

HOST-TO-HOST INTEGRATION GUIDE

Using Host-to-Host (H2H) integration, the payer inputs credit/debit card information on the merchant's website, and a payment is transmitted from the merchant server-to-server.

Flow Steps

1. Order creation

A merchant initiates a POST request in JSON format to the `/order` endpoint to create a new order.

! IMPORTANT

Use API key **only** for requests initiated from server-side (backend). **Do not share** API key with client-side applications (frontend). To call payment API from client-side use `sessionToken` received from order creation response.

Otherwise, the API key can be compromised.

```
curl --location 'https://api-pay-sandbox.amatyx.com/order' \
--header 'Authorization: Bearer sk_sand_7wYAAAAAADymgwFgPj1KH2LbvZKeybb/m2jn8L8FR7f6d6b8HU06Q' \
--header 'Content-Type: application/json' \
--data-raw '{
  "referenceNumber": 20260528164115840,
  "totalAmount": {
    "baseUnits": 20.11,
    "currency": "GBP"
  },
  "payer": {
    "contactEmail": "example@example.com"
  },
  "callbackUrl": "http://example.com/callback",
  "redirectUrl": "https://example.com/result"
}'
```

Response:

```
{
  "order": {
    "id": "J8fcAwAAAADTd5pungEAAAPYDAAA=",
    "referenceNumber": "20260528164115840",
    "status": "CREATED",
    "totalAmount": {
      "baseUnits": 20.11,
      "currency": "GBP",
      "localized": "GBP20.11",
      "minorSubunits": 2011
    },
    "redirectUrl": "https://example.com/result",
    "paymentPageUrl": "...",
    "expirationDate": "2026-05-28T13:01:17.523Z",
    "description": {},
    "requestor": {},
    "sessionToken": "eyJraWQ...5h9g",
    "payer": {
      "contactEmail": "example@example.com"
    },
    "purpose": "PURCHASE",
    "availablePaymentMethods": [
      "PAYMENT_CARD"
    ],
    "availableCardSchemes": [
      "MASTERCARD",
      "VISA"
    ],
    "merchantUrl": "https://example.com/result",
    "availableCardProductCategories": [
      "CREDIT",
      "PREPAID",
      "DEBIT"
    ],
    "availablePaymentCurrencies": [
      "GBP"
    ],
    "creationDate": "2026-05-28T12:41:17.523Z",
    "callbackUrl": "http://example.com/callback"
  }
}
```

2. Payment pre-creation

NOTE

This step is optional and should be used right after card number is known and before the payer has expressed a clear intention to make a payment (e.g. clicked the "Pay" button). The main purpose is to process hidden part of 3D-Secure flow (3DS Method) while the payer enters other payment data to achieve lower delays and better UX.

The merchant gathers all the necessary data on their website's payment page and initiates a POST (Payment Pre-creation) request to the endpoint

<https://{baseUrl}/order/{orderId}/payment> for payment creation.

