E-Mitra Services

Payment Gateway

Version – 4.1

E-Mitra

RAJCOMP INFO SERVICES LTD, 1ST FLOOR, C-BLOCK, YOJANA BHAWANTILAK MARG, C-SCHEME, JAIPUR-302005 (RAJ), INDIA

Contents

Revision History

Revision No	Revision Date	Created By	Approved By
1.0	15-DEC-2016	Jitendra Maurya	Dr. Yuvraj Singh
1.1	02-JAN-2017	Dr. Yuvraj Singh	Parag Kachhawa
2.1	25-SEP-2017	Gagandeep Singh	Dr. Yuvraj Singh
2.2	10-AUG-2018	Sanjay Atreya	Parag Kachhawa
3.0	27-NOV-2020	Akshay Sharma	Parag Kachhawa
3.1	01-Feb-2021	Akshay Sharma	Dr. Yuvraj Singh
4.0	25-June-2024	Pratap Singh Rathore	Parag Kachhawa
4.1	10-March-2025	Pratap Singh Rathore	Parag Kachhawa

Abbreviations

Abbreviation	Definition
Merchant	Department/Application who wants to integrate with e-Mitra.
Payment gateway	The facilitator who provides net banking, credit card, debit card and wallet facilities from the various bank in one place.

Introduction

This integration is used when merchant wants to accept the payments via Credit Card/ Debit Card/ Net Banking/ Wallets etc. on their web application.

In the facility, Merchant application redirects customer to the e-Mitra payment gateway page along with required transaction details. E-Mitra displays available payment options to the customer. Once payment transaction is completed, E-Mitra redirects customer to merchant application's return URL

Process flow

- Customer visits the merchant application/website and selects option for payment.
- Merchant application redirects the customer to E-Mitra page along with encrypted transaction details over the Internet.
- E-Mitra receives transaction detail and provides a list of payment options like payment gateways/ Banks/ Wallets etc to the customer for further selection.
- Once user chooses any payment gateway, E-Mitra forwards the transaction details to payment gateway.
- Customer makes the payment using the Credit Card/ Debit Card/ Net Banking/ Wallet etc at payment gateway.
- Payment gateway redirects the customer to E-Mitra along with payment details.
- E-Mitra process the payment details, logs the same and routes the response to the merchant application with details received from payment gateway along with e-Mitra transaction details.

Prerequisites

Before proceeding for integration, E-Mitra will provide the following details to the merchant so that integration can proceed.

- **Merchant Code**
  A unique code generated by E-Mitra to identify for merchant.

- **Service Id**
  A unique id to generated by E-Mitra to identify the service for which customer is making the payment.

- **Comm Type**
  Unique code is provided by E-Mitra for calculation of commission.

- **Office Code**
  Code of office where payment is to be sent.

- **Revenue Heads**
  Codes used for sent amount breakup.

- **Encryption Key**
  The symmetric encryption algorithm that is used for encryption and decryption of the data.

- **Checksum Key**
  A secret key provided to merchant for checksum calculation.

Payment URL

The purpose of this URL is to enable merchant's website/application to do online payment transactions using E-Mitra. It allows the merchant to send the transaction details through customer's web browser.

1. Payment Request (Merchant to E-Mitra)

To initiate a transaction, the merchant needs to generate a POST request which must consist of three mandatory parameters given in section 1.1.1. This POST request needs to be hit on the endpoints below endpoints

Test (Stagging) Server: *http://emitrauat.rajasthan.gov.in/emuatsso/aggregator/payment/start*

Parameter details and sample HTML form is given in later section of this document.

1.1. Request Format (Sample HTML Form)

To post the transaction details to E-Mitra, Merchant application must create a HTML form and submit this form to E-Mitra's URL using the customer's web browser.

```
<form action="<Payment URL Endpoint>" method="POST">
    <input type="hidden" name="MERCHANTCODE" value="<MERCHANTCODE>">
    <input type="hidden" name="SERVICEID" value="<SERVICEID>">
    <input type="hidden" name="ENCDATA" value="<ENCDTA>">
</form>
```

1.2. How to create "ENCDATA" parameter?

ENCDATA parameter is generated by creating the string of the request parameters with the parameters separated by two colons ("**::**") and the parameter name and value separated are by equals ("**=**") operators. Fields must not have any separators between them and must not include any null terminating characters.

Format: -

PRN=<PRN>::CHANNEL=<CHANNEL>::REQTIMESTAMP=<REQTIMESTAMP>::AMOUNT=<AMOUNT>::SUCCESSURL=<SUCCESSURL>::FAILUREURL=<FAILUREURL>::USERNAME=<USERNAME>::USERMOBILE=<USERMOBILE>::USEREMAIL=<USEREMAIL>::CONSUMERKEY=<CONSUMERKEY>::OFFICECODE=<OFFICECODE>::REVENUEHEAD=<REVENUEHEAD>::UDF1=<UDF1>::UDF2=<UDF2>::LOOKUPID=<LOOKUPID>::COMMTYPE=<COMMTYPE>::CHECKSUM=<CHECKSUM>

Here, Sequence of parameters are not important as values will be picked by their name and not by their sequence. Once, the above string is generated then this value will be encrypted using the encryption algorithm given at the end of this document in merchant's programming language. This encrypted value will be passed in the "ENCDATA" parameter.

For example, consider the sample values given in the request parameters table in section 1.4 and the encryption key is "XYZENCKEY". The ENCDATA string before encryption will be as follows

PRN=1234567890::CHANNEL=ONLINE::REQTIMESTAMP=20201127125959987::AMOUNT=12.34::SUCCESSURL=https://localhost/success::FAILUREURL=https://localhost/failed::USERNAME=Test User::USERMOBILE=9876543210::USEREMAIL=test@testmail.com::CONSUMERKEY=123456789012345::OFFICECODE=OFF123::REVENUEHEAD=RH1-10|RH2-2.34::UDF1=Sample UDF1::UDF2=Sample

UDF2::LOOKUPID=1234567890::COMMTYPE=3::CHECKSUM=cd368cb2313fc94db4bb1954c29d6f2469ef212f130177ed7f9fc70d3bdab61f

After encryption the above string following string will be produced and passed in ENCDATA parameter.

d3x9u4AXm0LqvYJCRuUayStyxQm9fkZmPzcOsGyL81UbRFZ8vR/omu2h0knzCpfnKg4NlMp4DGREVXPjRnCp8kMwbZt9I2U+FdP5K3QqIxy4QAreyarPk9ARkw0DGfS9emCzTkFmUo4wB+mFuydvkBVsWYVCnBc8rKoAref11UZ56ojDUTcwJgBP9ik6vRe+ehr2H0nR7Qeu1rYWyhmxkfp+dy+4NhGFpGTaRsQAACosOhK/g/fxd7XRwO+zHB0HxTIWeqzM7Jr7VP8w2YFXuesEnpq/exLcinoErhzpnxvH4aVou4dEwwWIAJbxshHzff/cQiY40azPZS9ZGFLz+2oabVOnL1/znEV9YHGYHtzY7mQ7bVWtuA7UghF5Czvu1MfNgUlYvgqt2XMnT5RgW8r+J5J34gItwbRnzAPdsZYwGWL0aszvtxMFCOuoFKu6/fKXOCEtOcjEn4PkQf/JQkcMYzptxX0aznQNzpcApqaFnjS0f6plcAg85WR3kEdRoaW+rkQEnKHt2vfMDWUMsymuk4nPAoeY6uZUkzTQACp+LO+uzn/tH2LULR8JYu/J

1.3. Calculation of CHECKSUM

Merchant application need to implement checksum generation logic using their system programming language. For calculation, Merchant application must create a string appending the request parameters without using any delimiter in the following format. Then hashing of this string is done using the SHA-256 algorithm.

