MEKAPAY

DEVELOPER DOCUMENTATION

Payment Gateway API

Integration guide for USDT deposits on BNB Smart Chain (BEP20) and TRON (TRC20): signed requests, per-user deposit addresses, the hosted payment page and callbacks.

Version	v1.0
Published	11 November 2026
API base URL	https://api.mekapay.online/api/gateway/v1
Online docs	https://doc.mekapay.online

Contents

The online version at doc.mekapay.online is always the latest. Every entry links to its page in this document.

GETTING STARTED

Overview 6

- How it works 6
- At a glance 6
- What you receive at onboarding 7

Quickstart 8

- Before you start 8
- Store your credentials 8
- Set your callback URL 8
- Check signing with GET /ping 8
- Create a payment order 9
- Pay the order 10
- Handle the callback 10

Authentication & signing 11

- Credentials 11
- IP whitelist (required) 11
- Request headers 13
- The string to sign 13
- Worked example: POST /orders 14
- Worked example: a GET with a query string 14
- Clock and replay window 15
- Common signing mistakes 15
- Test with GET /ping 15
- Authentication errors 15
- Rate limit 16
- Response envelope 16

CONCEPTS

Payment modes 17

- Fixed amount and any amount compared 17
- Fixed amount 17
- Any amount 18
- Which mode should I use? 18

Order lifecycle & statuses 19

- Order statuses 19
- Transitions 19
- The time window 20
- Late transfers 21
- Manual approval by MekaPay staff 21
- One address per user and chain 22
- One active order per address 22
- Deposits 22
- Unmatched deposits 23

API REFERENCE

Create an order 24

- Request body 24
- Example request 25
- Idempotency and retries 25
- Response 25
- The order object 26
- Errors 27

Retrieve / list orders 29

- Retrieve an order 29
- Find an order by your id 29
- List orders 29
- Polling 30

Cancel an order 31

- Request 31
- Rules 31
- Response 31

List deposits 33

- Query parameters 33
- The deposit object 34
- Reconciliation 34

BALANCES & SETTLEMENTS

Balance 35

- How the balance is kept 35
- Request 35
- Response 35

Settlement addresses 37

- Adding an address 37
- Rules 37
- States 37
- List your addresses 38

Settlements 39

- Fees and limits 39
- Request a settlement 39
- Statuses 40
- Cancel a settlement 41
- Retrieve and list settlements 41
- The settlement object 41
- Errors 42
- In the merchant portal 42

INTEGRATION

Payment page 43

- The payment link 43
- What the user sees 43
- Page states 43
- Returning to your site 44
- Showing the payment in your own interface 44
- Privacy and security 44

Callbacks 45

- Your callback URL 45
- Events 45
- The request 46
- Payloads 46
- Verifying the signature 50
- Worked example 50
- Responding and retries 51
- Idempotency and ordering 51

Errors 52

- Error envelope 52
- Error codes 52
- Retrying safely 53

Chains & confirmations 55

- Supported networks 55
- Detection and confirmation 55
- Amounts 55
- Wrong network or token 56

GO LIVE

Testing & go-live checklist 57

- How to test 57
- Test plan 57
- Go-live checklist 58
- Support 58

Code samples 59

- Requirements and configuration 59
- Create an order 59
- Receive and verify callbacks 63
- Self-test values 66
- Using a web framework 66

Changelog 67

- Balances and settlements · 11 November 2026 67
- Manual approval by staff · 11 November 2026 67
- IP whitelist required · 11 November 2026 67
- v1.0 · 11 November 2026 67

GETTING STARTED

MekaPay Payment Gateway

Accept USDT from your users on BNB Smart Chain (BEP20) and TRON (TRC20). Each of your users gets their own deposit address, MekaPay hosts the payment page, watches the chain and calls your server back when the payment is confirmed.

How it works

1 Your server → MekaPay API

Create a payment order

Your backend calls `POST /orders`, signed with your API Secret, with your own order id, the paying user's id, the payment mode, the chain and (for fixed-amount orders) the amount.

2 MekaPay API

Authenticate and assign an address

MekaPay verifies the signature and that the call comes from one of your whitelisted server IPs, creates an order with a unique `orderId` and `reference`, and assigns the user's deposit address. A user keeps the same address on a chain for every order.

3 MekaPay API → Your server

Return the address and a payment link

The response carries the deposit address, the amount, the expiry time, the reference and a `paymentUrl` such as `https://mekapay.online/pay/po_…`. Send the user there, or show the details in your own interface.

4 User → Blockchain

The user pays before the time limit

The user sends USDT on the right network to the address, by default within 30 minutes (you can choose 15–60). The payment page counts down and shows confirmation progress live.

5 MekaPay → Your callback URL

MekaPay confirms and calls you back

When the transfer is final on-chain, MekaPay sends a signed `payment.paid` callback to your callback URL. Your system verifies it, credits the user once, and carries on with its own flow.

At a glance

API base URL	`https://api.mekapay.online/api/gateway/v1`
Authentication	HMAC-SHA256 signature on every request: `X-MP-Client-Key`, `X-MP-Timestamp`, `X-MP-Signature`
IP whitelist	Required. API requests are accepted only from the server IP addresses MekaPay support has whitelisted for your account

Assets	USDT on BNB Smart Chain (`bsc`, BEP20) and TRON (`tron`, TRC20)
Payment modes	`fixed` (you set the amount) and `any` (the user chooses, above a minimum)
Time limit	30 minutes by default; 15–60 minutes per order
Minimum deposit	Platform floor 1 USDT on BSC, 5 USDT on TRON. Your account minimum may be higher; it applies to fixed amounts and is the default `minAmount`
Payment page	`https://mekapay.online/pay/<orderId>`, in English and Simplified Chinese
Callbacks	`payment.paid`, `payment.expired`, `deposit.unmatched`, signed with your API Secret
Format	JSON over HTTPS. Success: `{ "success": true, "data": … }`; error: `{ "success": false, "code": …, "message": … }`

What you receive at onboarding

When MekaPay onboards your business you receive an onboarding kit with:

- **Merchant code**: your account identifier. It is also your login ID for the merchant portal.
- **Merchant portal access** at portal.mekapay.online, where you see payments, deposits and callbacks, and set your callback URL. On request, MekaPay can also restrict portal access to IP addresses you list.
- **API Client Key** (`mpk_…`): identifies your requests. It is not secret on its own.
- **API Secret** (`mps_…`): signs your requests and our callbacks. It is shown **once**. Store it in your server's secret store right away.
- The API base URL and the address of this documentation.

In return, MekaPay support needs the static public IP addresses of the servers that will call the API. Requests from any other address are refused, even with valid credentials: see IP whitelist.

> **!** **Keep the API Secret on your server.**
> Anyone who has it can create orders in your name and forge callbacks to your system. Never put it in a web page, a mobile app, a repository or a log. If it leaks, ask MekaPay to rotate it: rotation replaces the key and the secret at once and the old pair stops working immediately.

GETTING STARTED

Quickstart

From credentials to a confirmed payment in six steps. Plan about an hour, plus 1 USDT on BSC for the test payment.

Before you start

- Your onboarding kit with the **API Client Key** and the **API Secret**.
- **A server whose public IP address MekaPay support has whitelisted** for your account. MekaPay refuses API requests from any other address, even when they are signed correctly, so run the steps below on that server, not on your own computer. See IP whitelist.
- `bash`, `curl` and `openssl` on that server, for the script in step 3.
- A wallet with at least 1 USDT on BNB Smart Chain (BEP20) and a little BNB for the network fee.

1 Store your credentials

Take the **API Client Key** and **API Secret** from your onboarding kit and put them in your server's environment or secret manager. The examples on this site read them from environment variables:

SHELL

```
export MEKAPAY_CLIENT_KEY="mpk_…"   # your API Client Key
export MEKAPAY_API_SECRET="mps_…"   # your API Secret (server-side only)
```

2 Set your callback URL

In the merchant portal, open **Integration** and enter the HTTPS URL that should receive callbacks, for example `https://api.your-site.example/mekapay/callback`. It must be publicly reachable over HTTPS. You can also give the URL to MekaPay during onboarding. See Callbacks.

3 Check signing with `GET /ping`

Run this script on your whitelisted server. It signs requests with `openssl` and sends them with `curl`. It first calls `/ping`, then creates a 1 USDT test order on BSC:

SHELL mekapay-quickstart.sh

```
#!/usr/bin/env bash
# mekapay-quickstart.sh: sign and send MekaPay requests with curl and openssl
#   MEKAPAY_CLIENT_KEY=mpk_... MEKAPAY_API_SECRET=mps_... bash mekapay-quickstart.sh
set -euo pipefail
HOST="${MEKAPAY_API_HOST:-https://api.mekapay.online}"

mekapay() { # usage: mekapay METHOD PATH [BODY]
  local method="$1" path="$2" body="${3:-}" timestamp signature
```

```
  timestamp="$(date +%s)"
  signature="$(printf '%s\n%s\n%s\n%s' "$timestamp" "$method" "$path" "$body" \
    | openssl dgst -sha256 -hmac "$MEKAPAY_API_SECRET" | sed 's/^.*= //')"
  curl -sS -X "$method" "$HOST$path" \
    -H 'Content-Type: application/json' \
    -H "X-MP-Client-Key: $MEKAPAY_CLIENT_KEY" \
    -H "X-MP-Timestamp: $timestamp" \
    -H "X-MP-Signature: $signature" \
    ${body:+--data-binary "$body"}
  echo
}

# 1. Check your keys and your clock.
mekapay GET /api/gateway/v1/ping

# 2. Create a 1 USDT fixed-amount order on BSC.
mekapay POST /api/gateway/v1/orders '{"merchantOrderId":"TEST-0001","userId":"test-user-
1","mode":"fixed","chain":"bsc","amount":"1"}'
```

A correct setup answers `/ping` with your merchant code:

JSON Response

```
{
  "success": true,
  "data": {
    "merchantCode": "example-merchant",
    "merchantName": "Example Merchant",
    "gatewayEnabled": true,
    "serverTime": "2026-11-11T10:00:00.000Z"
  }
}
```

If you get `INVALID_SIGNATURE` or `TIMESTAMP_OUT_OF_RANGE` instead, check common signing mistakes. A `403` with `IP_WHITELIST_REQUIRED` or `IP_NOT_WHITELISTED` means the signature is right but the request did not come from a whitelisted address: send the `ip` value from the response to MekaPay support (IP whitelist).

Create a payment order

4

The second call in the script creates a `fixed` order for 1 USDT. In your own system, create one order per payment attempt and use your own invoice or deposit id as `merchantOrderId`. Repeating the call with the same `merchantOrderId` returns the same order, so retries after a timeout are safe.

JSON Response (abridged)

```
{
  "success": true,
  "created": true,
  "data": {
    "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
    "reference": "MP2611117K2QDM",
    "merchantOrderId": "TEST-0001",
    "status": "pending",
    "chain": "bsc",
    "network": "BEP20",
    "amount": "1",
```

```
    "address": "0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed",
    "paymentUrl": "https://mekapay.online/pay/po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
    "expiresAt": "2026-11-11T10:30:00.000Z"
  }
}
```


The full response is documented in Create an order. Node.js, PHP and Python versions are in Code samples.

Pay the order

5 Open `paymentUrl` in a browser. The page shows the amount, the network, a QR code of the address and a countdown. Send exactly 1 USDT on **BNB Smart Chain (BEP20)** to the address. Within seconds the page switches to *Confirming*, and to *Paid* once the transfer is final (about 10 seconds on BSC).

Handle the callback

6 MekaPay now sends `payment.paid` to your callback URL. Your handler must:

1. verify `X-MP-Signature` over the **raw** request body (how);
2. ignore events whose `eventId` it has already processed;
3. credit `data.order.amountReceived` to `data.order.userId` at most once per `orderId`;
4. answer with any `2xx` status within 10 seconds.

