be initiated by redirecting the user to the below URL with the required parameters on a browser.

All below endpoints are to be concatenated with base URL **https://www.eduqfix.com/**

URL	Type	Description
/checkout/integration/payment	GET	Initiates a new transaction by redirecting the user to the browser
/erp/integration/verify	GET	Get the updated information of the transaction along with the status of the transaction
/erp/integration/refund	GET	Initiates a refund against a successful transaction
/erp/integration/verify/refund	GET	Get the updated information of the refund initiated against a successful transaction

/erp/integration/settlement	GET	Get the updated information of a transaction along with the settlement date.

Encryption

Below encryption code snippet can be used to encrypt the payload to be sent to the APIs.

Java:

```
public static String encrypt(String strToEncrypt, String secret, String salt) {
		try {
			IvParameterSpec ivspec = new IvParameterSpec(salt.getBytes("UTF-8"));

			MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
			messageDigest.update(secret.getBytes("UTF-8"));
			byte[] key = messageDigest.digest();
			SecretKeySpec secretKey = new SecretKeySpec(key, "AES");

			Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
			cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);

			byte[] encryptedBytes = cipher.doFinal(strToEncrypt.getBytes("UTF-8"));

			return Base64.encodeBase64URLSafeString(encryptedBytes);

		}catch(Exception e) {
			log.error("Problem while encrypting msg.", e);
		}
		return null;
	}
```

Python:

```python
from hashlib import sha256
from Crypto.Cipher import AES
import base64

def encrypt(strToEncrypt, secret, salt):
    BS = 16
    pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS)
    ivspec=salt.encode("UTF-8")
    digest = sha256(secret.encode('UTF-8')).digest()
    cipher = AES.new(digest, AES.MODE_CBC, ivspec)
    strToEncrypt=pad(strToEncrypt)
    encrypted_text = cipher.encrypt(strToEncrypt.encode('UTF-8'))
    print(base64.urlsafe_b64encode(encrypted_text))
    return base64.urlsafe_b64encode(encrypted_text)
```

PHP;

```php
<?PHP

function encrypt($string,$key,$salt) {
        $ivspec= utf8_encode($salt);
        $digest=hexToStr(hash('sha256', utf8_encode($key)));
        $cipher = openssl_encrypt($string, "AES-256-CBC", $digest, $options=OPENSSL_RAW_DATA, $ivspec);
             return rtrim(strtr(base64_encode($cipher), '+/', '-_'), '=');
}

function hexToStr($hex){
    $string='';
    for ($i=0; $i < strlen($hex)-1; $i+=2){
        $string .= chr(hexdec($hex[$i].$hex[$i+1]));
    }
    return $string;
}

?>
```

.NET:

```
public string Encrypt(string strToEncrypt, string secret, string salt)
{
    string encrypted = null;
    using (var sha256 = SHA256.Create())
    {
        byte[] Key = sha256.ComputeHash(Encoding.UTF8.GetBytes(secret));
        byte[] IV = Encoding.UTF8.GetBytes(salt);
        RijndaelManaged rj = new RijndaelManaged();
        rj.Key = Key;
        rj.IV = IV;
        rj.Mode = CipherMode.CBC;
        rj.Padding = PaddingMode.PKCS7;
        try
        {
            MemoryStream ms = new MemoryStream();
            using (CryptoStream cs = new CryptoStream(ms, rj.CreateEncryptor(Key, IV), CryptoStreamMode.Write))
            {
                using (StreamWriter sw = new StreamWriter(cs))
                {
                    sw.Write(strToEncrypt);
                    sw.Close();
                }
                cs.Close();
            }
            byte[] encoded = ms.ToArray();
            encrypted = Convert.ToBase64String(encoded);
            encrypted = encrypted.Replace('+', '-');
            encrypted = encrypted.Replace('/', '_');
            encrypted = encrypted.TrimEnd('=');

            ms.Close();
        }
        catch (CryptographicException e)
        {
            Console.WriteLine("A Cryptographic error occurred: {0}", e.Message);
            return null;
        }
        catch (UnauthorizedAccessException e)
        {
            Console.WriteLine("A file error occurred: {0}", e.Message);
            return null;
        }
        catch (Exception e)
        {
            Console.WriteLine("An error occurred: {0}", e.Message);
        }
        finally
        {
            rj.Clear();
        }
    }
    return encrypted;
}
```

Decryption:

Below decryption code snippet can be used to decrypt the encrypted response.