Format: -
<MERCHANTCODE><SERVICEID><PRN><CHANNEL><REQUESTTIMESTAMP><AMOUNT><SUCCESSURL><FAILUREURL><USERNAME><USERMOBILE><USEREMAIL><CONSUMERKEY><OFFICECODE><REVENUEHEAD><UDF1><UDF2><LOOKUPID><COMMTYPE><CHECKSUMKEY>

For example, consider that checksum key is "ABCD". Then checksum of values used to create the ENCDATA parameter will be generated as follows.

Step 1: - A following string with request parameters will be created.
RISLTEST12341234567890ONLINE20201127125959987 12.34https://localhost/successhttps://localhost/failedTest User9876543210test@testmail.com123456789012345OFF123RH1-10|RH2-2.34Sample UDF1Sample UDF212345678903ABCD

Step 2: - Hashing of above string will be done using SHA-256 algorithm. It will give the following output.
cd368cb2313fc94db4bb1954c29d6f2469ef212f130177ed7f9fc70d3bdab61f

1.4. Parameters Details

Parameter	Parameter Constraints	Description
MERCHANTCODE	Required Type: Alphanumeric Length: 5 - 32	A unique code generated by E-Mitra to identify the merchant. It will be fix for all transaction. Example: - RISLTEST
SERVICEID	Required Type: Numeric Length: 4 - 6	A unique ID shared by E-Mitra to identify the service for which customer is making the payment. It will be fix for all transaction under that service. Example: - 1234

CHANNEL	Required Type: Alphanumeric Length: 5 - 10	Mode through user is making payment. Possible values are as follows. • ONLINE
PRN	Required Type: Alphanumeric Length: 6 – 20	A **unique value** generated by merchant's end to identify the transaction at their end for each payment. Example: - 1234567890
REQTIMESTAMP	Required Type: Numeric Length: 17 Format: YYYYMMddHHmmssttt (Here 'ttt' denotes the milliseconds)	Timestamp of the transaction at merchant's end at the time of sending the request. Example: - 20201127125959987
AMOUNT	Required Type: Numeric Length: 1 – 15 (Including paisa) Format: 12,2	Total amount to be collected from the customer. Example: - 12.34, 100.00
SUCCESSURL	Required Type: Alphanumeric Length: 10 – 255 Format: URL (Starts with http or https)	The URL on which E-Mitra will redirect the final response if the transaction is successful. Example: - https://localhost/success
FAILUREURL	Required Type: Alphanumeric Length: 10 - 255 Format: Valid URL (Starts with http or https)	The URL on which E-Mitra will redirect the final response if the transaction is failed. Example: - https://localhost/failed
CALLBACKURL	Optional Type: Alphanumeric Length: 10 - 255 Format: Valid URL (Starts with http or https)	The URL on which E-Mitra will post the S2S response. Example: - https://localhost/callback
USERNAME	Required Type: Alphabetic Length: 3 - 50	Name of Customer. Example: - Test User
USERMOBILE	Required Type: Numeric Length: 10	Mobile number of Customer. Example: - 9876543210
USEREMAIL	Required Type: Alphanumeric Length: **5 - 50** Format: Email	Email of Customer. Example: - test@testmail.com
OFFICECODE	Required Type: Alphanumeric Max Length: 50	Code of Office to which this payment belongs. All funds collected against particular office will be remitted by e-Mitra team to that office in their Bank A/c or E-Gras Budget heads. Emitra team will

		create these code and share the same. Example: - OFF123
CONSUMERKEY	Required Type: Alphanumeric, Hyphen (-), Underscore (_), Dot (.) & Forward Slash (/) Max Length: 100	Consumer Key is the field which uniquely identity's a transaction against that consumer. As per agreement, **Emitra system will ensure that no duplicate Successful transaction can happen with same Consumer Key within defined time span (i.e., Unique for Timespan-Consumer Key)** **Note:** If details are fetched from the bill detail API then this field must contain the value given in the CONSUMERKEY field in bill detail API's response. Otherwise, transaction would not be updated at third party server. Example: - 123456789012345
REVENUEHEAD	Required Max Length: 255 Format: Head1-Amount1|Head2-Amount2 (Multiple revenue heads can be provided)	Head wise amount breakup of AMOUNT separated with Pipe (|) and delimited with hyphen (-). Revenue Head Code's will be generated and shared by e-Mitra team. Example: - RH1-10|RH2-2.34
UDF1	Optional Type: Alphanumeric Max Length: 255	User Define Parameter, an additional field that merchant can use to send any additional information about the transaction. Example: - Sample UDF1
UDF2	Optional Type: Alphanumeric Max Length: 255	User Define Parameter, an additional field that merchant can use to send any additional information about the transaction. Example: - Sample UDF2
COMMTYPE	Required Type: Numeric Max Length: 2	A fix value shared by E-Mitra Example: - 3
LOOKUPID	Optional Type: Numeric Max Length: 50	If details are fetched from the bill detail API then this field must contain the value given in the LOOKUPID field in bill detail API's response. Otherwise, transaction would not be updated at third party server.

		For all other cases, this will be left blank. Example: - 2324442
CHECKSUM	Required Type: Alphanumeric Length: 64	It is a critical parameter used specifically to avoid any tampering of data during the transaction. Example: - dce50e8f2220df8c7e47def66c893bd1caa384cf869e6a6f54dfbce2b40d783d

2. Payment Response (E-Mitra to Merchant)

When E-Mitra receives the response from Payment Gateway, it will process the response and update the transaction status accordingly. After this a **POST response** will be sent back to the merchant **with parameters specified** in this section.

NOTE: *If the transaction is successful at E-Mitra then response will be sent to the SUCCESSURL provided in the request otherwise response will be sent to the FAILUREURL.*

2.1. Response parameters

On receipt of transaction response from Original Bank/Payment Gateway (either **SUCCESS, FAILED, PENDING**), e-Mitra will send below mentioned parameters to Merchant's return URL (SUCCESSURL, FAILUREURL) provided in request through web browser using HTTP POST method.

- MERCHANTCODE
- SERVICEID
- STATUS
- ENCDATA

Parameter details are given in later section of this document.