Ready-to-run receivers in Node.js, PHP and Python are in Code samples. Before going live, work through the testing and go-live checklist.

GETTING STARTED

Authentication & signing

Every request to the gateway API is signed with HMAC-SHA256 using your API Secret, and must come from one of your whitelisted server IP addresses. The secret itself is never sent.

Credentials

CREDENTIAL	FORMAT	USE
Merchant code	your account id, e.g. `example-merchant`	Identifies your account in callbacks (`merchantCode`) and is your portal login ID.
API Client Key	`mpk_` + 24 letters and digits	Sent in `X-MP-Client-Key` on every request. Not secret on its own; it is shown in full in the portal.
API Secret	`mps_` + 43 characters (URL-safe base64)	Key for the request signature and for callback signatures. Shown once when it is issued. Only the last four characters are shown afterwards.

MekaPay issues and rotates credentials on request. Rotation replaces both the client key and the secret immediately; requests signed with the old pair fail with `INVALID_CLIENT_KEY`, and callbacks are signed with the new secret from then on.

IP whitelist (required)

MekaPay accepts API requests only from the server IP addresses whitelisted for your account. Before your first call, give MekaPay support the **static public IPv4 and IPv6 addresses** your servers call the API from, or CIDR ranges that contain them. A request from any other address is refused with `403`, even when its client key and signature are valid.

> ! **No whitelisted IP, no API access.**
> Until at least one address is on your whitelist, every API request fails with `403 IP_WHITELIST_REQUIRED`. After that, a request from an address that is not on the list fails with `403 IP_NOT_WHITELISTED`. Both answers include `ip`, the address MekaPay received the request from, so you can tell support exactly what to add.

RULE	DETAILS
Applies to	Every request to the signed gateway API (`/api/gateway/v1/*`, `GET /ping` included) and to the older client API (`/api/client/v1/*`).
Not affected	The hosted payment page your users open, and the callbacks MekaPay sends to your callback URL. Your users can pay from anywhere.
Entries	A single address such as `203.0.113.7` or `2001:db8::7`, or a CIDR range: IPv4 `/16` or narrower (`/16` to `/32`, e.g. `203.0.113.0/24`), IPv6 `/32` or narrower (`/32` to `/128`, e.g. `2001:db8:4a00::/48`). Up to 50 entries, each with a label such as "Production server".
Changes	Made by MekaPay support only, on your request. The merchant portal shows the list read-only.

RULE	DETAILS
Order of checks	On the gateway API the IP is checked after the signature, so a signing problem is reported first as a `401`. A `403 IP_…` answer means your key and signature are correct.

JSON 403 Forbidden (no address whitelisted yet)

```
{
  "success": false,
  "code": "IP_WHITELIST_REQUIRED",
  "message": "No server IP is whitelisted for this merchant. Ask MekaPay support to whitelist your server IP.",
  "ip": "203.0.113.7"
}
```

JSON 403 Forbidden (address not on the whitelist)

```
{
  "success": false,
  "code": "IP_NOT_WHITELISTED",
  "message": "Requests from 203.0.113.7 are not allowed for this merchant.",
  "ip": "203.0.113.7"
}
```

The whitelist is an extra lock, not a replacement for the signature: keep the API Secret as safe as before. It also signs the callbacks you receive, and the whitelist does not protect those.

How do I find my server's public IP?

Run these on the server that will call the API, not on your own computer:

SHELL

```
curl https://api.ipify.org    # the server's public IPv4 address
curl https://api6.ipify.org   # its public IPv6 address, if it has one
```

The surest answer comes from MekaPay itself: call `GET /ping` from the server. Until that address is whitelisted, the `ip` field of the `403` answer shows exactly the address MekaPay sees; send that value to support. The address must be **static**. Home and office broadband, and hosting whose outbound address changes, lock you out the moment the address changes.

What about NAT, proxies and load balancers?

- **Whitelist the address your requests leave your network from.** Not a private address (`10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`), and not the address your website is reached at.
- **NAT gateway or firewall**: servers in a private network reach the internet through it, so whitelist its public (static or elastic) IP.
- **Outbound HTTP proxy**: whitelist the proxy's public IP.
- **Load balancers and CDNs** handle traffic coming in to your site. API calls go out from your application servers or their NAT gateway, so whitelist those addresses instead.
- **Several servers, regions or a failover site**: whitelist the outbound address of each one, or a range you control.

- **Hosting without a fixed outbound IP** (many serverless, container and platform services): send API calls through a NAT gateway or proxy that has a static IP.

Does IPv6 matter?

Yes, if your server has IPv6. HTTP clients usually prefer IPv6 when both ends support it, and then the server's IPv6 address is the one MekaPay checks, even if you only whitelisted its IPv4 address. Either whitelist both addresses, or make your HTTP client connect over IPv4 only (for example `curl -4`, or `CURLOPT_IPRESOLVE` set to `CURL_IPRESOLVE_V4` in PHP). A server is often given a whole IPv6 prefix and may change its exact address within it; then whitelist the prefix your provider assigned (commonly a `/64`) rather than one address. The `ip` field of a `403` answer shows which address family a request used.

How do I change the whitelist?

Ask MekaPay support, with each address or range and a label for it. You cannot change the list yourself, through the API or in the merchant portal. Before you move to a new server, hosting provider or NAT gateway, have the new address added first, switch traffic once `GET /ping` succeeds from it, and then have the old address removed. The portal's **Security** page shows your current list; the IP it shows as yours is the computer you are browsing from, which is usually not your server.

Separately, and optionally, MekaPay can restrict the merchant portal to IP addresses you list, for example your office network. Ask support if you want this.

Request headers

HEADER	VALUE
`X-MP-Client-Key`	Your API Client Key.
`X-MP-Timestamp`	Current Unix time in **seconds**, as an integer, e.g. `1794391200`. Milliseconds are rejected.
`X-MP-Signature`	Lowercase hex `HMAC_SHA256(apiSecret, stringToSign)`: 64 characters.
`Content-Type`	`application/json` for requests with a body. A body sent with any other content type (or none) is refused with `415 UNSUPPORTED_MEDIA_TYPE`, because MekaPay can only verify the signature over a JSON body it has kept byte for byte.

The string to sign

TEXT

```
stringToSign = timestamp + "\n" + METHOD + "\n" + pathWithQuery + "\n" + rawBody
```

- `timestamp` is exactly the value you send in `X-MP-Timestamp`.
- `METHOD` is the HTTP method in upper case: `GET` or `POST`.
- `pathWithQuery` is the request path starting at `/api/`, plus the query string exactly as you send it, e.g. `/api/gateway/v1/orders?merchantOrderId=INV-10023`. It never includes the scheme or the host.
- `rawBody` is the exact body bytes you send, read as UTF-8. When there is no body (all `GET` requests and `POST /orders/:orderId/cancel`) it is the empty string, so the string to sign ends with a newline.
- The separators are single line feeds (`\n`, byte `0x0A`), never `\r\n`.

The signature is the HMAC-SHA256 of `stringToSign`, keyed with the **whole API Secret string** (including the `mps_` prefix) as UTF-8 bytes, written as lowercase hexadecimal. Do not base64-decode the secret.

Worked example: `POST /orders`

Use these values to test your implementation. The secret is a published test value, not a real credential.

API Secret	`mps_AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8`
X-MP-Timestamp	`1794391200`
METHOD	`POST`
pathWithQuery	`/api/gateway/v1/orders`
rawBody	`{"merchantOrderId":"INV-10023","userId":"player_123","mode":"fixed","chain":"bsc","currency":"USDT","amount":"100.50","expiresInMinutes":30,"returnUrl":"https://merchant.example/deposit/done","metadata":{"cartId":"C-778"}}`

stringToSign	`1794391200`
	`POST`
	`/api/gateway/v1/orders`
	`{"merchantOrderId":"INV-10023","userId":"player_123","mode":"fixed","chain":"bsc","currency":"USDT","amount":"100.50","expiresInMinutes":30,"returnUrl":"https://merchant.example/deposit/done","metadata":{"cartId":"C-778"}}`

X-MP-Signature	`59e0d2ed6ce46e58fe8f87885dc1100c3c71c36fe4e7d4ecde00ff503c77a394`

Worked example: a `GET` with a query string

X-MP-Timestamp	`1794391200`
METHOD	`GET`
pathWithQuery	`/api/gateway/v1/orders?merchantOrderId=INV-10023`
rawBody	`(empty string)`

stringToSign	`1794391200`
	`GET`
	`/api/gateway/v1/orders?merchantOrderId=INV-10023`

X-MP-Signature	`a1b80522211ffc539ab2c081439d6c6ed9fe122a007d023a69502d3bfea31f93`

Both examples use the same secret. The Node.js, PHP and Python samples in Code samples produce exactly these signatures when run with this secret at this timestamp.

Clock and replay window

MekaPay rejects a request when `X-MP-Timestamp` is more than **300 seconds** away from its own clock (`TIMESTAMP_OUT_OF_RANGE`). Keep your servers synchronised with NTP. The `serverTime` field of `GET /ping` shows MekaPay's clock if you need to compare.

Common signing mistakes

- **Signing a different body than you send.** Serialise the JSON once, sign that string and send that same string. Do not let your HTTP client re-encode an object after you signed it.
- **Leaving out** `/api/gateway/v1` or including the host in `pathWithQuery`. Sign `/api/gateway/v1/orders`, not `/orders` or `https://api.mekapay.online/…`.
- **Query strings that change after signing.** Build the query string yourself, sign it, and request exactly that URL. Parameter order and percent-encoding must be identical.
- **A missing trailing newline** on requests without a body. The string to sign still has four parts; the last is empty.
- **Milliseconds** in `X-MP-Timestamp` (`Date.now()` in JavaScript). Divide by 1000 and drop the fraction.
- **Upper-case hex or base64 output.** Send the 64-character lowercase hex digest.
- **A body without** `Content-Type: application/json`**.** Form-encoded or plain-text bodies are refused with `415 UNSUPPORTED_MEDIA_TYPE` before the signature is checked. Requests without a body need no content type.

Test with `GET /ping`

`GET` `/api/gateway/v1/ping`

Returns your account details if the request is signed correctly. It creates nothing and is safe to call at any time.

FIELD	DESCRIPTION
`merchantCode`	Your merchant code.
`merchantName`	Your business name as shown on the payment page.
`gatewayEnabled`	Whether the payment gateway is switched on for your account.
`serverTime`	MekaPay's current time (ISO 8601, UTC).

Authentication errors

HTTP	CODE	MEANING
401	`MISSING_AUTH_HEADERS`	One of the three `X-MP-*` headers is missing or empty.
401	`INVALID_CLIENT_KEY`	The client key is unknown, revoked or replaced by a rotation.
401	`TIMESTAMP_OUT_OF_RANGE`	The timestamp is malformed or more than 300 seconds from MekaPay's clock.
401	`INVALID_SIGNATURE`	The signature does not match. See the mistakes above.

HTTP	CODE	MEANING
415	`UNSUPPORTED_MEDIA_TYPE`	The request has a body that was not sent as JSON. Send it with `Content-Type: application/json`.
403	`GATEWAY_DISABLED`	The payment gateway is not enabled for your account yet.
403	`PROJECT_SUSPENDED`	Your merchant account is not active. Contact MekaPay.
403	`IP_WHITELIST_REQUIRED`	No server IP is whitelisted for your account yet. `ip` in the response is the address the request came from. See IP whitelist.
403	`IP_NOT_WHITELISTED`	The request came from an address (`ip` in the response) that is not on your account's whitelist.
429	`RATE_LIMITED`	More than 300 requests in a minute for this client key.

Rate limit

Each client key may make **300 requests per minute**. Above that, requests fail with `429 RATE_LIMITED`; wait a few seconds and retry with exponential back-off. Prefer callbacks over polling to stay well below the limit.

Response envelope

Every gateway endpoint answers in the same envelope. Branch on `success` and `code`; `message` is for people and may change.