Java:

```
public static String decrypt(String strToDecrypt, String secret, String salt)
                        throws NoSuchAlgorithmException, InvalidKeySpecException,
NoSuchPaddingException, InvalidKeyException,
                        InvalidAlgorithmParameterException, IllegalBlockSizeException,
BadPaddingException, UnsupportedEncodingException {

                byte[] decryptedBytes = Base64.decodeBase64(strToDecrypt);

                IvParameterSpec ivspec = new IvParameterSpec(salt.getBytes("UTF-8"));

                MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
                messageDigest.update(secret.getBytes("UTF-8"));
                byte[] key = messageDigest.digest();
                SecretKeySpec secretKey = new SecretKeySpec(key, "AES");

                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
                cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);

                byte[] decryptedMessageBytes = cipher.doFinal(decryptedBytes);

                return new String(decryptedMessageBytes);
}
```

Python:

```python
from hashlib import sha256
from Crypto.Cipher import AES
import base64

def decrypt(strToDecrypt, secret, salt):
    BS = 4
    pad = lambda s: s + (BS - len(s) % BS) * "="
    ivspec=salt.encode("UTF-8")
    digest = sha256(secret.encode('UTF-8')).digest()
    decode_text=base64.urlsafe_b64decode(pad(strToDecrypt))
    cipher = AES.new(digest, AES.MODE_CBC, ivspec)
    decrypted_text = cipher.decrypt(decode_text)
    print(decrypted_text)
    return decrypted_text
```

PHP:

```php
<?PHP
function decrypt($string,$key,$salt) {

       $encode_text= base64_decode(str_pad(strtr($string, '-_', '+/'), strlen($string) % 4,
'=', STR_PAD_RIGHT));
       $ivspec= utf8_encode($salt);
       $digest=hexToStr(hash('sha256', utf8_encode($key)));
       $cipher = openssl_decrypt($encode_text, "AES-256-CBC", $digest,
$options=OPENSSL_RAW_DATA, $ivspec);
            return $cipher;
}

function hexToStr($hex){
    $string='';
    for ($i=0; $i < strlen($hex)-1; $i+=2){
        $string .= chr(hexdec($hex[$i].$hex[$i+1]));
    }
    return $string;
}
?>
```

.NET:

```
public string Decrypt(string strToDecrypt, string secret, string salt)
{
    using (var sha256 = SHA256.Create())
    {
        strToDecrypt = strToDecrypt.Replace('-', '+');
        strToDecrypt = strToDecrypt.Replace('_', '/');
        int strlen = strToDecrypt.Length;
        int pad = strlen % 4;
        if(pad > 0)
        {
            pad = 4 - pad;
        }
        string concat = new String('=',pad);
        strToDecrypt = strToDecrypt = strToDecrypt + concat;
        byte[] key = sha256.ComputeHash(Encoding.UTF8.GetBytes(secret));
        byte[] iv = Encoding.UTF8.GetBytes(salt);

        try
        {
            using (var rijndaelManaged = new RijndaelManaged { Key = key, IV = iv, Mode =
CipherMode.CBC, Padding = PaddingMode.PKCS7})
            using (var memoryStream =
            new MemoryStream(Convert.FromBase64String(strToDecrypt)))
            using (var cryptoStream =
            new CryptoStream(memoryStream,
            rijndaelManaged.CreateDecryptor(key, iv),
            CryptoStreamMode.Read))
            {
                return new StreamReader(cryptoStream).ReadToEnd();
            }
        }
        catch (CryptographicException e)
        {
            Console.WriteLine("A Cryptographic error occurred: {0}", e.Message);
            return null;
        }
    }
}
```

Initiating a transaction:

A transaction can be initiated by redirecting a user to **/payment** endpoint along with the details of the transaction.