2.2. Parsing of ENCDATA parameter

ENCDATA will be created by the E-Mitra. On receipt of response from e-Mitra, Merchant application need to parse its value to validate the response and take appropriate action accordingly. Merchant has to use the same encryption algorithm and encryption key to decrypt the value which is used for request ENCDATA parameter while sending request for transaction.

Format of ENCDATA before encryption will be as follows
PRN=<PRN>::REQTIMESTAMP=<REQTIMESTAMP>::AMOUNT=<AMOUNT>::RECEIPTNO=<RECEIPTNO>::TRANSACTIONID=<TRANSACTIONID>::PAIDAMOUNT=<PAIDAMOUNT>::EMITRATIMESTAMP=<EMITRATIMESTAMP>::RPPTXNID=<RPPTXNID>::RPPTIMESTAMP=<RPPTIMESTAMP>::PAYMENTMODE=<PAYMENTMODE>::PAYMENTMODEBID=<PAYMENTMODEBID>::RESPONSECODE=<RESPONSECODE>::RESPONSEMESSAGE=<RESPONSEMESSAGE>::UDF1=<UDF1>::UDF2=<UDF2>::LASTTRANSACTIONDETAILS=<LASTTRANSACTIONDETAILS>::CHECKSUM=<CHECKSUM>

If LASTTRANSACTIONDETAILS field has been received in the ENCDATA parameter than it means merchant has initiated the new request with duplicate CONSUMERKEY. This parameter will contain the details of old transaction which is successful at E-Mitra. Merchant can parse this fields value as JSON object and can

initiate a status check for the given PRN in this JSON object. By doing this merchant can update the transaction status for the old PRN at their end.

For example, consider that merchant has received the below value in the response
MERCHANTCODE=RISLTEST
SERVICEID=1234
STATUS=SUCCESS
ENCDATA=mVVqHdxzalGYkcIaqv+WyGgT9oajb9wk6H6mtenUHd5jZBD3wJu9BJ0ia9Q0h/p1e1vgkHTeJVpcxbADgrqmROcnkM1d3Xnk88OtBUH7qr/zYMPfbtd/q52WhwJe8hO0FlnljUh3qRk/1U59N48R3i6jru6VEylgQjKn+E1OSP1gAVwAJ20wNH84OIINYFN82VrjuHwIwff4DfOlPdU1fTi3aDQmQG1G0YJt7fvi3X3Cg6atKzjOmgCBuuTCMLZjU2K+AmLYltzaUm4Ko6Nm6KHMt/1jhjSzRCHipvyP9T6YGNfgJL3Yn4ADxRc5ELGVcd4SpmN+Ndvyo5opX/C1qH+lm6xFVKjOxq05M7v2rWa1aDsLqVoGvF/3H47nI9k8s7G5M7RbWpEKeY3hufoonajzQSQjn/N2kYAe7y83rlr7EgJDIDn8ewakVsagNZzFY3HC4niLvGNj1fG22oqfom9p9v/lChGv/9oDUt2btqqYgge2fbgEczoE8KbSODLRPhlCQEoh8KGPzd6JqqxM+6zVlkNWVsPo0gTKPxXIV5ZfR1PgjrAnxywUrvVEovb0

After decryption of ENCDATA value given above, merchant will get the following string
PRN=1234567890::REQTIMESTAMP=20201127125959987::AMOUNT=12.34::RECEIPTNO=20000000000::TRANSACTIONID=200000000000::PAIDAMOUNT=14.34::EMITRATIMESTAMP=20201127130101123::RPPTXNID=9876543::RPPTIMESTAMP=20201127130101123::PAYMENTMODE=Billdesk(RPP)::PAYMENTMODEBID=U1234567890::RESPONSECODE=200::RESPONSEMESSAGE=Transaction is successful::UDF1=Sample UDF1::UDF2=Sample UDF2::CHECKSUM=a99f2364f8b08609bea1b7750d4cbe75b204f8980ed4e7e6e2922cef59d63d8a

NOTE: *It is mandatory requirement at Merchant Application to revalidate all parameters values received in response with those passed while creating transaction (e.g. Amount, PRN etc.). If needed, for SUCCESS transactions, 2-Factor Authentication can be used wherein Merchant Application will consume STATUS CHECK API to fetch the status of PRN and compare the values with those received in response provided in this section. And if found correct, then only take appropriate action to mark that transaction as SUCCESS in Merchant Application. Otherwise, for any deviation, notify e-Mitra team immediately.*

2.3. Calculation of CHECKSUM

Once merchant application has decrypted the ENCDATA parameter, they must generate the checksum with the response parameters. It is mandatory to match the generated checksum with the value of "CHECKSUM" parameter received in the "ENCDATA" before updating the transaction status at their end.

Hashing algorithm and checksum key will be same as used in request checksum generation.

Response Checksum Format: -
<MERCHANTCODE><SERVICEID><PRN><REQTIMESTAMP><AMOUNT><RECEIPTNO><TRANSACTIONID><PAIDAMOUNT><EMITRATIMESTAMP><STATUS><RPPTXNID><RPPTIMESTAMP><PAYMENTMODE><PAYMENTMODEBID><RESPONSECODE><RESPONSEMESSAGE><UDF1><UDF2><LASTTRANSACTIONDETAILS><CHECKSUMKEY>

For example, response parameters value given in section 2.2, the checksum will be created as follows
RISLTEST12341234567890202011271259599871 2.34200000000002000000000014.3420201127130101123SUCCESS98765432020112713010 1123Billdesk(RPP)U1234567890200Transaction is successfulSample UDF1Sample UDF2ABCDABCD

After hashing, below value will be generated
a99f2364f8b08609bea1b7750d4cbe75b204f8980ed4e7e6e2922cef59d63d8a

2.4. Parameter Details

Parameter	Parameter Constraints	Description
MERCHANTCODE	Required Type: Alphanumeric Length: 5 - 32	Same as sent in request
SERVICEID	Required Type: Numeric Length: 4 - 6	Same as sent in request
PRN	Required Type: Alphanumeric Length: 6 - 20	Same as sent in request
REQTIMESTAMP	Required Type: Numeric Length: 17 Format: YYYYMMddHHmmssttt (Here 'ttt' denotes the milliseconds)	Same as sent in request
AMOUNT	Required Type: Numeric Length: 1 -15 (Including paisa)	Same as sent in request
PAIDAMOUNT	Required Type: Numeric Length: 1 -15 (Including paisa)	Total amount paid by customer to E-Mitra. Including the E-Mitra commission & other charges and excluding the payment gateway charges (if any). If transaction is pending or failed then zero '0' will be provided in the response.
TRANSACTIONID	Required Type: Alphanumeric Length: 6 -20	A unique transaction id generated by E-Mitra for each transaction. Receipt of TRANSACTIONID does not guarantee that transaction is successful.
RECEIPTNO	Required Type: Alphanumeric Length: 6 -20	Unique receipt number generated by e-Mitra for each payment.
EMITRATIMESTAMP	Required Type: Numeric Length: 17 Format: YYYYMMddHHmmssttt (Here 'ttt' denotes the milliseconds)	Timestamp of the transaction when E-Mitra has received the request.
STATUS	Required Type: Alphabetic Length: 5 - 10	This is the status of the transaction at E-Mitra's end. This can have the following values only:

		• SUCCESS • FAILED • PENDING
RPPTXNID	Required Type: Alphanumeric Length: 6 - 20	Unique transaction reference number generated by the Rajasthan Payment Platform.
RPPTIMESTAMP	Required Type: Numeric Length: 17 Format: YYYYMMddHHmmssttt (Here 'ttt' denotes the milliseconds)	Transaction timestamp of the Rajasthan Payment Platform. If this value is not available with E-Mitra then this field will contain the current timestamp.
PAYMENTMODE	Required Type: Alphanumeric with special characters like -,)(Length: 2 - 255	Name of the payment gateway through user has paid. If this value is not available with E-Mitra then 'NA' will be provided in the response.
PAYMENTMODEBID	Required Type: Alphanumeric Length: 2 - 100	Unique transaction id of the payment gateway through user has paid. If this value is not available with E-Mitra then 'NA' will be provided in response.
RESPONSECODE	Max Length: 10	This is the response code for the transaction status. List of codes and their meanings are appended in annexures.
RESPONSEMESSAGE	Max Length: 255	This is the human-readable text message associated with the response code.
UDF1	Optional Max Length: 255	Same as sent in request
UDF2	Optional Max Length: 255	Same as sent in request
LASTTRANSACTIONDETAILS	Optional Max Length: 255	If emitra found another transaction with given consumer key and duplicate transaction is not allowed for the service than this parameter will be added in the ENCDATA parameter and checksum field. This parameter's value will be the JSON object containing the last transaction details and **PRN**, **RECEIPTNO**, **TRANSACTIONID** as its keys.

CHECKSUM	Required Max Length: 64	Checksum calculated on the response parameters.

IMPORTANT: -
Merchant must store the values of **TRANSACTIONID**, **RECEIPTNO**, **RPPTXNID** and **PAYMENTMODEBID** in its database as these values will be used for reconciliation purpose.

Generation of Receipt for Chargeback Protection: Merchant must generate Receipt for every successful transaction with details of their **PRN**, **e-Mitra TRANSACTIONID**, **e-Mitra RECEIPTNO**, **RPPTXNID** and **PAYMENTMODEBID.** For any chargeback received against this transaction, this receipt will be required to be presented to respective Bank/PG. In absence of these details, Bank/PG may settle chargeback in favour of Consumer which will result in return of money from Merchant/Department to Consumer.

2.5. Description of transaction status

Status	Description
SUCCESS	The transaction is successful at E-Mitra. All SUCCESS transactions funds will be settled in Department Bank/E-Gras account as per scheduled defined.
PENDING	Transaction status is to be confirmed at E-Mitra. Transaction may be successful or failed based on the status received from the payment gateway. Conversion of transaction status from PENDING to SUCCESS or FAILED may take time from few minutes to 2-3 days.
FAILED	The transaction is failed at E-Mitra. If money got deducted in consumer a/c then it will be refunded back to consumer within 7-10 RISL working days.

Webhook

Once the transaction has been successful at E-Mitra, the merchant application can receive the server to server (S2S) push call (Webhook) in real time along with the browser-based response by providing the callback URL in the transaction initiation request.

Merchant application can use the data received via push call to update the transaction status at its end. But care must be taken while consuming this data. Since merchant application might receive both S2S push response and redirection-based response around the same time. This can happen due to network fluctuations. So, care must be taken to ensure that such scenarios are handled too.

If merchant server is not reachable when E-Mitra is attempting the push call, then E-Mitra would reattempt it for another 2 more times. The interval between these invocations will be 5 minutes.

NOTE:

- S2S response data would be the same as the browser-based response. Please refer **section 2** of this document for more details.
- Public IPs of merchant application must be shared before. So that whitelisting can be done.
- The S2S call would be triggered only if merchant application provided the CALLBACKURL parameter in transaction initiation request.

APIs

E-Mitra has made various rest APIs for merchants. These APIs can be accessed by making a server to server call on the designated URL of each API.

1. Request Envelope

Every API requires data in request body only. Merchant must include **"Content-Type"** header to **"application/json"** to prevent request failure**.**

In addition, Merchant has to provide "**X-Api-Name**" header in each API call, otherwise request will not be processed. Value of this header will be different for each API. Sample value for this header is given in respective API section.

2. Response Envelope

The REST API uses a JSON envelope to send metadata back to clients in a way that is simple to parse and easily customized. All successful requests will return data in below format.

```
{

   "success": true,
   "message": "<message>",
   "data": {
      "MERCHANTCODE": "<MERCHANTCODE>"
   }
}
```

If something goes wrong while processing the request, the REST API will send back a **response envelope** that does not contain the "data" property. Instead of this it will contain new property "error". This error property will then contain two more properties which explain the problem that occurred.

Error response envelop will be as follows.

```
{
   "success": false,
   "message": "<message>",
   "error": {
      "code ": "<error code>",
      "reason": "<error reason>"
   }
}
```

Details of each property is given in below table

Property	Description
success	Indicate whether a request is successfully processed or not. Possible values are as follows • true – Request is successfully processed • false – Request is not processed
message	A human readable message defining the success property.
data	Contains the actual resource or resources being fetched
error	Contains the error details

code	A alphamuneric code specific to the particular error.
reason	A longer more detailed explanation of what went wrong

NOTE: In every response, E-Mitra will include a header **"X-Session-Id"**. Merchant can save its value and share with E-Mitra for debugging purpose in case of any issue.

3. Available APIs

Below API's are provided by E-Mitra

3.1. Generate OAUTH token API

The provided API is used to generate an authorization token, which is required to access and interact with all other APIs. This authorization token serves as a security measure, ensuring that only authorized users can use the APIs. Therefore, before making any API requests, it is mandatory to obtain the authorization token using this API and include it in the header as shown below:

Authorization: *Bearer <access-token>*

3.1.1 Endpoints

Test (Stagging) Server: *http://emitrauat.rajasthan.gov.in/emuatsso/oauth/merchant/token*