If 3D-Secure Method data is received, merchant must execute 3D-Secure Method (see Processing Requirements chapter).

```
curl --location --request PUT 'https://api-pay-sandbox.amatyx.com/order/J8fcAwAAAADTd5pungEAAFYDAAA=/payment' \
--header 'Authorization: Bearer eyJraWQiOiJjcGctb3JkZXIiLCJlGsiOmsia3R5IjoiaRUMiLCJjcjYiOiJQLTUyMSIsIngiaWQiOiJkZmE1Qay1ybnNva1VEQXdnSGFPNUg2YndtODUwMWNsbTJTTeDlyeV9kSnQ3WUhrYkF4Z2XVLUdhja2Z5MWlyYV9jRjc4UmtjRm9FYVByVWZfekw1Z1I4bmImIiwieSI6IkFaT0c5aWZuMnpWb3VGTFZOVVR1UFhyV1hBb1VyWGNSZWFIrZFHnzVuU0RIN0c5clJDVGJnMFVeVWZvdmsxUy1lREo4ams5bEdGbWR6VXRvWF95WGdENTIifSwiYWxnIjoiaRUNCSC1FUYtBMjU2S1ciLCJlbnMiOiJBMjU2R0NNIn0.fAwYgNxz2LfHf9ZRazOsPCZ1Bp0rBV6EaeqNaLCX9GS6Yww6GIRpTg.TW1MzT2sJTWEbN6x.ov7D0BbDOK7ArVtMp0wFA2cQixOwX_DUHoMicFY1yosPRmnzAF7gvAQZ0540Kj4yx768t8zp0Rgs0UzpwXyb2CAdyzbxXngGc4jnzSuo-GuJHEUvQCJ_Xz1QVeMkupDFISCdaFTmUbc42Wjb72GdU-zh-LpXxDoONbu92uEqlrrLiDoPnhKoo_cceLJGSaOdSaRfPhPzlvY_Ncri1YFB73fiZl1teke0lsBtHyRR1sF4G-dpqdw0VY_v_QP9p5w.yccaT76x7pwSs1B3sn5h9g' \
--header 'Content-Type: application/json' \
--data-raw '{
  "paymentMethod": {
    "type": "PAYMENT_CARD",
    "pan": "555555000000010"
  },
  "deviceData": {
    "browserData": {
      "acceptHeader": "text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8",
      "userAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0",
      "javascriptEnabled": true,
      "javaEnabled": true,
      "language": "en-GB",
      "screenHeight": 768,
      "screenWidth": 1024,
      "timeZoneOffset": 300,
      "screenColorDepth": 24
    },
    "ip": "64.121.150.32"
  }
}'
```

Response:

```
{
  "requirements": {
    "threeDSMethod": {
      "data":
      "eyJ0aHJlZURTU2VydMvYVHJhbnNJRCI6ImQwNTU2OTcwLTlyZzZTatNDFjNS05YTJhLTly4NGIzOTBhODUwNSIsIngiaWQiOiJkZmE1Qay1ybnNva1VEQXdnSGFPNUg2YndtODUwMWNsbTJTTeDlyeV9kSnQ3WUhrYkF4Z2XVLUdhja2Z5MWlyYV9jRjc4UmtjRm9FYVByVWZfekw1Z1I4bmImIiwieSI6IkFaT0c5aWZuMnpWb3VGTFZOVVR1UFhyV1hBb1VyWGNSZWFIrZFHnzVuU0RIN0c5clJDVGJnMFVeVWZvdmsxUy1lREo4ams5bEdGbWR6VXRvWF95WGdENTIifSwiYWxnIjoiaRUNCSC1FUYtBMjU2S1ciLCJlbnMiOiJBMjU2R0NNIn0.fAwYgNxz2LfHf9ZRazOsPCZ1Bp0rBV6EaeqNaLCX9GS6Yww6GIRpTg.TW1MzT2sJTWEbN6x.ov7D0BbDOK7ArVtMp0wFA2cQixOwX_DUHoMicFY1yosPRmnzAF7gvAQZ0540Kj4yx768t8zp0Rgs0UzpwXyb2CAdyzbxXngGc4jnzSuo-GuJHEUvQCJ_Xz1QVeMkupDFISCdaFTmUbc42Wjb72GdU-zh-LpXxDoONbu92uEqlrrLiDoPnhKoo_cceLJGSaOdSaRfPhPzlvY_Ncri1YFB73fiZl1teke0lsBtHyRR1sF4G-dpqdw0VY_v_QP9p5w.yccaT76x7pwSs1B3sn5h9g",
      "url": "https://acs-sandbox.amatyx.com/threeDSMethod"
    }
  }
}
```

3. Payment execution

After the payment data is fully entered and payment started by payer, the merchant initiates a POST (Payment Flow Execution) request to the endpoint

<https://{baseUrl}/order/{orderId}/payment/execute> for payment execution.

If 3D-Secure Challenge data is received, merchant must execute 3D-Secure Challenge. See Processing Requirements chapter.

If redirectUrl is received you need to redirect browser to this URL.

Response on challenge flow:

Response on frictionless flow or decline:

4. Payment continuation

This step is required if 2nd step was skipped.