JSON Success

```
{ "success": true, "data": { … } }
```

JSON Error

```
{ "success": false, "code": "INVALID_SIGNATURE", "message": "Signature does not match" }
```

CONCEPTS

Payment modes

Choose the mode per order in the `mode` field: `fixed` when you know the amount, `any` when the user decides how much to send.

Fixed amount and any amount compared

	`FIXED`	`ANY`
`amount`	Required	Must be absent or `null`
`minAmount`	Must be absent or `null`	Optional; defaults to your account minimum for the chain. A value you send only has to clear the platform floor
Your account minimum	`amount` must be at least this	The default `minAmount`
The order is paid when	Confirmed transfers inside the order window add up to at least `amount`	The first confirmed transfer of at least `minAmount` inside the window arrives
Several transfers	Add up (the user can pay in parts)	The first qualifying transfer pays; later ones are unmatched (`order_already_paid`)
A transfer below the minimum	Counts toward the total	Unmatched (`below_minimum`); the order keeps waiting
More than required	Paid, with `overpaid: true`	Paid; `amountReceived` is what was sent
Payment page shows	The exact amount, what is left to pay, a copy button	"Any amount · minimum X USDT"
Typical use	Checkout, invoices, fixed top-up packages	Wallet or balance top-ups

Fixed amount

Send `amount` as a decimal string with at most 6 decimal places, for example `"100.50"`. It must be at least your **account minimum** for the chain and at most `1000000`. The account minimum is the platform floor (1 USDT on BSC, 5 USDT on TRON) unless a higher minimum was agreed for your account; a lower amount is refused with `400 AMOUNT_BELOW_MINIMUM`. MekaPay stores and returns amounts normalised, so `"100.50"` comes back as `"100.5"`.

The payment page asks for exactly that amount. MekaPay adds up every confirmed transfer to the address inside the order window:

- **Exact or more**: the order becomes `paid`. If the total is greater than `amount`, `overpaid` is `true` and `amountReceived` shows the total.
- **Less**: the order stays `pending` and the page shows the remaining balance. The user can send the rest before the time limit.
- **Time runs out with a partial payment**: the order becomes `expired` with `underpaid: true`. The funds have arrived; decide with the user whether to credit the partial amount. The payment page tells the user to contact

you with the order reference. If a transfer sent in time confirms late and completes the order, you then receive `payment.paid` for the same order, with `amountReceived` covering the partial amount too: see Crediting rule.

- **A transfer sent in time confirms after expiry**: if it completes the amount, the order still becomes `paid` and you receive `payment.paid`. If it does not, it is reported on its own as `deposit.unmatched` (`after_expiry`). See Late transfers.

> ! **Exchange withdrawals often arrive short.**
> Many exchanges deduct their withdrawal fee from the amount sent. In fixed mode the order then stays pending until the user sends the difference. Ask users to make sure the amount *received* covers the order.

Any amount

Leave out `amount`. Optionally set `minAmount`. Without it, the order uses your account minimum for the chain. A `minAmount` you send only has to be at least the platform floor (1 USDT on BSC, 5 USDT on TRON), even if your account minimum is higher, so you can accept smaller top-ups on a particular order. The first confirmed transfer of at least `minAmount` pays the order, and `amountReceived` holds the amount actually sent: credit that amount.

A transfer below `minAmount` does not pay the order. MekaPay reports it with a `deposit.unmatched` callback (`unmatchedReason: "below_minimum"`) and the order keeps waiting for a qualifying transfer until it expires.

Which mode should I use?

- Use **fixed** when the payment settles a specific price: an invoice, a basket, a package. You know exactly what was owed and what arrived.
- Use **any** for "deposit to your balance" flows where users choose the amount in their wallet. It avoids failed payments caused by fees deducted on the way.

Both modes use the same address for the same user and chain, the same time limit rules and the same callbacks. See Order lifecycle & statuses.

CONCEPTS

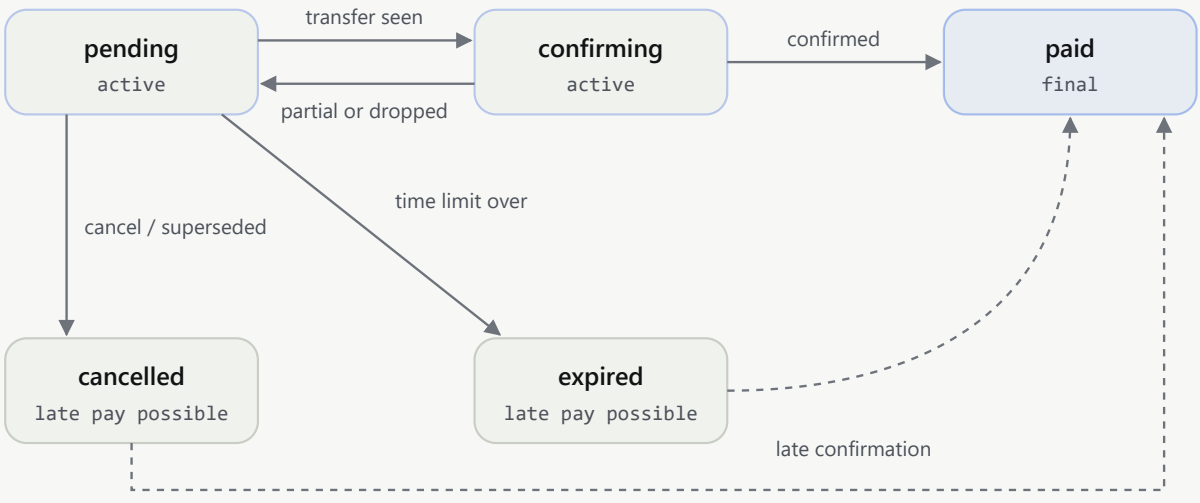
Order lifecycle & statuses

An order is *active* while it waits for money, and ends *paid*, *expired* or *cancelled*. Each transfer into the address is tracked separately as a deposit.

Order statuses

STATUS	MEANING	FINAL?
`pending`	Waiting for the user's transfer. Also the status of a fixed-amount order that has received part of the amount.	No (active)
`confirming`	A transfer to the address has been seen on-chain and is waiting for confirmations.	No (active)
`paid`	Confirmed funds satisfy the order, or MekaPay staff approved it by hand (see Manual approval). MekaPay sends `payment.paid`.	Yes
`expired`	The time limit passed without payment. MekaPay sends `payment.expired`.	Almost: it can still become `paid` if a transfer sent in time confirms late and completes it
`cancelled`	Cancelled by you, replaced by a newer order for the same user and chain, or cancelled by MekaPay staff (`cancelReason`: `merchant`, `superseded`, `staff`).	Almost: it can still become `paid` if a transfer sent before the cancellation confirms afterwards and completes it

Transitions



A fixed-amount order goes back to `pending` when a confirmed transfer covers only part of the amount. Dashed lines: a transfer sent before the order expired or was cancelled that confirms afterwards and completes the order.

FROM	TO	WHEN	CALLBACK
`pending`	`confirming`	A transfer to the address is detected inside the order window.	none
`confirming`	`paid`	The transfer is final and the confirmed total satisfies the order.	`payment.paid`
`confirming`	`pending`	The confirmed transfer was only part of a fixed amount, or the transfer was dropped from the chain.	none
`pending`	`expired`	At the earliest 180 seconds after `expiresAt`, once MekaPay has read the chain up to `expiresAt` + 120 seconds and no transfer is still confirming.	`payment.expired`
`pending`	`cancelled`	You cancel it, a new order for the same user and chain replaces it, or MekaPay staff cancel it.	none
`expired`	`paid`	A transfer sent inside the window confirms after the order expired and completes it.	`payment.paid`
`cancelled`	`paid`	A transfer sent before the cancellation confirms afterwards and completes the order.	`payment.paid`
`pending` or `expired`	`paid`	MekaPay staff approve the order by hand. See Manual approval.	`payment.paid`

The time window

Each order has a time limit: `expiresInMinutes` when you create it (15–60), or your account default (30 minutes). Users copy the address, open a wallet and sometimes withdraw from an exchange, which can take 5–20 minutes, so 30 minutes is a good default. MekaPay is lenient at the edges so that a payment sent in time is never lost:



`createdAt`
Transfers count from the moment the order is created.

`expiresAt`
Default 30 min later. The payment page countdown ends here.

`expiresAt + 120 s`
Last block time that still counts for the order.

`expiresAt + 180 s`
Earliest expiry. The order becomes `expired` once MekaPay has read the chain up to `expiresAt` + 120 s and no transfer is still confirming.

- **The transfer's block time decides.** A transfer belongs to an order when its block time is between `createdAt` and `expiresAt` + 120 seconds, however late MekaPay sees or confirms it.
- The order switches to `expired` at the earliest 180 seconds after `expiresAt`, and only once MekaPay's chain scanner has read every block up to `expiresAt` + 120 seconds, and not while a transfer for it is still confirming. If the scanner is catching up, `payment.expired` comes a little later than `expiresAt` + 3 minutes; it never comes before a transfer sent in time has been seen.
- A transfer sent inside the window that confirms after expiry still pays the order (`expired` → `paid`) if it completes it, and you receive `payment.paid` after the earlier `payment.expired`. See Late transfers.

Late transfers

Sometimes a transfer that belongs to an order is confirmed only after the order has already ended: the order expired while the transfer was still on its way, or you cancelled the order just after the user sent the money. MekaPay then decides by the transfer's block time:

ORDER	TRANSFER	RESULT
`expired`	Block time inside the window, and the order's confirmed total plus this transfer satisfies the order (fixed: reaches `amount`; any: this transfer is at least `minAmount`)	The order becomes `paid`: `payment.paid`, after the earlier `payment.expired`
`expired`	Block time inside the window, but the order is still not satisfied	`deposit.unmatched` with `after_expiry` (or `below_minimum` for a transfer below an any-amount order's `minAmount`); the order stays `expired`
`cancelled`	Block time before `cancelledAt`, and it satisfies the order as above	The order becomes `paid`: `payment.paid`
`cancelled`	Block time before `cancelledAt` but the order is not satisfied, or block time after `cancelledAt`	`deposit.unmatched` with `order_cancelled`; the order stays `cancelled`
`paid`	Any further transfer, also after a manual approval	`deposit.unmatched` with `order_already_paid`

An order that has ended never becomes partly paid by a late transfer: the transfer either completes it, or you are told about it separately. **Every confirmed deposit is reported to you**: either inside a `payment.paid` or `payment.expired` callback (in `order.deposits`), or in its own `deposit.unmatched` callback. A partial deposit of an expired order appears in `payment.expired` and again in a later `payment.paid` if a late transfer completes the order, so credit by order total (see Crediting rule).

Manual approval by MekaPay staff

MekaPay staff can mark a `pending` or `expired` order as paid by hand, for example so that your integration tests can reach `paid`, or after a support decision on an underpaid order. An order can be approved once, and not while a transfer for it is still confirming.

- You receive a normal signed `payment.paid` callback. Its `deposit` is `null`, because no transfer completed the order.
- `amountReceived` is the approved total. `manualApproval.amountOnChain` is what had actually arrived on-chain when the order was approved (it can be `0`); the approved total is never less than that.
- `deposits` lists only real transfers.
- Credit by the Crediting rule as usual: `amountReceived` minus what you already credited for that `orderId`. If you credited a partial amount on `payment.expired`, you credit only the difference.
- A transfer that arrives later for that order is reported on its own as `deposit.unmatched` with `order_already_paid`, and never changes `amountReceived`.
- `paidAt` is the time of the approval. An approved expired order keeps its `expiredAt`.