3.1.2 Request

```
{
    "clientId": "MOBILE_API",
    "clientSecret" : "gfdgdfgdfgfdgf"
}
```

3.1.3 Response


```
{
 "statusCode": 200,
   "statusMessage": "Success",
   "data": {
       "access_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiRW1pdHJhIl0sInVzZXJfbmFtZSI6IlNBTVBBUksyMDI0X0VNSVRSQSIsInNjb3BlIjpbInJlYWQiLCJ3cml0ZSJdLCJleHAiOjE3MTY5ODY4NzAsImF1dGhvcml0aWVzIjpbIlJPTEVfU0lHTklOIl0sImp0aSI6IjczYzFlMWYyLTU2MWQtNDJlYy05OGIzLTUyYjUyNzI3Nzk1NiIsImNsaWVudF9pZCI6IlNBTVBBUksyMDI0In0.M0WNMRsiz__AZFxbv0kCv7BsttA8wWmkilfo4n_zdtmkY2ktLiZUgOKG6JXJdUnZ96bkLEa9zGBnm_6CTS2FawVCXrheSB5M46kaVTNdgeMkBTH5N5FHGbIOW3s05Q6aiPI7WI9jM6hQSHGavTmT6ZO719mqkRxPiEiJIxQtBhqwxVXgbuAB6uBMEQOhdexblhc-jHo8Lf9OmrqLX5lnOLjKDMdIPTmpGnEG6xtnlaLVb9QFNu28CC4Sw4z5r5H12Pu5JEvY5EwWfGcjVyUfnbGsGZ1SAXwqHf-i-x1jWGHtfltyjV6uiF0KuNrfK-O_ZNy6RsgfkDBjnkWTalgyVg",
    "token_type": "bearer",
       "refresh_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOlsiRW1pdHJhIl0sInVzZXJfbmFtZSI6IlNBTVBBUksyMDI0X0VNSVRSQSIsInNjb3BlIjpbInJlYWQiLCJ3cml0ZSJdLCJhdGkiOiI3M2MxZTFmMi01NjFkLTQyZWMtOThiMy01MmI1MjcyNzc5NTYiLCJleHAiOjE3MTY5ODY4NzAsImF1dGhvcml0aWVzIjpbIlJPTEVfU0lHTklOIl0sImp0aSI6Ij
```

ZhNjFjMWZkLWZiMWQtNGYxYi05ZWFiLWQyNjEyYzEzMmFmYyIsImNsaWVudF9pZCI6IlNBTVBBUksyMDI0In0.WC90afms_9HdC73p7--1Gh51FuOcj3FnQaATVOpAxvS3ikbZrdBaQu51L3pCaRZLcOPwmYFVHq3hWci4OIbyTOHE7e-JjMFuqcxjEk-1gtny3QqfBZtxdJ7SnAXYSdpl5y8Sh29rSWcW4euGdxLrOhnyAs3rR5xL7Lvl6Xa3T37ONSuwOR9TIOER6uKr9vnvipb27xhhtYhZB7EikA0-SK3hAKJ7S16yVCmGK7z5eLU4nnLouWSS4w0IDBcl2msQFBEGhkGhgJiBq6lUGzuZlU92o3yjf0M7JArCMB_wOdGZfUq0MXLJKWy9B04_r4fj6r2eUIlEfhjqk0hPBZLxfQ",
"expires_in": 1803
}
}

3.2. Status check API

In some cases, payment response posted by the E-Mitra may not be received by the merchant application in real time due to n-number of reasons, few of which are - network issue, session timeout etc. To handle such cases, merchant can call status check API to get the current status of the transaction at E-Mitra. It is also useful as an additional layer of verification of a transaction before delivering the services to customer.

3.2.1 Endpoints

Test (Stagging) Server: *http://emitrauat.rajasthan.gov.in/emuatsso/aggregator/api/payment/status*

3.2.2 Request

To check status of any transaction, Merchant must create a JSON string which contains the four parameters "**MERCHANTCODE**", "**SERVICEID**", "**PRN**" & "**AMOUNT**" and post this string to the above-mentioned API endpoint. Value of these parameters must be same as the value passed in the transaction API.

Always set below headers with every API call.

Header Parameters		
Parameter Name	**Value**	**Required**
X-Api-Name	PAYMENT_STATUS	true
Authorization	Bearer <access-token>	true

Below is the format of JSON request

```
{
  "MERCHANTCODE": "<MERCHANTCODE>",
  "SERVICEID": "<SERVICEID>",
  "PRN": "<PRN>",
  "AMOUNT": "<AMOUNT>"
}
```

Sample Request: - (With data given in section 1.1.4)

```
POST /aggregator/api/payment/status HTTP/1.1
Host: <API Endpoint>
X-Api-Name: PAYMENT_STATUS
Content-Type: application/json
{
  "MERCHANTCODE": "RISLTEST ",
  "SERVICEID": "1234",
```

```
    "PRN": "1234567890",
    "AMOUNT": "12.34"
}
```

3.2.3 Response

E-Mitra will provide the JSON response format

```
{
    "success": true,
    "message": "<message>",
    "data": {
        "MERCHANTCODE": "<MERCHANTCODE>",
        "SERVICEID": "<SERVICEID>",
        "STATUS": "<STATUS>",
        "ENCDATA": "<ENCDATA>"
    }
}
```

Parsing of Response: -

Parsing of ENCDATA field has is done in the same way as payment response specified in Section 2 of Payment URL. Response parameters and checksum generation logic will also remain same.

Note: -

Merchant will receive the PENDING status in response if the verification is initiated within 10 minutes after initiation of the transaction.

3.3. Last transaction by consumer key

In some cases, may want to check the duplicasy of CONSUMERKEY before initiating the payment then merchant can call this API.

3.3.1 Endpoints

Test (Stagging) Server:

http://emitrauat.rajasthan.gov.in/emuatsso/aggregator/api/consumerkey/lasttransactiondetails

3.3.2 Request

To check status of any transaction, Merchant must create a JSON string that contains the four parameters "**MERCHANTCODE**", "**SERVICEID**" & "**CONSUMERKEY**" and post this string to the above-mentioned API endpoint.

Always set below headers with every API call.

Header Parameters		
Parameter Name	**Value**	**Required**
X-Api-Name	LAST_TRANSACTION_DETAILS_BY_CONSUMER_KEY	true
Authorization	Bearer <access-token>	true

Below is the format of JSON request

```
{
    "MERCHANTCODE": "<MERCHANTCODE>",
    "SERVICEID": "<SERVICEID>",
```

```
    "CONSUMERKEY": "<CONSUMERKEY>"
}
```

Sample Request: - (With data given in section 1.1.4)

```
POST /aggregator/api/consumerkey/lasttransactiondetails HTTP/1.1
Host: <API Endpoint>
X-Api-Name: LAST_TRANSACTION_DETAILS_BY_CONSUMER_KEY
Content-Type: application/json
{
    "MERCHANTCODE": "RISLTEST ",
    "SERVICEID": "1234",
    "CONSUMERKEY": "123456789012345"
}
```

3.3.3 Response

E-Mitra will provide the JSON response format

```
{
    "success": true,
    "message": "<message>",
    "data": {
        "PRN": "<PRN>",
        "RECEIPTNO": "<RECEIPTNO>"
        "TRANSACTIONID": "<TRANSACTIONID>"
        "ALLOWNEWTRANSACTION": <ALLOWNEWTRANSACTION>
    }
}
```

Parsing of Response: -

In response merchant will receive **ALLOWNEWTRANSACTION** field which determines that whether merchant can initiate the new transaction for the given CONSUMERKEY or not. Value of this field will be Boolean i.e.; merchant will receive **true** or **false** in response. Here, **true** means new transaction can be initiated and **false** means new transaction cannot be initiated.

