MULTI-SYS TRANSACTION API DOCUMENTATION

Base URL:

Sandbox: https://beepay-lgu.beesites.net/beepay-lgu-sandbox/public

Production: https://lgu-api.mybusybee.net

Endpoint: POST /api/multi-sys-transaction

Authentication

This API uses **HMAC-SHA256 signature authentication** using:

Name	Type	Description
apiKey	string	Public API key used to identify the client
apiKey	string	Private secret key used for generating the HMAC signature

Example credentials:

```
apiKey = test_api_key_1
secret = test_api_secret_1
```

HMAC Signature Generation

To authenticate a request, generate a **signature** using the following steps.

1. Generate Timestamp
   The timestamp must be in **ISO 8601 format**.

   ```
   const timestamp = new Date().toISOString();
   ```

   Example

2026-03-10T16:39:28+08:00

2. Prepare Request Payload
   Create the JSON payload for the transaction.

```
let reference_id = crypto.randomUUID();

const payload = `{
"amount": 1,
"reference_id": "${reference_id}",
"webhook_url": "https://portal-sandbox.nexusstrades.com/webhook/test",
"return_url": "google.com",
"remarks": "valid remarks"
}`;
```

3. Create String to Sign
   Combine the payload with the timestamp.

```
const dataToSign = payload + "timestamp=" + timestamp;
```

4. Encode Payload
   Encode the payload using **Base64**.

```
const encodedStr = btoa(payload);
```

5. Generate SHA256 Hash

```
const encodedWord  = CryptoJS.enc.Utf8.parse(dataToSign);
const wordArray    = CryptoJS.SHA256(encodedWord);
const base64String = CryptoJS.enc.Base64.stringify(wordArray);
```

6. Generate HMAC Signature
   Generate the HMAC-SHA256 signature using the **secret key**.

```
const signature = CryptoJS.HmacSHA256(base64String, secret)
 .toString(CryptoJS.enc.Hex);
```

Request Headers

Header	Example	Description
X-API-KEY	test_api_key_1	Public API key
X-TIMESTAMP	2026-03-10T16:39:28+08:00	Request timestamp
X-SIGNATURE	2545b365de594118baf9bb23ac4715dc4141143520c1a1c1ffcccf613dca6735	HMAC-SHA256 signature
X-HASH	Base64EncodedPayload	Base64 encoded payload
Content-Type	application/json	Request body type

Request Body

```
{
"amount": 1,
"reference_id": "eb3bf4bd-9523-4019-aa45-c02fd3ef2d23",
"webhook_url": "https://sample.com/webhook/test",
"return_url": "sample.com",
"remarks": "valid remarks"
}
```

Fields

Field	Type	Required	Description
amount	integer	Yes	Transaction amount
reference_id	string	Yes	Unique client reference ID
webhook_url	string	Yes	Callback URL for payment notification
return_url	string	Yes	URL where user will be redirected after payment
remarks	string	No	Additional transaction remarks

Example Implementation (JavaScript)

```
const timestamp = new Date().toISOString();
const apiKey = "test_api_key_1";
const secret = "test_api_secret_1";
let reference_id = crypto.randomUUID();
```

```
const payload = `{
"amount": 1,
"reference_id": "${reference_id}",
"webhook_url": "https://sample.com/webhook/test",
"return_url": "sample.com",
"remarks": "valid remarks"
}`;

const dataToSign = payload + "timestamp=" + timestamp;
const encodedStr = btoa(payload);

const encodedWord = CryptoJS.enc.Utf8.parse(dataToSign);
const wordArray = CryptoJS.SHA256(encodedWord);
const base64String = CryptoJS.enc.Base64.stringify(wordArray);

const signature = CryptoJS.HmacSHA256(base64String, secret)
.toString(CryptoJS.enc.Hex);
```

Example Request (cURL)

```
curl -X POST "https://beepay-lgu.beesites.net/beepay-lgu-staging/public/api/multi-sys-transaction" \
-H "X-API-KEY: test_api_key_1" \
-H "X-TIMESTAMP: 2026-03-10T16:39:28+08:00" \
-H "X-SIGNATURE:
2545b365de594118baf9bb23ac4715dc4141143520c1a1c1ffcccf613dca6735" \
-H "X-HASH: eyJhbW91bnQiOjEsInJlZmVyZW5jZV9pZCI6IjEyMyJ9" \
-H "Content-Type: application/json" \
-d '{
"amount": 1,
"reference_id": " a2b47a45-d7ab-4c1e-b0b8-faf187226327",
"webhook_url": "https://sample.com/webhook/test",
"return_url": "sample.com",
"remarks": "valid remarks"
}'
```

Response

{
"status": "success",
"data": {
"message": "Transaction created successfully",
"reference_id": "a2b47a45-d7ab-4c1e-b0b8-faf187226327",
"url": "https://beepay-lgu.beesites.net/?reference_id=a2b47a45-d7ab-4c1e-b0b8-faf187226327"
},
"signature": "2545b365de594118baf9bb23ac4715dc4141143520c1a1c1ffcccf613dca6735"
}

Error Responses

Authentication Error (401 / 400)

{
"status": "error",
"message": "Invalid signature or API key."
}

Validation Error (422)

{
"status": "error",
"message": " Invalid amount"
}

Transaction Failure (500)

```
{
"status": "error",
"message": "Transaction failed"
}
```

Verify Signature

This API uses a **multi-step HMAC-SHA256 signature mechanism** to ensure request authenticity and data integrity. The signature is generated using the request payload, timestamp, and a shared secret key.