If Payment Flow Execution response contains 3D-Secure Method requirements, the merchant executes 3D-Secure Method (see [Processing Requirements chapter](#)) and then initiates a POST (Payment Flow Continuation) request to the endpoint `https://{baseUrl}/order/{orderId}/payment/continue` for payment flow continuation.

If 3D-Secure Challenge data is received, merchant must execute 3D-Secure Challenge (see [Processing Requirements chapter](#)).

If `redirectUrl` is received you need to redirect browser to this URL.

```
curl -location 'https://api-pay-sandbox.amatyx.com/order/vcfcwAAAAABkx7VungEAAFYDAAA=/payment/continue' \
--header 'Authorization: Bearer eyJraWQ0IjJjcGctb3JkZXIIiCJlcG50nsia3R5IjoiriRUMiLCJcnYiOiJQLTUyMSIsIngioiJBZE1Qay1ybNva1VEQXdnSGFPNUg2YndtODUwMWNsbTJTEdlYeV9kSnQ3WUhrYKf4ZXLVDHja2Z5MWlvYaV9jrJc4UmtJrm9FYVBvWVzfeKw1Z1I4bmImIiwieSI6IkFaTo0c5awZuMnpBw3VGTFZOVRv1UFhyVlhBbl1yWNgNsZWFIrZFHNzVuUORIN0c5clJDVGJnMFVoVVZvdmsxUy1REo4ams5bedGbWR6VXRWF95WGdENTIfSwiYXniIjoiriRUNESC1FuytBMjU2S1clLCl3bmIoIjBMjU2RONNIIn0.fAwYgNxz2LfHf9ZRazOsPCZ1Bp0rBV6EaeqNaLCX9GS6Yww6GIRPtg.TW1Mz2TSjtWEbN6x.oV7D0DbDok7ArvtMpOFA2cQiXowX_DUHOmicFY1yosPRmnzAF7gvAQ20540Kj4yx768tzp0RgsOUzpwXyb2CAdyzbXngGca4jnzSuo-GuJUEhuUe0Xsl2lPveMnhkPOoISCdaFTmUbc42vjbf72gdU-z-LpXxD0Neu92ueEqrlrliDveMphKoo_cceLJGsaOdSaSRFPhZlvY_Ncri1YFB73fiZlteke01sBtHyRR1sf4G-dpqdwOVY_v_QP9p5w.yccaT76x7pw5s1B3sn5h9g' \
--header 'Content-Type: application/json' \
--data-raw '{}'
```

Response on challenge flow:

```
{
  "requirements": {
    "threedsChallenge": {
      "data":
"eyJ0aHJlZURTU2VydmVyVHJhbnNJRCI6ImQwNTU2OTcwLTZzZTAtdmFjNS05YTJhLTlY4NGIzOTBhODUwNSIsIm1lc3NhZ2VUeXB1IjoiQ1JlcSIsIm1lc3NhZ2VWZXJzaW9uIjoiMi4yLjAiLCJhY3NUcmFuc01EIjoiYzdhMTI2NTgtMjc3Yi00MDhhLWlzeYjMtZGVhOWRlNjgyZWlxiIiwia2hhbGxlbmdlV2luZG93U2l6ZSI6IjA1In0",
      "url": "https://acs-sandbox.amatyx.com/challenge/browser"
    }
  }
}
```

Response on frictionless flow or decline:

```
{
  "redirectUrl": "https://example.com/result"
}
```

5. Redirection to final page

Redirect payer to final page using `redirectUrl` from latest response or any other URL you need to wait for completion and display payment result.

6. Receive callback with payment result

The merchant receives comprehensive information about the order state through a callback in JSON format, utilizing the POST method. See more about callbacks processing in the [Callbacks](#) chapter.

I NOTE

You can use `requestor.customData` block of order to pass some data and get it back in callbacks.