A manually approved order carries a `manualApproval` object; the key is absent on every other order. An expired fixed-amount order that had received 40 of 100.5 USDT, approved at the full amount (excerpt):

JSON Order (excerpt)


```
{
  "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
  "status": "paid",
  "amount": "100.5",
  "amountReceived": "100.5",
  "overpaid": false,
  "underpaid": false,
  "paidAt": "2026-11-11T11:02:40.000Z",
  "expiredAt": "2026-11-11T10:33:04.000Z",
  "manualApproval": {
    "approvedAt": "2026-11-11T11:02:40.000Z",
    "amount": "100.5",
    "amountOnChain": "40"
  },
  "deposits": [ { "depositId": "6650a1f2c3d4e5f6a7b8c9d0", "amount": "40", "…": "…" } ]
}
```


One address per user and chain

The first order for a user (`userId`) on a chain assigns that user a deposit address from MekaPay's pool. The address is **permanent**: every later order for the same user on the same chain uses the same address, and it is never given to anyone else. The same user has one BSC address and one TRON address.

Because the address stays the same, users may send money to it again later without creating an order first. Such transfers are still detected and reported to you as `deposit.unmatched` with `unmatchedReason: "no_active_order"`; see unmatched deposits.

One active order per address

An address can have only one active (`pending` or `confirming`) order at a time, so every transfer maps to exactly one order. When you create a new order for a user and chain that already has an active order:

- if no transfer has been detected for the active order, it is cancelled with `cancelReason: "superseded"` and the new order is created;
- otherwise the request fails with `409 ACTIVE_ORDER_IN_PROGRESS`, and the response includes the `orderId` of the order that is waiting for its transfer. Wait for that order to finish, or show it to the user again.

A retry of the same create request never replaces its own order: an active order with the same `merchantOrderId` is not superseded, even when two copies of the request arrive at the same moment. See Idempotency and retries.

Deposits

Every USDT transfer into a MekaPay deposit address becomes a *deposit* with its own status:

DEPOSIT STATUS	MEANING
`detected`	Seen on-chain, waiting for confirmations (`confirmations` of `requiredConfirmations`).
`confirmed`	Final. Only confirmed deposits count toward an order.
`orphaned`	The transfer dropped out of the chain (re-organisation) or could not be verified. Never counted.

DEPOSIT STATUS	MEANING
`failed`	The transaction failed on-chain. Never counted.

Once a deposit is confirmed, MekaPay matches it to an order: `matchStatus` is `pending` until then, and `matched` or `unmatched` afterwards.

Unmatched deposits

A confirmed deposit that cannot pay an order is `unmatched`. MekaPay still receives the funds, records the deposit and sends you a `deposit.unmatched` callback so you can decide what to do, for example credit the user manually after a support check.

UNMATCHEDREASON	WHY
`no_active_order`	No order for this address covered the transfer's block time: it was sent before any order, after the window closed, or long after.
`below_minimum`	Any-amount order, and the transfer was smaller than `minAmount`.
`after_expiry`	The transfer was sent inside the window of an order that has since expired, and it did not complete that order (see Late transfers).
`order_cancelled`	The order that covered the transfer was cancelled: the transfer was sent after the cancellation, or sent before it without completing the order.
`order_already_paid`	The order was already paid; this is an extra transfer, for example a second payment of the same amount.

The callback includes the related order when there is one (for example the expired or cancelled order), else `order` is `null`. Treat any reason you do not recognise as unmatched; new reasons may be added. Your agreed fee applies to every confirmed deposit, matched or unmatched.

API REFERENCE

Create an order

Creates a payment order for one of your users and returns the deposit address and the payment link.

`POST` `/api/gateway/v1/orders`

Request body

FIELD	RULES
`merchantOrderId` `string`	**required** Your id for this payment, unique within your account. 1–64 characters: letters, digits and `. _ : -` (`^[A-Za-z0-9._:\-]{1,64}$`).
`userId` `string`	**required** Your id for the paying user. 1–64 characters: letters, digits and `. _ : @ -` (`^[A-Za-z0-9._:@\-]{1,64}$`). The same `userId` always gets the same address on a chain. Ids starting with `staff-test-` are reserved for MekaPay test orders and are refused with `400 VALIDATION_ERROR`.
`mode` `string`	**required** `fixed` or `any`. See Payment modes.
`chain` `string`	**required** `bsc` (BNB Smart Chain, BEP20) or `tron` (TRON, TRC20).
`currency` `string`	**optional** Must be `USDT` if sent.
`amount` `string`	**fixed: required** Decimal string with at most 6 decimals, at least your account minimum for the chain (the platform floor of 1 USDT BSC / 5 USDT TRON, or the higher minimum agreed for your account), at most `1000000`. Must be absent or `null` in any mode.
`minAmount` `string | null`	**any: optional** Smallest transfer that pays the order. Defaults to your account minimum for the chain. A value you send only has to be at least the platform floor (1 USDT BSC, 5 USDT TRON), even if your account minimum is higher. Must be absent or `null` in fixed mode.
`expiresInMinutes` `integer`	**optional** 15–60. Defaults to your account setting (30).
`returnUrl` `string`	**optional** `https://` URL, at most 512 characters. The payment page offers a "Return to merchant" button to it once the order is paid, expired or cancelled.
`metadata` `object`	**optional** Any JSON object up to 2048 bytes when serialised. Returned unchanged in the order and in callbacks. It is never shown on the payment page.

Example request

JSON POST /api/gateway/v1/orders

```
{
  "merchantOrderId": "INV-10023",
  "userId": "player_123",
  "mode": "fixed",
  "chain": "bsc",
  "currency": "USDT",
  "amount": "100.50",
  "expiresInMinutes": 30,
  "returnUrl": "https://merchant.example/deposit/done",
  "metadata": { "cartId": "C-778" }
}
```

An any-amount order on TRON with a 10 USDT minimum:

JSON

```
{
  "merchantOrderId": "DEP-55120",
  "userId": "player_123",
  "mode": "any",
  "chain": "tron",
  "minAmount": "10"
}
```

Idempotency and retries

`merchantOrderId` is the idempotency key. If you send it again:

- with the same `userId`, `mode`, `chain`, `amount` and `minAmount`, you get `200` with the existing order and `"created": false`, whatever its status is now;
- with any of those fields different, you get `409 DUPLICATE_MERCHANT_ORDER_ID`.

This also holds when two copies of the same request arrive at the same moment (for example a client retry racing the original): one order is created, the other copy gets it back with `"created": false` (or `409 DUPLICATE_MERCHANT_ORDER_ID` if the details differ). A retry never supersedes its own order.

So after a timeout or a network error, retry the same request with the same `merchantOrderId`: you never create two orders for one payment. Use a new `merchantOrderId` for each new payment attempt.

Response

`201 Created` for a new order, `200 OK` with `"created": false` for an existing one.

JSON 201 Created

```
{
  "success": true,
  "created": true,
  "data": {
    "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
    "reference": "MP2611117K2QDM",
    "merchantOrderId": "INV-10023",
```

```
    "userId": "player_123",
    "mode": "fixed",
    "status": "pending",
    "chain": "bsc",
    "network": "BEP20",
    "currency": "USDT",
    "tokenContract": "0x55d398326f99059fF775485246999027B3197955",
    "decimals": 18,
    "amount": "100.5",
    "minAmount": null,
    "amountReceived": "0",
    "overpaid": false,
    "underpaid": false,
    "address": "0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed",
    "paymentUrl": "https://mekapay.online/pay/po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
    "requiredConfirmations": 15,
    "expiresAt": "2026-11-11T10:30:00.000Z",
    "createdAt": "2026-11-11T10:00:00.000Z",
    "paidAt": null,
    "expiredAt": null,
    "cancelledAt": null,
    "cancelReason": null,
    "returnUrl": "https://merchant.example/deposit/done",
    "metadata": { "cartId": "C-778" },
    "deposits": []
  }
}
```


The order object

The same object is returned by every order endpoint and sent in callbacks.

FIELD	DESCRIPTION
`orderId`	MekaPay's id: `po_` + 24 random letters and digits. Unguessable, because it appears in the public payment link.
`reference`	Short reference for people: `MP` + date (`yyMMdd` , UTC) + 6 characters, e.g. `MP2611117K2QDM` . Shown on the payment page; quote it to MekaPay support.
`merchantOrderId` , `userId`	Your values, as sent.
`test`	Present and `true` only on a test order that MekaPay staff created for your account with the console's Test Payment tool (absent on every order you create). Test orders send callbacks only when staff choose to; you can recognise them by this field and by their `merchantOrderId` starting with `TEST-` .
`mode`	`fixed` or `any` .
`status`	`pending` , `confirming` , `paid` , `expired` or `cancelled` . See Order lifecycle.
`chain` , `network` , `currency`	`bsc` / `BEP20` or `tron` / `TRC20` ; always `USDT` .
`tokenContract` , `decimals`	The USDT contract on that chain and its on-chain decimals (18 on BSC, 6 on TRON).
`amount`	Fixed mode: the amount due, as a normalised decimal string. Any mode: `null` .

FIELD	DESCRIPTION
`minAmount`	Any mode: the smallest transfer that pays the order. Fixed mode: `null`.
`amountReceived`	Sum of the confirmed transfers matched to this order, as a decimal string. On BSC it can have up to 18 decimals. On an order MekaPay staff approved by hand, the approved total (see `manualApproval`).
`overpaid`	Fixed mode: `true` when `amountReceived` is greater than `amount`.
`underpaid`	`true` when the order expired after receiving some, but not enough, confirmed funds.
`address`	The user's deposit address: EIP-55 checksummed `0x…` on BSC, Base58 `T…` on TRON.
`paymentUrl`	The hosted payment page: `https://mekapay.online/pay/<orderId>`.
`requiredConfirmations`	Confirmations a transfer needs: 15 on BSC, 19 on TRON. See Chains.
`createdAt`, `expiresAt`	ISO 8601 timestamps in UTC.
`paidAt`, `expiredAt`, `cancelledAt`	When the order reached that status, or `null`. An order paid by a late transfer keeps its `expiredAt` or `cancelledAt`.
`cancelReason`	`merchant`, `superseded`, `staff` or `null`.
`returnUrl`, `metadata`	Your values, or `null`.
`manualApproval`	Present only on an order MekaPay staff marked as paid by hand (absent on every other order): `approvedAt` (ISO 8601), `amount` (the approved total, also in `amountReceived`) and `amountOnChain` (what had actually arrived on-chain when it was approved). See Manual approval.
`deposits`	The transfers linked to this order (deposit objects), oldest first. Left out in list responses. Only real transfers, also on a manually approved order.

Errors

HTTP	CODE	WHEN
400	`VALIDATION_ERROR`	A field is missing or malformed. The response names it in `field`.
400	`INVALID_AMOUNT`	`amount` or `minAmount` is not a positive decimal with at most 6 decimals, or is above 1,000,000.
400	`AMOUNT_BELOW_MINIMUM`	Fixed mode: `amount` is below your account minimum. Any mode: `minAmount` is below the platform floor. The response names the `field` and the `minimum`.
409	`DUPLICATE_MERCHANT_ORDER_ID`	The `merchantOrderId` exists with different order details.
409	`ACTIVE_ORDER_IN_PROGRESS`	The user already has an active order on this chain with a detected transfer. The response includes its `orderId`.
503	`POOL_EXHAUSTED`	No deposit address is free for a new user on this chain. Retry later; MekaPay is alerted automatically.

JSON 400 Bad Request

```
{
  "success": false,
  "code": "VALIDATION_ERROR",
  "message": "amount is required when mode is fixed",
  "field": "amount"
}
```

JSON 409 Conflict

```
{
  "success": false,
  "code": "ACTIVE_ORDER_IN_PROGRESS",
  "message": "This user has an order waiting for its transfer to confirm",
  "orderId": "po_3hT9wQ2mZ7xK5cV1bN8rL4dF"
}
```

The authentication errors on Authentication & signing apply to every endpoint.

API REFERENCE

Retrieve / list orders

Read one order by MekaPay's id or by yours, or page through your orders.

Retrieve an order

`GET` `/api/gateway/v1/orders/:orderId`

Returns the order object, including its `deposits`. Answers `404 ORDER_NOT_FOUND` when the order does not exist or belongs to another merchant.