Ex:
https://www.eduqfix.com/checkout/integration/payment?merchantId={merchantId}&command={command}&schemeCode={schemeCode}&payload={payload}

Replace the above values denoted within the curly braces {***} with the below-defined params.

Request Parameters:

Parameter	Description	Type	Mandatory
merchantId	The unique merchant id	String	Yes
command	INITIATE	String	Yes
payload	The encrypted transaction payload	String	Yes
schemeCode	The unique scheme code assigned for the transaction. There can be more than one scheme codes depending on the integration. For more information, connect with your RM.	String	Yes

Below are the parameters required in the ***payload*** request parameter from the above table separated by pipe (|) char:

Example payload param:

merchant_unique_reference_number=0000001|amount=1.00|currency_code=INR|customer_name=John Doe|customer_email=abc@gmail.com|customer_number=9999999999|customer_address=Mumbai|customer_city=Mumbai|customer_state=Maharashtra|customer_country=IND|customer_pincode=111111|return_url_code=T00001R1|udf_1=|udf_2=|udf_3=|udf_4=|udf_5=|udf_6=|udf_7=|udf_8=|udf_9=|udf_10=|date=2023-01-02|split_payment_data=|return_url=|product_details=|

Parameter	Description	Min	Max	Type	Mandatory
merchant_unique_reference_number	A unique reference number provided for each transaction initiated.	1	45	Alphanumeric No special chars allowed	Yes
amount	The amount to be charged	1.00		String value precision up to the second place	Yes
date	Date and time at which the transaction is initiated. This is the timestamp generated at the merchant system. This timestamp will not be used at Qfix for transaction time, this is just for a reference purpose.	Not Null	Not Null	String	Yes
currency_code	Only **INR** is supported for now	3	3	String	Yes
customer_name	Name of the customer	1	100	String	Yes
customer_email	Email of the customer	1	100	String	Yes
customer_number	Mobile Number of the customer	10	10	String	Yes
customer_address	Billing address of the customer. This will appear on the issued invoice of the transaction	1	200	String	Yes
customer_city	City of the billing address. This will appear on the issued invoice of the transaction	1	50	String	Yes
customer_state	State of the billing address. This will appear on the issued invoice of the transaction	1	50	String	Yes

customer_country	Country code of the billing address. This will appear on the issued invoice of the transaction **Ex: IND** for India	3	3	String	Yes
customer_pincode	Pin code of the billing address. This will appear on the issued invoice of the transaction.	1	10	String	Yes
return_url_code	This is the code assigned to the return URLs	1		String	Yes
udf_1	User-defined field 1	1	255	String	No
udf_2	User-defined field 2	1	255	String	No
udf_3	User-defined field 3	1	255	String	No
udf_4	User-defined field 4	1	255	String	No
udf_5	User-defined field 5	1	255	String	No

NOTE: All of the User-defined fields (UDFs) are not mandatory. While getting the status of a transaction these fields will be returned in the response, but no APIs are provided to query on these fields.

It is not recommended to use the same merchant code and the pair of keys on testing and production and when moving to production, the merchant will be provided with a new set of keys and the merchant code.

Encrypting "*payload*" parameter:

The ***payload*** parameter to be sent as a request parameter to the transaction initiation URL must be encrypted. Refer to Encryption implementations above for the specific scripting languages code snippet.

The **merchant_unique_reference_number** must be unique globally and cannot be reused. It is strongly recommended to store this reference number in a persistent database like MySQL, MongoDB for future references. When checking the status of a transaction from Qfix, it can only be done using the above reference number.