With above field, merchant will also receive PRN, RECEIPTNO & TRANSACTIONID fields if E-Mitra finds a success transaction with given CONSUMERKEY. Merchant can use these details and call status check API to update the transaction details instead of initiating the new transaction.

3.4. Transaction cancel API

In some cases, consumer wants to cancel transaction and wants to get refund then, the merchant can use this API.

3.4.1 Endpoints

Test (Stagging) Server: *http://emitrauat.rajasthan.gov.in/emuatsso/aggregator/api/payment/refund*

3.4.2 Request

To cancel any transaction, Merchant must create a JSON string that contains the four parameters "**MERCHANTCODE**", "**SERVICEID**", **"TRANSACTIONID**" & "**CHECKSUM**" and post this string to the above-mentioned API endpoint.

*Always set below headers with every API call.

Header Parameters		
Parameter Name	**Value**	**Required**
X-Api-Name	PAYMENT_REFUND	true
Authorization	Bearer <access-token>	true

Below is the format of JSON request

```
{
   "MERCHANTCODE": "<MERCHANTCODE>",
   "SERVICEID": "<SERVICEID>",
   "TRANSACTIONID": "<TRANSACTIONID>"
    "CHECKSUM": "<CHECKSUM>"
}
```

Sample Request: - (With data given in section 1.1.4)

```
{
   "MERCHANTCODE":"RISLTESTNEWPG",
   "SERVICEID": "5477",
   "TRANSACTIONID": "210000245257",
   "CHECKSUM": "c87fd545ba36cac634690a171e9da3929b36fce5c04d50d0df20e7ef8e92caa9"
}
```

The hashing algorithm and checksum key used will be the same as those used in request checksum generation.

Checksum Generation Parameters: Merchant code + Service ID + Transaction ID + Checksum key

Example: RISLTESTNEWPG5477210000245257ChecksumKey

Hashing the above string using the SHA-256 algorithm will produce the following output:
c87fd545ba36cac634690a171e9da3929b36fce5c04d50d0df20e7ef8e92caa9

3.4.3 Response

E-Mitra will provide the JSON response format

```
{
   "success": true,
   "message": "Transaction Cancelled Successfully.",
   "data": {
      "TRANSACTIONID": "210000245257"
   }
```

```
}
```

```
{
   "success": false,
   "message": "Request could not processed.",
   "error": {
      "code": "ERR20903",
      "reason": "Transaction already cancelled."
   }
}
```

Parsing of Response: -
In response merchant will receive a Boolean response i.e.; merchant will receive **true** or **false** in response. Here, **true** means Transaction cancelled successfully with TRANSACTIONID and **false** means Transaction already cancelled or transaction not cancelled due to some reason with error code "ERR20903".

3.5. Common API to check transaction status

This API is used to check the current status of transactions at E-Mitra. While API 3.1 (**Status Check API**) is specifically for checking the status of PG transactions, this common API allows us to check the status of both wallet and PG transactions.

3.5.1 Endpoints

Test (Stagging) Server: *http://emitrauat.rajasthan.gov.in/emuatsso/aggregator/api/payment/commonVerify*

3.5.2 Request

To check status of any transaction, Merchant must create a JSON string which contains the four parameters "**MERCHANTCODE**", "**SERVICEID**", "**PRN**" & "**AMOUNT**" and post this string to the above-mentioned API endpoint. Value of these parameters must be same as the value passed in the transaction API.

*Always set below headers with every API call.

Header Parameters		
Parameter Name	**Value**	**Required**
X-Api-Name	COMMON_VERIFY	true
Authorization	Bearer <access-token>	true

Below is the format of JSON request

```
{
   "MERCHANTCODE": "<MERCHANTCODE>",
   "SERVICEID": "<SERVICEID>",
   "PRN": "<PRN>",
```

```
        "AMOUNT": "<AMOUNT>"
    }
```

Sample Request: - (With data given in section 1.1.4)

```
    POST /aggregator/api/payment/commonVerify HTTP/1.1
    Host: <API Endpoint>
    X-Api-Name: COMMON_VERIFY
    Content-Type: application/json
    {
        "MERCHANTCODE":"RISLTEST",
        "SERVICEID": "6336",
        "PRN": "1252S4S5S281333",
        "AMOUNT": 500
}
```

3.5.3 Response

E-Mitra will provide the JSON response format

```
{
  "statusCode": 200,
  "success": true,
  "message": "Request successfully processed.",
  "data": {
    "MERCHANTCODE": "RISLTEST",
    "SERVICEID": "6336",
    "STATUS": "SUCCESS",
    "PRN": "1252S4S5S281333",
    "REQTIMESTAMP": "20240430102001000",
    "AMOUNT": "500",
    "RECEIPTNO": "24000277466",
    "TRANSACTIONID": "240000286769",
    "PAIDAMOUNT": "510.00",
    "EMITRATIMESTAMP": "20240430101959510",
    "PAYMENTMODE": "Rajasthan Payment PlatForm",
    "PAYMENTMODEBID": "339871_20240430102018633",
    "RESPONSECODE": "200",
    "RESPONSEMESSAGE": "Transaction is successful",
    "CHECKSUM": "2a9be9c4a45482e4ca13f1595bea24e3a22838b44453dc6d6636c46fe7173716"
  }
}
```

```
{
  "statusCode": 200,
  "success": true,
  "message": "Request successfully processed.",
  "data": {
    "MERCHANTCODE": "RISLTEST",
    "SERVICEID": "8788",
    "STATUS": "FAILED",
    "PRN": "20240311152754580",
    "REQTIMESTAMP": "20240311152754580",
    "AMOUNT": "10",
    "RECEIPTNO": "24000274777",
```

```
    "TRANSACTIONID": "240000284187",
    "PAIDAMOUNT": "0.00",
    "EMITRATIMESTAMP": "20240311152752845",
    "PAYMENTMODE": "Rajasthan Payment PlatForm",
    "PAYMENTMODEBID": "0",
    "RESPONSECODE": "Transaction is failed",
    "RESPONSEMESSAGE": "300",
    "CHECKSUM": "9966d45ef539b9cdf4e73e3a5430ef7193b21821e6ee96d34a46f468afbeef61"
  }
}
```


Response Codes

Response Code	Response Message
200	Transaction successful
600	Transaction pending
300	Transaction failed

Error Codes

Error Code	Error Description
ERR0101	MERCHANTCODE field must not be blank.
ERR0102	Length of MERCHANTCODE field must be between {min} and {max} characters.
ERR0103	MERCHANTCODE field must be alphanumeric.
ERR0106	SERVICEID field must not be null.
ERR0107	SERVICEID field is invalid.
ERR0111	PRN field must not be blank.
ERR0112	Length of PRN field must be between {min} and {max} characters.
ERR0113	PRN field must be alphanumeric only.
ERR0116	AMOUNT field must not be null.
ERR0117	AMOUNT field can have maximum {integer} of integral digits and maximum {fraction} fractional digits.
ERR0118	AMOUNT field value must be greater than zero.
ERR0121	REQUESTTIMESTAMP field must not be blank.
ERR0122	Format of REQUESTTIMESTAMP field is {format}.
ERR0123	Difference between REQUESTTIMESTAMP field and E-Mitra server current time can be {0} minutes only.
ERR0126	SUCCESSURL field must not be blank.