Find an order by your id

`GET` `/api/gateway/v1/orders?merchantOrderId=INV-10023`

Returns the order with that `merchantOrderId`, or `404 ORDER_NOT_FOUND`. Useful when a create request timed out and you are not sure whether it reached MekaPay (or simply retry the create; it is idempotent).

List orders

`GET` `/api/gateway/v1/orders`

QUERY PARAMETER	DESCRIPTION
`status`	`pending`, `confirming`, `paid`, `expired` or `cancelled`.
`chain`	`bsc` or `tron`.
`userId`	Only this user's orders.
`from`, `to`	Creation time range, ISO 8601, e.g. `2026-11-01T00:00:00Z`.
`page`	Page number, starting at 1. Default 1.
`limit`	Items per page, at most 100. Default 25.

Remember to sign the query string exactly as you send it, for example `/api/gateway/v1/orders?status=paid&chain=bsc&page=2`.

JSON 200 OK


```
{
  "success": true,
  "data": [
    {
      "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "reference": "MP2611117K2QDM",
      "merchantOrderId": "INV-10023",
```

```
      "userId": "player_123",
      "mode": "fixed",
      "status": "paid",
      "chain": "bsc",
      "amount": "100.5",
      "amountReceived": "100.5",
      "paidAt": "2026-11-11T10:05:02.000Z",
      "…": "every other order field except deposits"
    }
  ],
  "pagination": { "page": 1, "limit": 25, "total": 1, "pages": 1 }
}
```


Lists leave out `deposits`; fetch a single order to see them.

Polling

Callbacks tell you about every status change you need to act on. If you also poll, for example to show live progress in your own interface, poll a single order no more often than every 5–10 seconds, stop when it reaches a final status, and stay within the rate limit. Use the list endpoint for reconciliation, not for real-time updates.

API REFERENCE

Cancel an order

Stops an active order that has not received a transfer, for example when the user abandons checkout.

`POST` `/api/gateway/v1/orders/:orderId/cancel`

Request

No request body. Sign it with an empty `rawBody`, so the string to sign ends with a newline:

TEXT

```
1794391200
POST
/api/gateway/v1/orders/po_8Kd2Qx7mN4pR1sT6vW9yZ3bC/cancel
```

Rules

- Only an **active** order (`pending` or `confirming`) with **no detected or confirmed transfer** can be cancelled.
- Otherwise the request fails with `409 ORDER_NOT_CANCELLABLE`: the order is already final, or money is on its way. Let it finish.
- Cancellation does not send a callback; the response is your confirmation.
- You do not need to cancel before creating a new order for the same user and chain: the new order replaces an untouched active one automatically (`cancelReason: "superseded"`).

! **The address stays the user's.**
If the user pays after you cancelled, the transfer is still detected and reported as `deposit.unmatched` with `unmatchedReason: "order_cancelled"`. Tell users not to pay a cancelled order.

A transfer the user sent *before* you cancelled, which MekaPay had not seen yet, still counts: if it completes the order, the order becomes `paid` and you receive `payment.paid`; otherwise it is reported as `order_cancelled`. See Late transfers.

Response

The cancelled order object:

JSON 200 OK (abridged)

```
{
  "success": true,
  "data": {
    "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
    "status": "cancelled",
    "cancelledAt": "2026-11-11T10:07:41.000Z",
```

```
    "cancelReason": "merchant"
  }
}
```

HTTP	CODE	WHEN
404	`ORDER_NOT_FOUND`	No such order for your account.
409	`ORDER_NOT_CANCELLABLE`	The order is final, or a transfer has been detected for it.

API REFERENCE

List deposits

Every USDT transfer into your users' deposit addresses, matched to an order or not. Use it for reconciliation and to review unmatched deposits.

`GET` `/api/gateway/v1/deposits`

Query parameters

PARAMETER	DESCRIPTION
`status`	`detected`, `confirmed`, `orphaned` or `failed`. Default `confirmed`.
`matchStatus`	`pending`, `matched` or `unmatched`.
`userId`	Only this user's deposits.
`from`, `to`	Time range, ISO 8601.
`page`, `limit`	Page number from 1; at most 100 per page, default 25.

JSON GET /api/gateway/v1/deposits?matchStatus=unmatched


```
{
  "success": true,
  "data": [
    {
      "depositId": "8277e0910d750195b4487976",
      "chain": "tron",
      "txHash": "02a2fc5ce4e87fabe6d589e85f801cd7b8a296208a96cae9f3f89e0e6f28e565",
      "logPos": 0,
      "from": "TV311EvMxfXQzVbJFNbBvtyH1XjKp6qswU",
      "to": "TW28meJ4JPqGG8GeV9tTykaep8WQhvUNwk",
      "amount": "25",
      "status": "confirmed",
      "confirmations": 19,
      "requiredConfirmations": 19,
      "blockNumber": 76543210,
      "blockTimestamp": "2026-11-11T12:41:03.000Z",
      "detectedAt": "2026-11-11T12:41:09.000Z",
      "confirmedAt": "2026-11-11T12:42:06.000Z",
      "matchStatus": "unmatched",
      "unmatchedReason": "no_active_order",
      "orderId": null,
      "explorerUrl":
"https://tronscan.org/#/transaction/02a2fc5ce4e87fabe6d589e85f801cd7b8a296208a96cae9f3f89e0e6f28e565",
      "userId": "player_123"
    }
  ],
  "pagination": { "page": 1, "limit": 25, "total": 1, "pages": 1 }
}
```

The deposit object

FIELD	DESCRIPTION
`depositId`	MekaPay's id for the transfer.
`chain`	`bsc` or `tron`.
`txHash`	Transaction hash in lower case: `0x` + 64 hex on BSC, 64 hex without a prefix on TRON.
`logPos`	Position of the transfer inside its transaction. One transaction can pay several addresses, so (`chain`, `txHash`, `logPos`) identifies a deposit.
`from`, `to`	Sender address and the user's deposit address.
`amount`	USDT received, as a decimal string.
`status`	`detected`, `confirmed`, `orphaned` or `failed`. See Deposits.
`confirmations`, `requiredConfirmations`	Progress toward finality (15 on BSC, 19 on TRON).
`blockNumber`, `blockTimestamp`	Block that contains the transfer and its time. The block time decides which order the transfer belongs to.
`detectedAt`, `confirmedAt`	When MekaPay saw the transfer and when it became final.
`matchStatus`, `unmatchedReason`	Whether the deposit paid an order; see unmatched deposits.
`orderId`	The order it is linked to, or `null`.
`explorerUrl`	Link to the transaction on BscScan or Tronscan.
`userId`	The user who owns the address. Included in this list and in `deposit.unmatched` callbacks.

Reconciliation

- Once a day, list the previous day's `confirmed` deposits and check that each matched one is credited in your system exactly once.
- List `matchStatus=unmatched` to find money that arrived without a payable order, and resolve each case with the user.
- The merchant portal shows the same data under **Deposits**, with an unmatched filter.

Balance

What MekaPay holds for you on each chain: your confirmed deposits minus the deposit fee, minus your settlements. You withdraw it with a settlement.

`GET` `/api/gateway/v1/balance`

How the balance is kept

- **One balance per chain.** USDT on BNB Smart Chain (BEP20) and on TRON (TRC20) are kept apart and never converted into each other.
- **Every confirmed deposit is credited**, matched to an order or not (an unmatched deposit is still your users' money), minus your deposit fee. The fee is your fee percentage of the amount, rounded *down* to the token's smallest unit (BEP20: 18 decimals, TRC20: 6).
- An order MekaPay staff approved by hand adds nothing to the balance: only money that arrived on-chain is credited.
- A settlement request is deducted from the balance at once and shows in `pendingSettlements` until it is completed. A cancelled or rejected settlement is credited back in full.
- The ledger behind the balance is append-only and every entry is chained to the previous one by a hash, so a changed or missing entry is detected. MekaPay verifies every ledger daily and after every settlement.

Request

No query parameters. Sign it like any `GET` (empty body).

Response

JSON 200 OK


```
{
  "success": true,
  "data": {
    "settlementsEnabled": true,
    "balances": [
      {
        "chain": "bsc",
        "network": "BEP20",
        "currency": "USDT",
        "balance": "1520.75",
        "pendingSettlements": "200",
        "settlementFee": "1",
        "settlementMinimum": "10"
      },
      {
        "chain": "tron",
        "network": "TRC20",
        "currency": "USDT",
        "balance": "8030.4",
        "pendingSettlements": "0",
```

```
        "settlementFee": "3",
        "settlementMinimum": "10"
      }
    ]
  }
}
```


FIELD	MEANING
`settlementsEnabled` `boolean`	Whether you can request settlements. Off until MekaPay turns it on for your account.
`balance` `string`	What you can settle now, in USDT. Settlements in progress are already deducted.
`pendingSettlements` `string`	The total of your settlements that are requested, approved or being verified.
`settlementFee` `string`	The fixed fee per settlement on this chain, deducted from the settled amount.
`settlementMinimum` `string`	The smallest settlement on this chain.

All amounts are decimal strings. The same balance is shown in the merchant portal under **Balance & Settlements**.

BALANCES & SETTLEMENTS

Settlement addresses

Your own wallets that settlements are paid to. You add them in the merchant portal with your two-factor code, MekaPay approves them, and they can receive settlements 48 hours after approval.

`GET` `/api/gateway/v1/settlement-addresses`

Adding an address

1. Turn on two-factor authentication for the merchant portal (**Security**).
2. In the portal, open **Balance & Settlements → Settlement addresses** and add the chain, the address, an optional label (up to 60 characters) and the current code from your authenticator app.
3. MekaPay reviews the address and approves or rejects it. Approval needs a MekaPay staff member's own two-factor code.
4. The address is `cooling` for **48 hours** after approval, then `active`: only active addresses can receive settlements.

> **Why addresses are locked.**
> Addresses cannot be added through the API, and MekaPay staff cannot add them for you. Even if your API credentials leaked, money could only ever be sent to wallets you added yourself with your two-factor code, and every new wallet waits 48 hours after approval, so there is time to notice and remove one you did not add.

Rules

- The address must be valid for the chain: a `0x…` BNB Smart Chain address (lowercase or with a correct EIP-55 checksum) or a TRON `T…` address. It is stored in its checksummed / Base58 form.
- It must be a wallet you control: MekaPay deposit addresses, the zero address and the USDT token contract are refused.
- At most **3 addresses per chain** that are pending approval or approved (cooling or active). Rejected and removed addresses do not count, and the same address cannot be added twice while it is pending or approved.
- You can remove an address at any time in the portal, with your two-factor code. MekaPay can also remove one, with a reason. Removing an address does not change settlements you already requested to it.

States

STATE	MEANING
`pending_approval`	Added, waiting for MekaPay's review.
`cooling`	Approved; usable from `activatesAt`, 48 hours after approval.
`active`	Can receive settlements.

STATE	MEANING
`rejected`	Not approved; see `rejectReason`. Remove it or add another address.
`removed`	Removed by you or by MekaPay; listed only when you ask for it.

`status` is the stored status (`pending_approval`, `approved`, `rejected` or `removed`); `state` splits `approved` into `cooling` and `active`. Use `state`.

List your addresses

QUERY PARAMETER	DESCRIPTION
`chain`	`bsc` or `tron`.
`status`	A state from the table above, `approved` (cooling or active), or `all`. Without it, every address except removed ones.

JSON 200 OK


```
{
  "success": true,
  "data": [
    {
      "id": "6731f0c2a9e4b1d2c3f40512",
      "chain": "tron",
      "network": "TRC20",
      "currency": "USDT",
      "address": "TPRgcsJLsCUpYcCwiiXxxQYM4rof95eDVZ",
      "label": "Main TRON wallet",
      "status": "approved",
      "state": "active",
      "requestedAt": "2026-11-08T09:12:40.000Z",
      "approvedAt": "2026-11-08T11:30:02.000Z",
      "activatesAt": "2026-11-10T11:30:02.000Z",
      "rejectedAt": null,
      "rejectReason": null,
      "removedAt": null,
      "removedBy": null
    }
  ]
}
```


Use `id` as `settlementAddressId` when you request a settlement.