Decrypting the response payload

Once the transaction is finished, the user will be redirected to the URL linked with the **return_url_code** with a ***payload, schemeCode*** parameter in the response. The parameter payload will be in encrypted format and can be decrypted by using the above-mentioned code snippets for decryption:

After decrypting the **payload** from the response, below are the parameters which will be available separated by pipe (|) char.

amount=1.0|errorDescription=|paymentMode=BANK|trackId=00000001|errorCode=|result=SUCCESS|udf_5=|udf_4=|udf_3=|udf_2=|paymentId=ABCDE12345|udf_1=|currency=INR

Parameter	Description	Type
result	SUCCESS/ERROR defining whether the transaction was successful or not.	String
paymentId	A unique reference number assigned by Qfix to the transaction. The merchant can store for their reference.	String
trackId	The unique reference number generated and sent by the merchant in the initiate command of the transaction	String
paymentMode		String

amount	Amount Charged	String
currency	Currency in which the user made the payment. Ex: **INR**	String
errorCode	Error code defining the reason for the error. Below is the list of the error codes which can be exposed	String

Error codes

Code	**Description**
VVRF001	Return Url missing
VVRF002	Merchant unique reference number is missing
VVRF003	Amount is missing
VVRF004	The currency code is missing
VVRF005	Date is missing
VVRF006	Customer name is missing
VVRF007	Customer email is missing
VVRF008	Merchant Id missing
VVRF009	Invalid transaction date
VVRF010	Invalid Customer name
VVRF011	Invalid Email Id
VVRF012	Merchant code is not registered
VVRF013	Return URL id is missing
VVRF014	Merchant reference number must be alphanumeric

VVRF015	Invalid amount
VVRF016	Duplicate merchant reference number
VVRF017	Merchant reference must not be greater than 45 character
VVRF018	Currency code must be less than or equal to 3 character
VVRF019	Customer name must be less than or equal to 100 character
VVRF020	Customer email must be less than or equal to 15 character
VVRF021	Customer number must be less than or equal to 15 character
VVRF022	Customer address must be less than or equal to 200 character
VVRF023	Customer city must be less than or equal to 50 character
VVRF024	Customer state must be less than or equal to 50 character
VVRF025	Customer country code must be less than or equal to 3 character
VVRF026	Customer City must be less than or equal to 10 character
VVRF027	Invalid customer mobile number
VVRF028	Invalid command name
VVRF029	Invalid merchant id

Verifying and checking the transaction:

Once a transaction is finished, it might be required to get the updated status of the transaction in scenarios where somehow the transaction wasn't successful, or the user did not redirect to the response page successfully, or merchant's application did not capture the status in real-time.

To get the status of a transaction, send a **GET** request to **/verify** API with the below-listed request parameters.

Parameter	Description	Type	Mandatory
merchantId	The unique merchant id	String	Yes
type	The format in which the request and response is expected. Default PIPE_SEPARATED. Options: JSON/PIPE_SEPARATED Make sure the payload is in the same format of type	String	No
encPayload	Encrypted payload with the reference number for which the information is required	String	Yes
schemeCode	The unique scheme code assigned for the	String	Yes

The **encPayload** parameter will have parameter **merchantReferenceNumber={the merchant reference number}** in an encrypted format using the above-mentioned encryption method(s).

The received response body will be available either in the JSON or Pipe separated format (depending on the value specified for the responseType in the request) with two keys **merchantReferenceNumber** and **encPayload**. The key merchantReferenceNumber will have the reference number for which the status is being checked and the key encPayload will have the information of the transaction in an encrypted format.

Sample pipe separated response:

merchantReferenceNumber=12345&encPayload=status=success|qfixReferenceNumber=ABCD121QWERTY123|merchantReferenceNumber=12345|paymentMode=DEBIT_CARD|date=2019-02-10|amount=1.00|totalCharges=0.00|currency=INR|settlement_date=|errorCode=|errorDesc=

Initiating a refund:

The merchant can initiate a refund against a successful transaction by sending a **GET** request to **/refund** API along with the below request params.

Parameter	Description	Type	Mandatory
merchantId	The unique merchant id	String	Yes
type	The format in which the request and response is expected. Default PIPE_SEPERATED. Options: JSON/PIPE_SEPERATED Make sure the payload is in the same format of type	String	No
encPayload	Encrypted payload with the reference number for which the information is required	String	Yes
schemeCode	The unique scheme code assigned for the transaction. There can be more than one scheme codes depending on the integration. For	String	Yes

	more information, connect with your RM.		