ERR0127	Length of SUCCESSURL field must be between {min} and {max} characters.
ERR0128	SUCCESSURL field must be a valid URL and can have {protocols} protocols.
ERR0131	FAILUREURL field must not be blank.
ERR0132	Length of FAILUREURL field must be between {min} and {max} characters.
ERR0133	FAILUREURL field must be a valid URL and can have {protocols} protocols.
ERR0134	Length of CALLBACKURL field must be between {min} and {max} characters.
ERR0135	CALLBACKURL field must be a valid URL and can have {protocols} protocols.
ERR0136	USERNAME field must not be blank.
ERR0137	Length of USERNAME field must be between {min} and {max} characters.
ERR0138	USERNAME field can have alphabets and space.
ERR0141	USEREMAIL field must not be blank.
ERR0142	Length of USEREMAIL field must be between {min} and {max} characters.
ERR0143	USEREMAIL field must be a valid email.
ERR0146	USERMOBILE field must not be blank.
ERR0147	Length of USERMOBILE field must be {min} numbers.
ERR0148	USERMOBILE field must be numeric only.
ERR0151	OFFICECODE field must not be blank.
ERR0152	Length of OFFICECODE field must be between {min} and {max} characters.
ERR0153	OFFICECODE field must be alphanumeric.
ERR0156	CONSUMERKEY field must not be blank.
ERR0157	Length of CONSUMERKEY field must be between {min} and {max} characters.
ERR0158	CONSUMERKEY field can be alphanumeric and can have underscore(_), dot(.), hyphen(-) and forward slash(/).
ERR0159	Another transaction is already successful with the given CONSUMERKEY.
ERR0161	REVENUEHEAD field must not be blank.
ERR0162	Length of REVENUEHEAD field must be between {min} and {max} characters.
ERR0163	Format of REVENUHEAD field is invalid.
ERR0164	Invalid revenue head {0}. It must be alphanumeric and its length must not be greater than {1} characters.
ERR0165	Multiple Revenue Head {0} found.
ERR0166	Revenue Head {0}'s amount {1} is invalid.
ERR0167	Total of REVENUEHEAD amount must equals to AMOUNT field.
ERR0171	Length of UDF1 field must not be more than 50 characters.

ERR0172	UDF1 field can have alphanumeric characters and spaces.
ERR0173	Length of UDF2 field must not be more than 50 characters.
ERR0174	UDF2 field can have alphanumeric characters and spaces.
ERR0176	COMMTYPE field must not be blank.
ERR0177	Length of COMMTYPE field must not be more than {max} digits.
ERR0178	COMMTYPE field must be numeric and greater than zero only.
ERR0179	Given COMMTYPE value is not defined for the merchant.
ERR0181	Length of LOOKUPID field must not be more than {max} digits.
ERR0182	LOOKUPID field must be numeric and greater than zero only.
ERR0186	CHECKSUM field must not be blank.
ERR0187	Length of CHECKSUM field must not be more than {max} numbers.
ERR0188	CHECKSUM field is not matched.
ERR0191	CHANNEL field must not be blank.
ERR0192	Length of CHANNEL field must be between {min} and {max} characters.
ERR0193	CHANNEL field value must be in {values}.
ERR0196	KIOSKCODE field can have alphanumeric characters only.
ERR0197	Length of KIOSKCODE field must be between {min} and {max} characters.
ERR0202	Length of SSOID field must be between {min} and {max} characters.
ERR0206	KIOSKCODE and SSOID both fields must not be blank if any of these fields have value.
ERR0211	EMITRAPLUSMACHINESERIALNUMBER field can have alphanumeric characters only.
ERR0212	Length of EMITRAPLUSMACHINESERIALNUMBER field must be between {min} and {max} characters.
ERR20100	Invalid merchant code or service code or the access to this service is not enabled for the merchant.
ERR20200	Service not supported.
ERR20201	Service not found.
ERR20202	Invalid Lookup Value.
ERR20203	Invalid commission type.
ERR20204	Department charges are not configured for the service.
ERR20205	Department charges sum mismatch.
ERR20206	Error occurred while calculating the department charges.
ERR20207	Commission charges are not configured for the service.

ERR20208	Error occurred while calculating the commission charges.
ERR20209	Tax configuration not found for the service.
ERR20210	Other charges not found for the service.
ERR20211	Invalid Entity Id.
ERR20212	No user found with given SSO Id.
ERR20213	Service is not allowed for this Kiosk.
ERR20214	Emitra Plus machine is not mapped with the given Kiosk/SSO Id.
ERR20300	No session found.
ERR20600	Another transaction has already been processed with the given PRN.
ERR20601	Application type service not supported.
ERR20602	Transaction with same consumer key is already successful for another merchant.
ERR20603	Service validity expired.
ERR20604	Transaction amount is less than the minimum transaction amount defined for the service.
ERR20605	Invalid office code.
ERR20606	Office code mismatch.
ERR20607	Total amount given in REVENUEHEAD breakup is mismatch.
ERR20609	SSO Id is not related to any Kiosk
ERR20610	Emitra plus machine is not mapped to the given kiosk/sso.
ERR20611	Duplicate consumer key.
ERR20612	One or more revenue head(s) are invalid or not associated with the given service.
ERR20700	Invalid payment gateway transaction id.
ERR20701	Duplicate payment gateway transaction id.
ERR20702	Paid amount is not matched with the total transaction amount.
ERR20703	Transaction is already successful or failed.
ERR20704	Transaction posting failed as transaction was not successful.