Order created callback request example:

```
POST {callbackUrl} HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Content-Type: application/json
Webhook-Event: ORDER_STATE_CHANGED
Webhook-Signature: v1=84c25c0a0f5a24e10a310005cab771dbd31d83aa77d1b4a89289a1c0b93dc38d
Webhook-Timestamp: 1779972077

{
  "order": {
    "id": "J8fcAwAAADTd5pungEAAPYDAAA=",
    "referenceNumber": "20260528164115840",
    "status": "CREATED",
    "totalAmount": {
      "baseUnits": 20.11,
      "localized": "GBP20.11",
      "minorSubunits": 2011,
      "currency": "GBP"
    },
    "redirectUrl": "https://example.com/result",
    "paymentPageUrl": "...",
    "expirationDate": "2026-05-28T13:01:17.523Z",
    "description": {},
    "requestor": {
      "application": {}
    },
    "sessionToken": "eyJraWQ...5h9g",
    "payer": {
      "contactEmail": "example@example.com"
    },
    "purpose": "PURCHASE",
    "availablePaymentMethods": [
      "PAYMENT_CARD"
    ],
    "availableCardSchemes": [
      "MASTERCARD",
      "VISA"
    ],
    "merchantUrl": "https://example.com/result",
    "availableCardProductCategories": [
      "CREDIT",
      "PREPAID",
      "DEBIT"
    ],
    "availablePaymentCurrencies": [
      "GBP"
    ],
    "creationDate": "2026-05-28T12:41:17.523Z",
    "callbackUrl": "http://example.com/callback"
  },
  "event": "ORDER_STATE_CHANGED"
}
```

Order paid callback request example:

```
POST {callbackUrl} HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Content-Type: application/json
Webhook-Event: ORDER_STATE_CHANGED
Webhook-Signature: v1=42e0dccc6652de8c2332712eb7507c6cfb37c330831b30f98906dd2094b8ea53
Webhook-Timestamp: 1779972282

{
  "order": {
    "id": "J8fcAwAAADTd5pungEAAPYDAAA=",
    "referenceNumber": "20260528164115840",
    "status": "PAID",
    "totalAmount": {
      "baseUnits": 20.11,
      "localized": "GBP20.11",
      "minorSubunits": 2011,
      "currency": "GBP"
    },
    "redirectUrl": "https://example.com/result",
    "paymentPageUrl": "...",
    "expirationDate": "2026-05-28T13:01:17.523Z",
    "description": {},
    "sessionToken": "eyJraWQ...5h9g",
    "payer": {
      "contactEmail": "example@example.com"
    },
    "purpose": "PURCHASE",
    "availablePaymentMethods": [
      "PAYMENT_CARD"
    ],
    "availableCardSchemes": [
      "MASTERCARD",
      "VISA"
    ],
    "merchantUrl": "https://example.com/result",
    "availableCardProductCategories": [
      "CREDIT",
      "PREPAID",
      "DEBIT"
    ],
    "availablePaymentCurrencies": [
      "GBP"
    ],
    "creationDate": "2026-05-28T12:41:17.523Z",
    "callbackUrl": "http://example.com/callback"
  },
  "payment": {
    "id": "h1ARAAAAAAB_ZpxungEAAPYDAAA=",
    "date": "2026-05-28T12:44:42Z",
    "rrn": "0712816082",
    "authCode": "54gr1sc",
    "result": {
      "code": "0",
      "message": "Successfully Completed"
    },
    "paymentMethod": {
      "maskedPan": "555555*0010",
      "cardScheme": "MASTERCARD",
      "panLastFourDigits": "0010",
      "type": "PAYMENT_CARD"
    },
    "amount": {
      "baseUnits": 20.11,
      "localized": "GBP20.11",
      "minorSubunits": 2011,
      "currency": "GBP"
    },
    "paymentMethodType": "PAYMENT_CARD",
    "reversed": false
  },
  "event": "ORDER_STATE_CHANGED"
}
```