Fields

Field	Type	Required	Description
Timestamp	string	Yes	ISO 8601 timestamp of the request
Signature	string	Yes	HMAC-SHA256 signature generated by the client
Hash	string	Yes	Base64-encoded request payload
Secret Key	string	Yes	Public identifier for the client

1. Decode the payload
   The payload is first decoded from Base64 format:

   ```
   const payload = atob(hash);
   ```

2. Reconstruct the string to sign
   The system concatenates the decoded payload with the timestamp:

   ```
   const dataToSign = payload + "timestamp=" + timestamp;
   ```

3. Generate SHA-256 hash
   The concatenated string is hashed using SHA-256 and then encoded in Base64:

   ```
   const sha256 = CryptoJS.SHA256(CryptoJS.enc.Utf8.parse(dataToSign));
   const sha256Base64 = CryptoJS.enc.Base64.stringify(sha256);
   ```

4. Generate HMAC-SHA256 signature
   The Base64-encoded hash is then signed using HMAC-SHA256 with the client's secret key:

   ```
   const expectedSignature = CryptoJS.HmacSHA256(sha256Base64, secret)
   .toString(CryptoJS.enc.Hex);
   ```

5. Compare signatures
   The generated signature is compared against the signature provided in the request:

   ```
   if (expectedSignature == signature) {
   console.log("Valid Signature");
   } else {
   console.log("Invalid Signature");
   }
   ```

Example Implementation (JavaScript)

```
// Get values from headers or globals
const timestamp = "2026-04-20T01:56:40.272Z";
const signature =
"7d4643c835ac7c15bed5c408203017582cf97d4df254f8327aed5ac76dc7b2bf";
const hash =
`eyJhbW91bnQiOiAxLC2icmVmZXJlbmNlX2lkIjogIjExMzg1ZTA2LTFkN2Et3GFkYS1hYzUzLT
ZiMGQ1MTFjN2M0NCIsICJ3ZWJob22rX3VybCI6ICJodHRwczovL3BvcnRhbC1zYW5kYm94L
m5leHVzc3RyYWRlcy5jb24vd2ViaG9vay90ZXN0IiwgInJldHVybl915mwiOiAiZ29vZ2xlLmNvb
SIsICJyZW1hcmtzIjogInZhbGlkIHJlbWFya3MifQ==`;
const secret = "sample_api_secret";

// 1. Decode payload from Base64 (same as PHP: base64_decode)
const payload = atob(hash);

// 2. Rebuild string to sign
const dataToSign = payload + "timestamp=" + timestamp;

// 3. SHA256 → Base64
const sha256 = CryptoJS.SHA256(CryptoJS.enc.Utf8.parse(dataToSign));
const sha256Base64 = CryptoJS.enc.Base64.stringify(sha256);
```

```
// 4. HMAC-SHA256 → hex
const expectedSignature = CryptoJS.HmacSHA256(sha256Base64, secret)
  .toString(CryptoJS.enc.Hex);

// 5. Compare signatures
pm.test("Signature is valid", function () {
  pm.expect(signature).to.eql(expectedSignature);
});

if(expectedSignature == signature){
  console.log('Valid Signature')
}else{
  console.log('Invalid Signature')
}

// Debug logs
console.log("Decoded Payload:", payload);
console.log("Timestamp:", timestamp);
console.log("Received Signature:", signature);
console.log("Expected Signature:", expectedSignature);
```

Webhook / Callback Response

When the payment is completed, the system will send a **POST request to the webhook_url** provided in the transaction request.

Example Callback Payload

```
{
"data": {
"amount": 1,
"payment_date": 1773305185000000,
"trx_no": "TRX_f690e865ab014887906ca3e4a7c6a24d28",
"payment_method": "gcash",
"remarks": "sample remarks",
"invoice_no": "079042",
```

"reference_no": " a2b47a45-d7ab-4c1e-b0b8-faf187226327",
"message": "Payment Received"
},
"status": "success",
"signature": "52e9790404b0c6f79b78481f05f2a4a33452b1f573fb5f4e287ca83ceeeaac19"
}

Callback Fields

Field	Type	Description
amount	integer	Payment amount
payment_date	integer	Payment timestamp
trx_no	string	Transaction number
payment_method	string	Payment method used (gcash, etc.)
remarks	string	Payment remarks
invoice_no	string	Invoice number
reference_no	string	Payment reference number
message	string	Payment status message
status	string	success or failed
signature	string	HMAC signature for webhook verification

Callback Statuses

Status	Message
success	Payment Received
pending	Payment Pending
failed	Payment Expired/ Payment Reject

OTC Webhook

Endpoint: POST /webhook/beepay-lgu-otc

This endpoint receives **OTC (Over-the-Counter) webhook events** from the Beepay LGU system. It is used to capture and store incoming transaction or event payloads for logging, tracking, and further processing.

Accepts **raw JSON payload**.

Example:

```
{
    "transaction_id": "123456",
    "amount": 1000,
    "status": "SUCCESS"
}
```

Response:

```
{
  "message": "Webhook processed successfully"
}
```

Error Response:

```
{
"error": "Error message details"
}
```

Notes

- Timestamp validation
  The server may reject requests that are too old. Always send the current ISO-8601 timestamp.
- Signature validation
  The server verifies the signature using:
  HMAC_SHA256(Base64(SHA256(payload + timestamp)), secret)
- Payload consistency
  The payload used for signature generation must match the payload sent in the request.
- Secret security
  Never expose your **secret key** in client-side applications.