Sample Code

1. JAVA

SHA-256 Hashing

```
private static String bytesToHex(byte[] hash) {
  StringBuilder hexString = new StringBuilder(2 * hash.length);
  for (byte element: hash) {
    String hex = Integer.toHexString(0xff & element);
    if (hex.length() == 1) {
      hexString.append('0');
    }
    hexString.append(hex);
  }
  return hexString.toString();
}

public byte[] sha256(String toBeHashString) {
  try {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    return digest.digest(toBeHashString.getBytes(StandardCharsets.UTF_8));
  } catch(Exception ex) {
    ex.printStackTrace(); //Handle exception here
  }
  return null;
}

public String sha256Hex(String toBeHashString) {
  return bytesToHex(sha256(toBeHashString));
}
```

Encryption

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class AesEncryption {

  public byte[] md5(String toBeHashString) {
    try {
      MessageDigest digest = MessageDigest.getInstance("MD5");
      return digest.digest(toBeHashString.getBytes(StandardCharsets.UTF_8));
    } catch(Exception ex) {
      ex.printStackTrace();
    }
    return null;
  }

  public byte[] sha256(String toBeHashString) {
    try {
      MessageDigest digest = MessageDigest.getInstance("SHA-256");
      return digest.digest(toBeHashString.getBytes(StandardCharsets.UTF_8));
    } catch(Exception ex) {
      ex.printStackTrace();
```

```
    }
    return null;
  }

  public String encrypt(String key, String toBeEncryptString) throws Exception {
    try {
      SecretKeySpec secretKey = new SecretKeySpec(sha256(key), "AES");
      IvParameterSpec ivParameterSpec = new IvParameterSpec(md5(key));
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivParameterSpec);
      return Base64.getEncoder().encodeToString(cipher.doFinal(toBeEncryptString.getBytes(StandardCharsets.UTF_8)));
    } catch(Exception ex) {
      throw ex;
    }
  }

  public String decrypt(String key, String toBeDecryptString) throws Exception {
    try {
      SecretKeySpec secretKey = new SecretKeySpec(sha256(key), "AES");
      IvParameterSpec ivParameterSpec = new IvParameterSpec(md5(key));
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      cipher.init(Cipher.DECRYPT_MODE, secretKey, ivParameterSpec);
      return new String(cipher.doFinal(Base64.getDecoder().decode(toBeDecryptString)));
    } catch(Exception ex) {
      throw ex;
    }
  }
}
```

Note: - *To use AES-256 encryption, Merchant has to enable unlimited cryptographic policy for Java. The JCE Policy JAR files are available in the following link:*
Oracle Java 7: - http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html
Oracle Java 8: - https://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

2. .NET

SHA-256 Hashing

```
static string CreateSHA256Hash(string rawData) {
    using (SHA256 sha256Hash = SHA256.Create()) {
        byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(rawData));
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++){
            builder.Append(bytes[i].ToString("x2"));
        }
        return builder.ToString();
    }
}
```

Encryption

```
using System;
using System.Text;
using System.Security.Cryptography;

namespace Aes256CbcEncrypterApp {
  public class MainClass {
    public static void Main(string[] args) {
      // The sample encryption key. Must be 32 characters.
```

```
    string Key = "E8AFBF4516C4E63FBE2512F9993FE";
    // The sample text to encrypt and decrypt.
    string Text = "This is sample text which needs to be encrypted using the AES 256 bits encryption algorithm.";
    // Encrypt and decrypt the sample text via the Aes256CbcEncrypter class.
    string Encrypted = Aes256CbcEncrypter.Encrypt(Text, Key);
    string Decrypted = Aes256CbcEncrypter.Decrypt(Encrypted, Key);
    // Show the encrypted and decrypted data and the key used.
    Console.WriteLine("Original: {0}", Text);
    Console.WriteLine("Key: {0}", Key);
    Console.WriteLine("Encrypted: {0}", Encrypted);
    Console.WriteLine("Decrypted: {0}", Decrypted);
  }
}

// A class to encrypt and decrypt strings using the cipher.
class Aes256CbcEncrypter {
  private static readonly Encoding encoding = Encoding.UTF8;
  public static string CreateMD5Hash(string input) {
    // Step 1, calculate MD5 hash from input
    MD5 md5 = System.Security.Cryptography.MD5.Create();
    byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(input);
    byte[] hashBytes = md5.ComputeHash(inputBytes);
    // Step 2, convert byte array to hex string
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < hashBytes.Length; i++) {
      sb.Append(hashBytes[i].ToString("X2"));
    }
    return sb.ToString();
  }

  public static string Encrypt(string plainText, string key) {
    try {
      RijndaelManaged aes = new RijndaelManaged();
      aes.KeySize = 256;
      aes.BlockSize = 128;
      aes.Padding = PaddingMode.PKCS7;
      aes.Mode = CipherMode.CBC;
      aes.Key = SHA256.Create().ComputeHash(encoding.GetBytes(key));
      aes.IV = MD5.Create().ComputeHash(encoding.GetBytes(key));
      ICryptoTransform AESEncrypt = aes.CreateEncryptor(aes.Key, aes.IV);
      byte[] buffer = encoding.GetBytes(plainText);
      return Convert.ToBase64String(AESEncrypt.TransformFinalBlock(buffer, 0, buffer.Length));
    }
    catch(Exception e) {
      throw new Exception("Error encrypting: " + e.Message);
    }
  }

  public static string Decrypt(string plainText, string key) {
    try {
      RijndaelManaged aes = new RijndaelManaged();
      aes.KeySize = 256;
      aes.BlockSize = 128;
      aes.Padding = PaddingMode.PKCS7;
      aes.Mode = CipherMode.CBC;
      aes.Key = SHA256.Create().ComputeHash(encoding.GetBytes(key));
      aes.IV = MD5.Create().ComputeHash(encoding.GetBytes(key));
      ICryptoTransform AESDecrypt = aes.CreateDecryptor(aes.Key, aes.IV);
```

```
      byte[] buffer = Convert.FromBase64String(plainText);
      return encoding.GetString(AESDecrypt.TransformFinalBlock(buffer, 0, buffer.Length));
     }
     catch(Exception e) {
      throw new Exception("Error decrypting: " + e.Message);
     }
    }
  }
}
```

3. PHP

SHA-256 Hashing

```
<?php
$sha256Hash = hash("sha256", $key);
?>
```

Encryption

```
<?php
      class Aes256Encryption
      {
         public $key;
         public $iv;
         public $method = 'AES-256-CBC';
         public function encrypt($toBeEncryptString)
         {
            return openssl_encrypt($toBeEncryptString, $this->method, hex2Bin($this->key) , 0, hex2Bin($this->iv));
         }

         public function decrypt($toBeDecryptString)
         {
            return openssl_decrypt($toBeDecryptString, $this->method, hex2Bin($this->key) , 0, hex2Bin($this->iv));
         }

         public function setKey($key)
         {
            $this->key = hash("sha256", $key);
            $this->iv = md5($key);
         }
      }
?>

<?php
      require "Aes256Encryption.php";
      $encryption_key = 'E8AFBF4516C4E63FBE2512F9993FE';
      $plainData = 'This is sample text which needs to be encrypted using the AES 256 bits encryption algorithm.';

      $encryption = new Aes256Encryption();
      $encryption->setKey($encryption_key);
      $encryptedStr = $encryption->encrypt($plainData);

      echo "Encrypted String:= ". $encryptedStr;
      echo "<br/>";
      echo "Decrypted String:= ". $encryption->decrypt($encryptedStr);

?>
```

Latest Updates

A. Handle the changes in Chrome which is resulting into more failure of transactions.

   Google Chrome Changes February 2020:
   The Chrome 80 release, scheduled for February 2020, changes the default cross-domain (Same Site) behaviour of cookies. This change enhances security and privacy, but requires customers and partners to test custom integrations that rely on cookies.