Order declined callback request example:

```
POST {callbackUrl} HTTP/1.1
Accept: */*
Accept-Encoding: gzip
Content-Type: application/json
Webhook-Event: ORDER_STATE_CHANGED
Webhook-Signature: v1=42e0dccc6652de8c2332712eb7507c6cfb37c330831b30f98906dd2094b8ea53
Webhook-Timestamp: 1779972282

{
  "order": {
    "id": "9cbcAwAAAAAwg4dungEAAPYDAAA=",
    "referenceNumber": "20260528162033776",
    "status": "DECLINED",
    "totalAmount": {
      "baseUnits": 20.11,
      "localized": "GBP20.11",
      "minorSubunits": 2011,
      "currency": "GBP"
    },
    "redirectUrl": "https://example.com/result",
    "paymentPageUrl": "...",
    "expirationDate": "2026-05-28T12:40:35.248Z",
    "description": {},
    "sessionToken": "eyJraWQ...5h9g",
    "payer": {
      "contactEmail": "example@example.com"
    },
    "purpose": "PURCHASE",
    "availablePaymentMethods": [
      "PAYMENT_CARD"
    ],
    "availableCardSchemes": [
      "MASTERCARD",
      "VISA"
    ],
    "merchantUrl": "https://example.com/result",
    "availableCardProductCategories": [
      "CREDIT",
      "PREPAID",
      "DEBIT"
    ],
    "availablePaymentCurrencies": [
      "GBP"
    ],
    "creationDate": "2026-05-28T12:20:35.248Z",
    "callbackUrl": "http://example.com/callback"
  },
  "payment": {
    "id": "K1ARAAAAAConYxungEAAPYDAAA=",
    "result": {
      "code": "14",
      "message": "3D-Secure authentication could not be performed"
    },
    "paymentMethod": {
      "maskedPan": "555555**0010",
      "cardScheme": "MASTERCARD",
      "panLastFourDigits": "0010",
      "type": "PAYMENT_CARD"
    },
    "amount": {
      "baseUnits": 20.11,
      "localized": "GBP20.11",
      "minorSubunits": 2011,
      "currency": "GBP"
    },
    "paymentMethodType": "PAYMENT_CARD",
    "reversed": false
  },
  "event": "ORDER_STATE_CHANGED"
}
```

7. Fetch actual order status

! IMPORTANT

Use this method as backup if you did not receive callback in time.

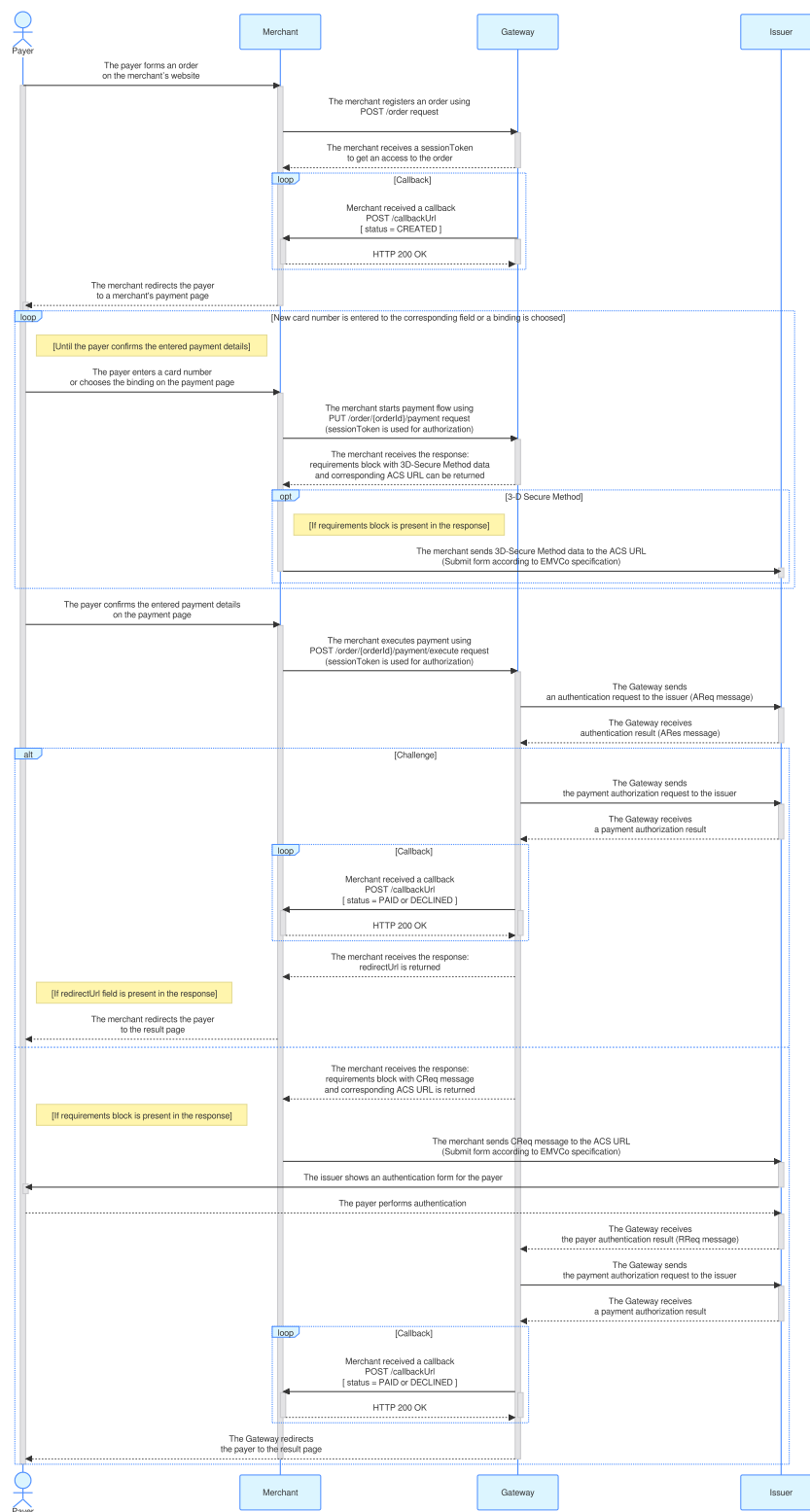
Do not use active polling instead of callbacks.