Settlements

Withdraw your balance to one of your active settlement addresses. MekaPay reviews every settlement, sends the USDT and confirms the transfer on-chain before it is completed.

Fees and limits

CHAIN	FEE PER SETTLEMENT	MINIMUM SETTLEMENT
BNB Smart Chain (BEP20)	1 USDT	10 USDT
TRON (TRC20)	3 USDT	10 USDT

- The fee is **deducted from the settled amount**: you request `amount`, it is taken from your balance, and `netAmount = amount - fee` arrives at your address. A 100 USDT settlement on TRON sends 97 USDT.
- `amount` is a decimal string with at most 6 decimals and at most your balance on that chain.
- Settlements must be turned on for your account by MekaPay; until then every request is refused with `403 SETTLEMENTS_DISABLED` (`GET /balance` shows `settlementsEnabled`).
- The destination must be one of your settlement addresses in state `active`.

Request a settlement

`POST` `/api/gateway/v1/settlements`

FIELD	RULES
`chain` `string`	**required** `bsc` or `tron`.
`amount` `string`	**required** Decimal string, at most 6 decimals, at least the chain's minimum and more than its fee.
`settlementAddressId` `string`	**required** The `id` of one of your active settlement addresses on the same chain.
`merchantSettlementId` `string`	**optional** Your id for this settlement, unique within your account, 1–64 characters of letters, digits and `. _ : -`. Makes retries safe (see below).

`JSON` `POST /api/gateway/v1/settlements`

```
{
  "chain": "tron",
  "amount": "100",
  "settlementAddressId": "6731f0c2a9e4b1d2c3f40512",
  "merchantSettlementId": "WD-20261111-001"
}
```

JSON 201 Created


```
{
  "success": true,
  "created": true,
  "data": {
    "settlementId": "st_4Gm8Qx2Lr7Vt1Zp9Kc3Wd6Yb",
    "reference": "MPS261111K7QD2M",
    "merchantSettlementId": "WD-20261111-001",
    "chain": "tron",
    "network": "TRC20",
    "currency": "USDT",
    "amount": "100",
    "fee": "3",
    "netAmount": "97",
    "address": "TPRgcsJLsCUpYcCwiiXxxQYM4rof95eDVZ",
    "addressLabel": "Main TRON wallet",
    "settlementAddressId": "6731f0c2a9e4b1d2c3f40512",
    "status": "requested",
    "requestedVia": "api",
    "createdAt": "2026-11-11T10:20:31.000Z",
    "approvedAt": null,
    "completedAt": null,
    "rejectedAt": null,
    "rejectReason": null,
    "cancelledAt": null,
    "txHash": null,
    "explorerUrl": null
  }
}
```


The amount leaves your balance immediately. If your balance is too low the request is refused with `409 INSUFFICIENT_BALANCE`, whose `available` field (and message) says how much you can settle; nothing is created.

Idempotency and retries

Send a `merchantSettlementId` and retry freely: the same id with the same `chain`, `amount` and `settlementAddressId` answers `200` with the existing settlement and `"created": false`; the same id with other details answers `409 DUPLICATE_MERCHANT_SETTLEMENT_ID`. Find a settlement by your id with `GET /settlements?merchantSettlementId=…`.

Statuses

STATUS	MEANING
`requested`	Received and deducted from your balance; waiting for MekaPay's approval. You can still cancel it.
`approved`	Approved by MekaPay; the transfer is being prepared.
`verifying`	The transfer has been sent and MekaPay is confirming it on-chain.
`completed`	Final: `netAmount` arrived at your address in `txHash`. You receive `settlement.completed`.
`rejected`	Final: MekaPay rejected it (`rejectReason`); the amount is back on your balance. You receive `settlement.rejected`.
`cancelled`	Final: you cancelled it; the amount is back on your balance.

A transfer is completed only after MekaPay has read it from the chain itself: the transaction succeeded, it is final (BNB Smart Chain: in a finalized block; TRON: in a solidified block), and it pays exactly `netAmount` of USDT to your settlement address. `txHash` and `explorerUrl` are filled in once the settlement is `completed`.

Cancel a settlement

`POST` `/api/gateway/v1/settlements/:settlementId/cancel`

Only while the settlement is `requested`. Optional body `{ "reason": "…" }` (up to 200 characters). Answers the cancelled settlement; the amount is credited back to your balance at once. Afterwards it fails with `409 SETTLEMENT_NOT_CANCELLABLE`. Cancelling does not send a callback.

Retrieve and list settlements

`GET` `/api/gateway/v1/settlements/:settlementId`

`GET` `/api/gateway/v1/settlements`

QUERY PARAMETER	DESCRIPTION
`status`	`requested`, `approved`, `verifying`, `completed`, `rejected` or `cancelled`.
`chain`	`bsc` or `tron`.
`merchantSettlementId`	Your id.
`from`, `to`	Creation time range, ISO 8601.
`page`, `limit`	As for orders: page from 1, at most 100 per page (default 25).

Lists are newest first and come with `pagination` (`page`, `limit`, `total`, `pages`).

The settlement object

FIELD	MEANING
`settlementId`	MekaPay's id, `st_` followed by 24 characters.
`reference`	Short reference for support, e.g. `MPS261111K7QD2M`.
`merchantSettlementId`	Your id, or `null`.
`amount`, `fee`, `netAmount`	Deducted from your balance, the fixed fee, and what is sent to you (`amount - fee`).
`address`, `addressLabel`, `settlementAddressId`	The destination as it was when you requested the settlement.
`status`	See Statuses.
`requestedVia`	`api` or `portal`.

FIELD	MEANING
`createdAt`, `approvedAt`, `completedAt`, `rejectedAt`, `cancelledAt`	ISO 8601 times, `null` until they happen.
`rejectReason`	Why MekaPay rejected it, on a rejected settlement.
`txHash`, `explorerUrl`	The verified transfer, on a completed settlement.

Errors

HTTP	CODE	WHEN
400	`VALIDATION_ERROR`	`field` is `chain`, `amount` (format, below the minimum or not above the fee), `settlementAddressId` (unknown, another chain, or not active: the message says pending, cooling until when, rejected or removed) or `merchantSettlementId`.
403	`SETTLEMENTS_DISABLED`	Settlements are not turned on for your account.
404	`SETTLEMENT_NOT_FOUND`	No such settlement for your account.
409	`INSUFFICIENT_BALANCE`	The amount is more than your balance; `available` says how much you can settle.
409	`DUPLICATE_MERCHANT_SETTLEMENT_ID`	The `merchantSettlementId` exists with other details (`settlementId` names it).
409	`SETTLEMENT_NOT_CANCELLABLE`	The settlement is no longer `requested`.

In the merchant portal

Balance & Settlements shows your balance per chain, lets you request a settlement (it shows the fee and the net amount before you confirm), lists your settlements, and manages your settlement addresses. Every action there (adding or removing an address, requesting or cancelling a settlement) asks for your current two-factor code, so turn on two-factor authentication first.

INTEGRATION

Payment page

Every order has a hosted payment page at its `paymentUrl`. Send the user there and MekaPay guides them through the transfer.

The payment link

TEXT

```
https://mekapay.online/pay/<orderId>
```

Redirect the user to it, open it in a new tab, or load it in your app's in-app browser. The page works on phones and desktops, and needs no login. Use the link exactly as returned in `paymentUrl`.

What the user sees

- Your business name ("Pay Example Merchant") and the MekaPay brand.
- The amount to pay (fixed mode, with a copy button) or "Any amount · minimum X USDT" (any mode).
- The network, for example **USDT · BNB Smart Chain (BEP20)**, with a clear warning to send only USDT on that network.
- A QR code of the deposit address and the address with a copy button.
- A live countdown to `expiresAt`.
- A progress tracker: *Waiting for payment* → *Confirming n/N* → *Paid*, with the detected transaction and a link to the block explorer.
- The order reference (`MP…`) and help text: send the exact amount, network fees are paid by the sender, exchanges that deduct fees from the amount can make a payment fall short.

The page is available in English and Simplified Chinese. It follows the browser's language and has a language switch. It refreshes the order every few seconds while it is waiting.

Page states

ORDER STATUS	THE PAGE SHOWS
`pending`	Payment instructions, QR code and countdown. In fixed mode after a partial payment, the remaining amount.
`confirming`	The detected transfer and its confirmation count.
`paid`	A success screen with the amount received, a note if the order was overpaid, and "Return to merchant" when you set `returnUrl`.
`expired`	The order has expired. If part of the amount arrived, the user is asked to contact you with the order reference.
`cancelled`	The order was cancelled and must not be paid.
unknown id	"Payment not found".

Returning to your site

When the order is paid, expired or cancelled, the page offers a "Return to merchant" button linking to your `returnUrl`. The return itself proves nothing: users can open that URL directly. Credit only on the signed `payment.paid` callback, or after reading the order with your API credentials.

Showing the payment in your own interface

You can skip the hosted page and show `address`, `amount` (or `minAmount`), `network` and `expiresAt` yourself. If you do:

- name the network and token plainly ("USDT on BNB Smart Chain (BEP20)" or "USDT on TRON (TRC20)") and warn that other tokens and networks are not credited;
- encode only the plain address in a QR code, not a payment URI, because wallets interpret URIs differently;
- show a countdown to `expiresAt` and the reference;
- update the status from callbacks or by reading the order, never from your own chain watching.

Privacy and security

- The link is public but unguessable: the `orderId` is 24 random characters. Anyone with the link sees the amount, the address, the status and your business name.
- The page never shows your `userId`, `merchantOrderId` or `metadata`.
- The page is not indexed by search engines and loads no third-party scripts or trackers.

INTEGRATION

Callbacks

MekaPay sends a signed HTTPS `POST` to your callback URL when an order is paid, when it expires, when money arrives that does not pay an order, and when a settlement is completed or rejected.

Your callback URL

- Set it in the merchant portal under **Integration**, or ask MekaPay to set it for you.
- It must be an `https://` URL with a valid certificate, reachable from the internet. Addresses that resolve to private, loopback or link-local networks are refused.
- Redirects are not followed: a `3xx` answer counts as a failure.
- Events that happen while no URL is set are recorded as failed deliveries (`NO_CALLBACK_URL`). Resend them from the portal once the URL is in place.

Events

EVENT	SENT WHEN	`DATA`
`payment.paid`	An order becomes `paid`, including a late transfer that completes an expired or cancelled order, and a manual approval by MekaPay staff.	`order` and the `deposit` that completed it (`null` for a manual approval)
`payment.expired`	An order becomes `expired`: at the earliest 3 minutes after `expiresAt`, a little later if MekaPay's chain scanner is catching up.	`order`; `deposit` is `null`
`deposit.unmatched`	A confirmed deposit could not pay an order: no order covered it, it was below `minAmount`, it did not complete an expired or cancelled order, or the order was already paid (`order_already_paid`).	`deposit` (with `userId` and `unmatchedReason`), and the related `order` if there is one, else `null`
`settlement.completed`	A settlement was verified on-chain: `netAmount` arrived at your address.	`settlement` (the settlement object, with `txHash`)
`settlement.rejected`	MekaPay rejected a settlement; the amount is back on your balance.	`settlement` (with `rejectReason`)

Ignore events you do not know. MekaPay may add new event types. Answer `2xx` to an event your code does not handle and do nothing else; do not fail on an unknown `event` name or on extra fields in `data`.

Order creation and cancellation do not send callbacks; your API call already has the answer. Every confirmed deposit is reported: in `order.deposits` of a `payment.paid` or `payment.expired`, or as the `deposit` of its own `deposit.unmatched`.