The **encPayload** parameter will have parameters

merchantReferenceNumber={the merchant reference number}|refundAmount={Amount to be refunded}|refundReferenceNumber={A new reference number for this refund transaction}

in an encrypted format using the above-mentioned encryption method(s).

The received response body will be available either in the JSON or Pipe separated format (depending on the value specified for the responseType in the request) with two keys **refundReferenceNumber** and **encPayload**. The key merchantReferenceNumber will have the reference number for which the status is being checked and the key encPayload will have the information of the transaction in an encrypted format.

Sample pipe seperated response:

refundReferenceNumber=123456&encPayload=status=success|qfixReferenceNumber=ABCD121Q WERTY123|merchantReferenceNumber=12345|refundAmount=1.00|refundReferenceNumber=1234 567|transactionDate=2019-02-10 14:35:20|refundDate=2019-02-12 08:10:25|totalAmount=1.00|remainingAmount=0.0|errorCode=|errorDesc=

Checking the status of a refund:

The merchant can check the status of a refund against a successful transaction by sending a **GET** request to **/verify/refund** API along with the below request params.

Parameter	Description	Type	Mandatory
merchantId	The unique merchant id	String	Yes
type	The format in which the request and response is expected. Default PIPE_SEPERATED.	String	No

	Options: JSON/PIPE_SEPERATED Make sure the payload is in the same format of type		
encPayload	Encrypted payload with the reference number for which the information is required	String	Yes
schemeCode	The unique scheme code assigned for the transaction. There can be more than one scheme codes depending on the integration. For more information, connect with your RM.	String	Yes

The **encPayload** parameter will have parameters

refundReferenceNumber={Refund reference number for which the status has to be checked}

in an encrypted format using the above-mentioned encryption method(s).

The received response body will be available either in the JSON or Pipe separated format (depending on the value specified for the responseType in the request) with two keys **refundReferenceNumber** and **encPayload**. The key merchantReferenceNumber will have the reference number for which the status is being checked and the key encPayload will have the information of the transaction in an encrypted format.

Sample pipe separated response:

refundReferenceNumber=123456&encPayload=status=success|qfixReferenceNumber=ABCD121QWERTY123|merchantReferenceNumber=12345|refundAmount=1.00|refundReferenceNumber=1234567|transactionDate=2019-02-10 14:35:20|refundDate=2019-02-12 08:10:25|totalAmount=1.00|remainingAmount=0.0|errorCode=|errorDesc=

Getting the settlement date for a transaction:

To get the settlement date of a transaction, send a **GET** request to **/settlement** API with the below-listed request parameters.

Parameter	Description	Type	Mandatory
merchantId	The unique merchant id	String	Yes
type	The format in which the request and response is expected. Default PIPE_SEPERATED. Options: JSON/PIPE_SEPERATED Make sure the payload is in the same format of type	String	No
encPayload	Encrypted payload with the reference number for which the information is required	String	Yes
schemeCode	The unique scheme code assigned for the transaction. There can be more than one scheme codes depending on the integration. For more information, connect with your RM.	String	Yes

The **encPayload** parameter will have parameter **merchantReferenceNumber={the merchant reference number}** in an encrypted format using the above-mentioned encryption method(s).

The received response body will be available either in the JSON or Pipe separated format (depending on the value specified for the responseType in the request) with two keys **merchantReferenceNumber** and **encPayload**. The key merchantReferenceNumber will have the reference number for which the status is being checked and the key encPayload will have the information of the transaction in an encrypted format.

Sample pipe separated response:

merchantReferenceNumber=12345&encPayload=status=success|qfixReferenceNumber=ABCD121QWERTY123|merchantReferenceNumber=12345|paymentMode=DEBIT_CARD|date=2019-02-10|amount=1.00|totalCharges=0.00|currency=INR|settlement_date=2019-02-11|errorCode=|errorDesc=

Test Payment Modes: (ONLY VALID FOR TEST KIT)

1- For Cards - Connect with respective RM.
2- For Net banking - Choose "TEST NET BANKING" option from list of banks showed on QFix checkout page.