To obtain full details about the created payment, the merchant can submit a new request with the received ID from the callback to the GET (Getting an Order Details) endpoint `https://{baseUrl}/order/{orderId}`.

Response:

To see more examples, please check ***ecom openapi.yaml***

Diagram



Processing Requirements

3D-Secure Method

Create a hidden iFrame with 3DS Method form on your web-page.

```
<form method="POST" action="{url}">
  <input type="hidden" name="threeDSMethodData" value="{data}">
</form>
```

Then, submit this form. Visually nothing will happen.

3D-Secure Challenge

Create a 3DS Challenge form on your web-page.

```
<form method="POST" action="{url}">
  <input type="hidden" name="creq" value="{data}" />
</form>
```

Then, submit this form. Browser will be redirected to ACS authentication form.

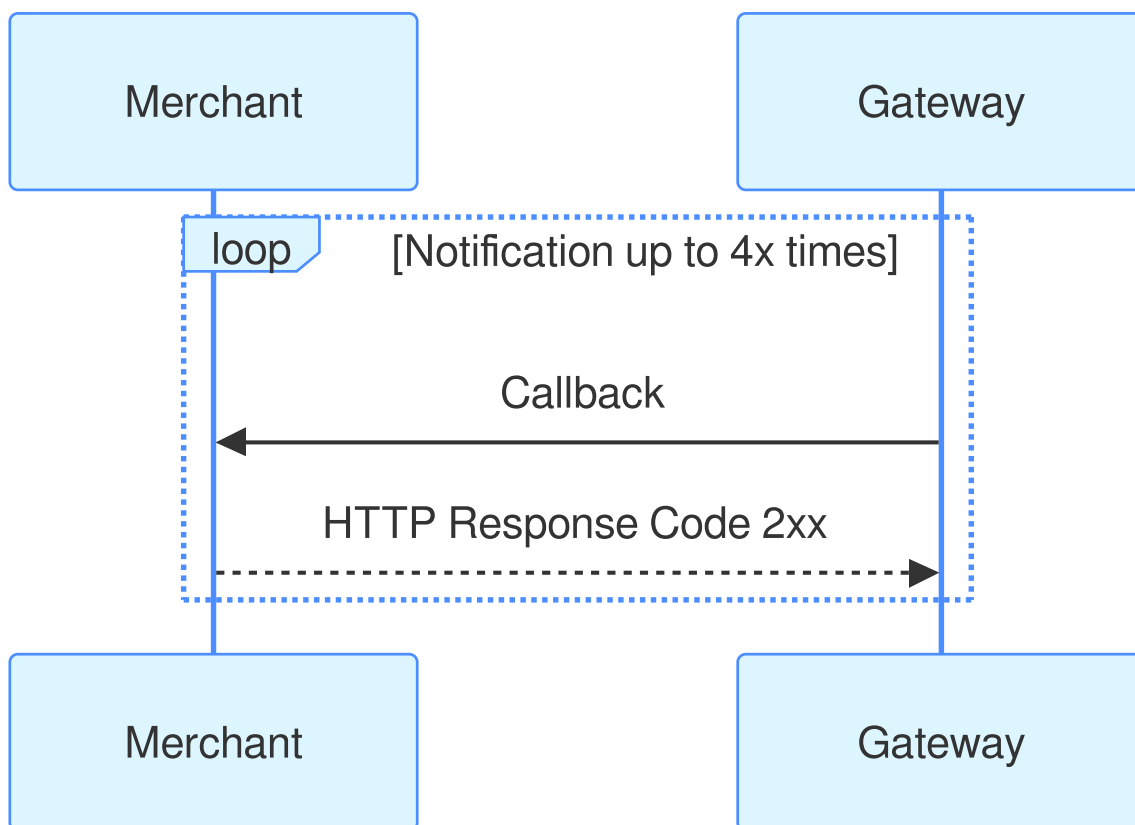
Callbacks

The API implements an asynchronous callback mechanism to notify merchants of transaction state changes.

- Real-time notifications upon transaction state changes
- Event-triggered for all key statuses (e.g., PAID, DECLINED)
- Detailed payload containing comprehensive transaction data
- Retry mechanism: 4 automatic retries on delivery failure (until HTTP 2xx response)

All callback requests are signed with security keys. The merchant must verify the request signature to avoid fraud.

1. Merchant provides a secure callbackUrl during order creation.
2. API sends HTTP POST requests with JSON payloads for each state transition.
3. Merchant responds with HTTP 2xx to acknowledge successful receipt.



Each callback request includes the `Webhook-Timestamp` and `Webhook-Signature` headers.