Crediting rule

`amountReceived` is the order's running total of confirmed, matched transfers. Keep, per `orderId`, the amount you have already credited, and on every `payment.paid` (and on a `payment.expired` you decide to honour) credit `amountReceived` minus that amount. This way a partial amount you credited on `payment.expired` is never credited again when a late `payment.paid` arrives for the same order, and a repeated callback credits nothing. Credit each `deposit.unmatched` by its `depositId`, at most once. The rule is the same for an order MekaPay staff approved by hand: its `amountReceived` is the approved total.

The request

HEADER	VALUE
`Content-Type`	`application/json`
`User-Agent`	`MekaPay-Gateway/1.0`
`X-MP-Event`	The event name, e.g. `payment.paid`.
`X-MP-Event-Id`	The event id (`evt_…`), the same as `eventId` in the body.
`X-MP-Delivery-Attempt`	1 for the first attempt, then 2, 3, …
`X-MP-Timestamp`	Unix time in seconds when this attempt was signed.
`X-MP-Signature`	Lowercase hex `HMAC_SHA256(apiSecret, timestamp + "." + rawBody)`.

Payloads

Every callback has the same envelope. `order` is the order object, `deposit` the deposit object and `settlement` the settlement object returned by the API.

PAYMENT.PAID

JSON payment.paid


```
{
  "eventId": "evt_3NpXa9Lm2Qr7Tz5Vb8Kd1Wy4",
  "event": "payment.paid",
  "createdAt": "2026-11-11T10:05:12.000Z",
  "merchantCode": "example-merchant",
  "data": {
    "order": {
      "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "reference": "MP2611117K2QDM",
      "merchantOrderId": "INV-10023",
      "userId": "player_123",
      "mode": "fixed",
      "status": "paid",
      "chain": "bsc",
      "network": "BEP20",
      "currency": "USDT",
      "tokenContract": "0x55d398326f99059fF775485246999027B3197955",
      "decimals": 18,
      "amount": "100.5",
      "minAmount": null,
      "amountReceived": "100.5",
      "overpaid": false,
```

```
      "underpaid": false,
      "address": "0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed",
      "paymentUrl": "https://mekapay.online/pay/po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "requiredConfirmations": 15,
      "expiresAt": "2026-11-11T10:30:00.000Z",
      "createdAt": "2026-11-11T10:00:00.000Z",
      "paidAt": "2026-11-11T10:05:02.000Z",
      "expiredAt": null,
      "cancelledAt": null,
      "cancelReason": null,
      "returnUrl": "https://merchant.example/deposit/done",
      "metadata": { "cartId": "C-778" },
      "deposits": [ { "depositId": "6650a1f2c3d4e5f6a7b8c9d0", "…": "same as data.deposit" } ]
    },
    "deposit": {
      "depositId": "6650a1f2c3d4e5f6a7b8c9d0",
      "chain": "bsc",
      "txHash": "0xf37603c2d85a42bb4f4f9f1f152732627539f28e7d2e6dc486db3be949dda686",
      "logPos": 0,
      "from": "0xfB6916095ca1df60bB79Ce92cE3Ea74c37c5d359",
      "to": "0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed",
      "amount": "100.5",
      "status": "confirmed",
      "confirmations": 15,
      "requiredConfirmations": 15,
      "blockNumber": 39000000,
      "blockTimestamp": "2026-11-11T10:04:41.000Z",
      "detectedAt": "2026-11-11T10:04:44.000Z",
      "confirmedAt": "2026-11-11T10:04:58.000Z",
      "matchStatus": "matched",
      "unmatchedReason": null,
      "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "explorerUrl": "https://bscscan.com/tx/0xf37603c2d85a42bb4f4f9f1f152732627539f28e7d2e6dc486db3be949dda686"
    }
  }
}
```


PAYMENT.EXPIRED

JSON payment.expired


```
{
  "eventId": "evt_7Qm2Lx9Pa4Rt8Vb1Nc6Kd3Wz",
  "event": "payment.expired",
  "createdAt": "2026-11-11T10:33:05.000Z",
  "merchantCode": "example-merchant",
  "data": {
    "order": {
      "orderId": "po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "reference": "MP2611117K2QDM",
      "merchantOrderId": "INV-10023",
      "userId": "player_123",
      "mode": "fixed",
      "status": "expired",
      "chain": "bsc",
      "network": "BEP20",
      "currency": "USDT",
      "tokenContract": "0x55d398326f99059fF775485246999027B3197955",
      "decimals": 18,
      "amount": "100.5",
      "minAmount": null,
      "amountReceived": "0",
      "overpaid": false,
      "underpaid": false,
```

```
      "address": "0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed",
      "paymentUrl": "https://mekapay.online/pay/po_8Kd2Qx7mN4pR1sT6vW9yZ3bC",
      "requiredConfirmations": 15,
      "expiresAt": "2026-11-11T10:30:00.000Z",
      "createdAt": "2026-11-11T10:00:00.000Z",
      "paidAt": null,
      "expiredAt": "2026-11-11T10:33:04.000Z",
      "cancelledAt": null,
      "cancelReason": null,
      "returnUrl": "https://merchant.example/deposit/done",
      "metadata": { "cartId": "C-778" },
      "deposits": []
    },
    "deposit": null
  }
}
```


If part of a fixed amount arrived before the order expired, `amountReceived` shows it, `underpaid` is `true` and `deposits` lists the transfers.

DEPOSIT.UNMATCHED

JSON deposit.unmatched


```
{
  "eventId": "evt_9Xc4Vb7Nm2Lk5Jh8Gf1Ds3Aq",
  "event": "deposit.unmatched",
  "createdAt": "2026-11-11T12:42:09.000Z",
  "merchantCode": "example-merchant",
  "data": {
    "deposit": {
      "depositId": "8277e0910d750195b4487976",
      "chain": "tron",
      "txHash": "02a2fc5ce4e87fabe6d589e85f801cd7b8a296208a96cae9f3f89e0e6f28e565",
      "logPos": 0,
      "from": "TV311EvMxfXQzVbJFNbBvtyH1XjKp6qswU",
      "to": "TW28meJ4JPqGG8GeV9tTykaep8WQhvUNwk",
      "amount": "25",
      "status": "confirmed",
      "confirmations": 19,
      "requiredConfirmations": 19,
      "blockNumber": 76543210,
      "blockTimestamp": "2026-11-11T12:41:03.000Z",
      "detectedAt": "2026-11-11T12:41:09.000Z",
      "confirmedAt": "2026-11-11T12:42:06.000Z",
      "matchStatus": "unmatched",
      "unmatchedReason": "no_active_order",
      "orderId": null,
      "explorerUrl":
"https://tronscan.org/#/transaction/02a2fc5ce4e87fabe6d589e85f801cd7b8a296208a96cae9f3f89e0e6f28e565",
      "userId": "player_123"
    },
    "order": null
  }
}
```


SETTLEMENT.COMPLETED

JSON settlement.completed


```
{
  "eventId": "evt_9Qd2Lm7Xa4Tz1Vb8Kp3Wr6Yc",
```

```
  "event": "settlement.completed",
  "createdAt": "2026-11-11T13:02:44.000Z",
  "merchantCode": "example-merchant",
  "data": {
    "settlement": {
      "settlementId": "st_4Gm8Qx2Lr7Vt1Zp9Kc3Wd6Yb",
      "reference": "MPS261111K7QD2M",
      "merchantSettlementId": "WD-20261111-001",
      "chain": "tron",
      "network": "TRC20",
      "currency": "USDT",
      "amount": "100",
      "fee": "3",
      "netAmount": "97",
      "address": "TPRgcsJLsCUpYcCwiiXxxQYM4rof95eDVZ",
      "addressLabel": "Main TRON wallet",
      "settlementAddressId": "6731f0c2a9e4b1d2c3f40512",
      "status": "completed",
      "requestedVia": "api",
      "createdAt": "2026-11-11T10:20:31.000Z",
      "approvedAt": "2026-11-11T12:41:10.000Z",
      "completedAt": "2026-11-11T13:02:44.000Z",
      "rejectedAt": null,
      "rejectReason": null,
      "cancelledAt": null,
      "txHash": "5366db73b35357876ba59c83659e93b8ddbc5145be8cd37ddfcff4fe8b256243",
      "explorerUrl":
"https://tronscan.org/#/transaction/5366db73b35357876ba59c83659e93b8ddbc5145be8cd37ddfcff4fe8b256243"
    }
  }
}
```


SETTLEMENT.REJECTED

JSON settlement.rejected


```
{
  "eventId": "evt_2Kc7Vb4Lm9Qa1Tz6Xd3Wp8Yr",
  "event": "settlement.rejected",
  "createdAt": "2026-11-11T12:15:09.000Z",
  "merchantCode": "example-merchant",
  "data": {
    "settlement": {
      "settlementId": "st_7Hn3Wr8Kd1Qx5Vb2Lm9Tc4Za",
      "reference": "MPS261111P3XR8N",
      "merchantSettlementId": "WD-20261111-002",
      "chain": "bsc",
      "network": "BEP20",
      "currency": "USDT",
      "amount": "250",
      "fee": "1",
      "netAmount": "249",
      "address": "0x5A1fD3b7C4e2908F6d1Ab3c9E8F70456B2d1C3E9",
      "addressLabel": "Treasury",
      "settlementAddressId": "6731f0c2a9e4b1d2c3f40513",
      "status": "rejected",
      "requestedVia": "portal",
      "createdAt": "2026-11-11T11:58:20.000Z",
      "approvedAt": null,
      "completedAt": null,
      "rejectedAt": "2026-11-11T12:15:09.000Z",
      "rejectReason": "Please confirm this request with your account manager first.",
      "cancelledAt": null,
      "txHash": null,
```

```
      "explorerUrl": null
    }
  }
}
```

Verifying the signature

Verify every callback before you act on it:

1. Read the **raw request body** as bytes, before any JSON parsing. Re-serialised JSON does not produce the same bytes.
2. Check that `X-MP-Timestamp` is an integer within 300 seconds of your clock. Each attempt is signed when it is sent, so retries carry fresh timestamps.
3. Compute `HMAC_SHA256(apiSecret, timestamp + "." + rawBody)` as lowercase hex, with your whole API Secret string as the key.
4. Compare it with `X-MP-Signature` using a constant-time comparison (`crypto.timingSafeEqual`, `hash_equals`, `hmac.compare_digest`).
5. Reject the request with `401` if any check fails.

JAVASCRIPT Node.js

```javascript
import crypto from 'node:crypto';

function isValidMekapayCallback(apiSecret, rawBody, timestamp, signature) {
  if (!/^\d{1,12}$/.test(timestamp || '') || !/^[0-9a-f]{64}$/i.test(signature || '')) return false;
  if (Math.abs(Date.now() / 1000 - Number(timestamp)) > 300) return false;
  const expected = crypto.createHmac('sha256', apiSecret).update(`${timestamp}.${rawBody}`, 'utf8').digest();
  return crypto.timingSafeEqual(expected, Buffer.from(signature, 'hex'));
}
```

Complete receivers in Node.js, PHP and Python are in Code samples.

Worked example

With the test secret from Authentication, this (shortened) body and timestamp must give the signature below.

API Secret	`mps_AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8`
X-MP-Timestamp	`1794391512`
rawBody	`{"eventId":"evt_3NpXa9Lm2Qr7Tz5Vb8Kd1Wy4","event":"payment.paid","createdAt":"2026-11-11T10:05:12.000Z","merchantCode":"example-merchant","data":{"order":{"orderId":"po_8Kd2Qx7mN4pR1sT6vW9yZ3bC","status":"paid"},"deposit":null}}`
Signed payload	`1794391512.{"eventId":"evt_3NpXa9Lm2Qr7Tz5Vb8Kd1Wy4","event":"payment.paid", … }`
X-MP-Signature	`c10c65cc2f03d52438d090c3823914dcc18df2f2886f3af11368376d27ad775f`

Responding and retries

Answer with any `2xx` status within **10 seconds**. Store the event and do slow work afterwards. Anything else (a timeout, a connection error, a `3xx`, `4xx` or `5xx`) is a failed attempt, and MekaPay retries on this schedule:

ATTEMPT	SENT
1	As soon as the event happens (within a few seconds)
2	60 seconds after attempt 1 failed
3	5 minutes later
4	15 minutes later
5	1 hour later
6	3 hours later
7	6 hours later
8	12 hours later
9	24 hours later; if this fails too, the delivery is marked `failed`

Retries cover about 46 hours. A failed delivery can be resent at any time from the portal (**Integration → Recent callbacks**) or by MekaPay support. A resend carries the same `eventId` and payload. A resend is refused with `409 RESEND_PENDING` while a delivery of the same event is still queued or retrying, and with `429 RESEND_LIMIT` while 50 resent callbacks of your account are still waiting to go out.