- `Webhook-Timestamp` is the time when the webhook signature or signatures were created, represented as a Unix timestamp in seconds.
- `Webhook-Signature` is a value or values that match your signature key(s). If you have more than one active signature key, the corresponding `Webhook-Signature` values will be represented as a comma separated list.

To verify if a callback is received from Gateway, follow the procedure below:

1. Check the `Webhook-Timestamp` value. It should not be too old. The recommended safe period is 30 seconds.
2. Prepare a payload: `Webhook-Timestamp` header + "." + `request.body` (UTF-8 encoding).
3. Calculate HMAC-SHA-256 (according to RFC 2104) for the prepared payload using your signature key (UTF-8 encoding).
4. Create the final value: `v1=` + lowercase HEX of the calculated HMAC-SHA-256 result.
5. The callback is authenticated if at least one `Webhook-Signature` value is equal to the value calculated with your signature.

See code examples in attachment.

Sandbox & testing

API key: `sk_sand_7wYAAAAAADymqwFqPj1KH2LbvZKeybb/m2jn8L8FR7f6d6b8HU06Q`

Signature key (for checking callback signatures):
`f47da4a757884bc1b52d50a3eb3e1892501ce09af02c9952eac72c950a6f23bac3b854cc0bc3875f5401d1b4cadae977cc43fb4bd6d182e612bbddfc4f62b627`

Use these cards for testing:

Card number	Card scheme	3D-Secure flow
5555550000000002	MASTERCARD	Frictionless
5555550000000010	MASTERCARD	Challenge
4111110000000005	VISA	Frictionless
4111110000000013	VISA	Challenge

Use different order amounts (any currency) to control authorization result:

Order amount (<code>baseUnits</code>)	Authorization result
< 1000	0 - Successfully Completed
5101	101 - Declined by Issuer