Slow endpoints are deferred. MekaPay sends at most 2 callbacks to one merchant at a time. After 3 timeouts or connection errors in a row, your pending callbacks are held back for 2 minutes, then 4, 8 and so on up to 30 minutes, until your endpoint answers `2xx` again. Held-back time does not use up retry attempts, but events reach you later. Answer quickly and do slow work after responding.

Idempotency and ordering

- **Deduplicate on `eventId`.** The same event can reach you more than once (a retry after your response was lost, or a manual resend). Store processed event ids with a unique constraint and answer `200` to repeats without acting again.
- **Credit at most once per `orderId`** on `payment.paid`. Make the credit and the record of it one database transaction.
- **Do not rely on order of arrival.** In particular, `payment.paid` can arrive after `payment.expired` for the same order when a transfer sent in time confirms late, and for an order you cancelled when the user had already paid. Credit it by the Crediting rule.
- **Use the payload, or re-read the order.** The payload is the order as it was when the event happened. For extra assurance, read it again with `GET /orders/:orderId` before crediting.

INTEGRATION

Errors

Errors use standard HTTP status codes and a stable machine-readable `code`.

Error envelope

JSON

```
{
  "success": false,
  "code": "AMOUNT_BELOW_MINIMUM",
  "message": "amount must be at least 5 USDT on tron"
}
```

Some errors add fields: `VALIDATION_ERROR` includes `field`, the request field at fault; `AMOUNT_BELOW_MINIMUM` includes `field` and `minimum`; `ACTIVE_ORDER_IN_PROGRESS` includes `orderId`, the order that blocks the request; `IP_WHITELIST_REQUIRED` and `IP_NOT_WHITELISTED` include `ip`, the address MekaPay received the request from; `INSUFFICIENT_BALANCE` includes `available`, the balance you can settle. Write your code against `code`; `message` is meant for logs and people and may change.

JSON 403 Forbidden

```
{
  "success": false,
  "code": "IP_NOT_WHITELISTED",
  "message": "Requests from 203.0.113.7 are not allowed for this merchant.",
  "ip": "203.0.113.7"
}
```

Error codes

HTTP	CODE	MEANING	WHAT TO DO
400	`VALIDATION_ERROR`	A field is missing or malformed (see `field`).	Fix the request.
400	`INVALID_AMOUNT`	Not a positive decimal string with at most 6 decimals, or above 1,000,000.	Send e.g. `"100.50"`.
400	`AMOUNT_BELOW_MINIMUM`	`amount` below your account minimum, or `minAmount` below the platform floor.	Raise it to at least `minimum` from the response.
401	`MISSING_AUTH_HEADERS`	An `X-MP-*` header is missing.	Send all three headers.
401	`INVALID_CLIENT_KEY`	Unknown, revoked or rotated client key.	Check the key; ask MekaPay if it was rotated.
401	`TIMESTAMP_OUT_OF_RANGE`	Timestamp malformed or more than 300 s off.	Use Unix seconds; sync your clock.

HTTP	CODE	MEANING	WHAT TO DO
401	`INVALID_SIGNATURE`	The signature does not match.	See signing mistakes.
415	`UNSUPPORTED_MEDIA_TYPE`	A request body was not sent as JSON.	Send it with `Content-Type: application/json`.
403	`GATEWAY_DISABLED`	The gateway is not enabled for your account.	Contact MekaPay.
403	`PROJECT_SUSPENDED`	Your merchant account is not active.	Contact MekaPay.
403	`IP_WHITELIST_REQUIRED`	No server IP is whitelisted for your account yet (see `ip`).	Send `ip` to MekaPay support to have it whitelisted.
403	`IP_NOT_WHITELISTED`	The request came from an IP that is not on your whitelist (see `ip`).	Call from a whitelisted server, or ask MekaPay support to add `ip`. See IP whitelist.
404	`ORDER_NOT_FOUND`	No such order for your account.	Check the id.
409	`DUPLICATE_MERCHANT_ORDER_ID`	The `merchantOrderId` exists with other details.	Use a new id for a new payment.
409	`ACTIVE_ORDER_IN_PROGRESS`	The user's active order on this chain has a transfer on its way.	Wait for `orderId` to finish.
409	`ORDER_NOT_CANCELLABLE`	The order is final or has a detected transfer.	Let it finish.
403	`SETTLEMENTS_DISABLED`	Settlements are not turned on for your account.	Contact MekaPay.
404	`SETTLEMENT_NOT_FOUND`	No such settlement for your account.	Check the id.
409	`INSUFFICIENT_BALANCE`	A settlement is larger than your balance (see `available`).	Request at most `available`.
409	`DUPLICATE_MERCHANT_SETTLEMENT_ID`	The `merchantSettlementId` exists with other details.	Use a new id for a new settlement.
409	`SETTLEMENT_NOT_CANCELLABLE`	The settlement is no longer `requested`.	Nothing to do; it is being processed.
429	`RATE_LIMITED`	More than 300 requests per minute.	Back off and retry.
503	`POOL_EXHAUSTED`	No free deposit address for a new user on this chain.	Retry later.

Retrying safely

- **4xx** errors other than `429` will fail again unchanged. Fix the request first. A `403 IP_WHITELIST_REQUIRED` or `IP_NOT_WHITELISTED` keeps failing until the whitelist or your server's address changes: alert someone with the `ip` from the response instead of retrying.
- **429, 503, other 5xx, timeouts and network errors** are temporary. Retry with exponential back-off (for example 1 s, 2 s, 4 s, up to a minute).

- Retrying `POST /orders` with the same `merchantOrderId` and details is always safe: you get the original order back.
- A retried request needs a new timestamp and therefore a new signature.

INTEGRATION

Chains & confirmations

MekaPay accepts USDT on two networks. A transfer counts only once it is final on its chain.

Supported networks

	BNB SMART CHAIN	TRON
`chain`	`bsc`	`tron`
Token standard	BEP20	TRC20
USDT contract	`0x55d398326f99059fF775485246999027B3197955`	`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`
On-chain decimals	18	6
Address format	`0x` + 40 hex, EIP-55 checksummed	Base58Check, starts with `T`
Final when	The block is at or below the chain's *finalized* block **and** has at least 15 confirmations	The block is solidified (confirmed by 19 of 27 super representatives) **and** the transaction receipt is `SUCCESS`
`requiredConfirmations`	15	19
Typical time to final	About 10 seconds	About 1 minute
Minimum deposit (platform floor)	1 USDT	5 USDT
Explorer	bscscan.com	tronscan.org

The TRON minimum is higher because moving funds out of a TRON address costs several TRX in network fees. Your account can have a higher minimum on either chain. That account minimum applies to fixed amounts and is the default `minAmount` of any-amount orders; a `minAmount` you send explicitly only has to clear the platform floor.

Detection and confirmation

MekaPay watches the USDT contract on both chains for transfers into assigned deposit addresses. On BSC a transfer is usually detected within a few seconds and becomes final about 10 seconds later. On TRON, MekaPay reads only confirmed events, so a transfer typically appears after about a minute and is final almost at once. While a transfer is detected but not yet final, the order is `confirming` and the payment page shows *Confirming n/N*; MekaPay re-checks the transfer against the chain before it counts. If a transfer drops out of the chain or its transaction failed, the deposit becomes `orphaned` or `failed` and never counts.

Amounts

- Send amounts as decimal **strings** with at most 6 decimals, for example `"100.50"`. Do not use floating-point numbers for money.
- MekaPay returns normalised strings without trailing zeros: `"100.50"` becomes `"100.5"`, `"1.000000"` becomes `"1"`.

- USDT on BSC has 18 on-chain decimals, so a received amount (`amountReceived`, `deposit.amount`) can have up to 18 decimals on BSC. Parse amounts with a decimal library or as integers of the smallest unit.
- Network fees are paid by the sender, in BNB on BSC and in TRX (or energy) on TRON, and do not reduce the USDT amount. Exchange withdrawal fees often do: see the warning in Payment modes.

Wrong network or token

> **Only USDT on the order's network is detected.**
> USDT sent on another network, other tokens (BNB, TRX, USDC and so on) and NFTs sent to a deposit address are not detected and not credited automatically. Contact MekaPay support with the transaction hash if this happens; recovery is manual and not guaranteed.

A BSC address is also a valid address on Ethereum and other EVM networks. Make sure users select **BNB Smart Chain (BEP20)** in their wallet or exchange, not Ethereum (ERC20).

GO LIVE

Testing & go-live checklist

Test every path your users can take with small real payments, then check the list below before you switch real users over.

How to test

Version 1 of the gateway has no separate sandbox: tests run against the live API with real USDT. Small BSC orders are the cheapest way to test (the minimum is 1 USDT and BEP20 network fees are low). Your agreed fee applies to test deposits as well, so keep test amounts small. Use test user ids such as `test-user-1` so you can recognise test orders in the portal.

Run API tests from a server whose IP is on your whitelist. Requests from a developer's computer, a CI runner or a staging server are refused with `403 IP_NOT_WHITELISTED` unless MekaPay support has whitelisted that address too.

Before sending money, use the worked examples to unit-test your signing code, and point the sample receivers at your own verification logic.

Test plan

#	SCENARIO	EXPECTED RESULT
1	`GET /ping` from each whitelisted server	Your merchant code, `gatewayEnabled: true`.
2	`GET /ping`, correctly signed, from a computer that is not whitelisted	`403 IP_NOT_WHITELISTED` with that computer's address in `ip`; your code reports it and does not retry.
3	Fixed 1 USDT order on BSC, pay exactly 1	Page shows Paid; `payment.paid` with `amountReceived: "1"`; credited once.
4	Same `POST /orders` sent twice	Second answer `200` with `"created": false` and the same `orderId`.
5	Fixed 2 USDT order, pay 1 and then 1	Page shows the remaining 1 after the first transfer; paid after the second.
6	Fixed 1 USDT order, pay 1.2	Paid with `overpaid: true`, `amountReceived: "1.2"`.
7	Any-amount order on BSC, pay 3	Paid with `amountReceived: "3"`.
8	Any-amount order on BSC with `minAmount: "2"`, pay 1	`deposit.unmatched` with `below_minimum`; the order keeps waiting.
9	Order with `expiresInMinutes: 15`, do not pay	About 18 minutes later (a little more if MekaPay's scanner is catching up): `payment.expired`, page shows Expired.
10	Create an order and cancel it	`status: "cancelled"`, `cancelReason: "merchant"`; no callback.
11	Two orders in a row for the same user and chain	The first is `cancelled` (`superseded`); same address on both.
12	Callback with a changed body or wrong signature	Your endpoint answers `401` and does nothing.
13	Resend a delivered callback from the portal	Your endpoint answers `200` and does not credit again.

#	SCENARIO	EXPECTED RESULT
14	`GET /deposits`	Every test transfer is listed with its match status.
15	Ask MekaPay support to approve an unpaid test order by hand	`payment.paid` with `deposit: null`, `amountReceived` = the approved total and a `manualApproval` object; credited once by the Crediting rule.

Go-live checklist

- [ ] The API Secret is stored only in your server-side secret store: not in front-end code, mobile apps, repositories, tickets or logs.
- [ ] Every server that calls the API (production, failover, background workers) has a static public IP that MekaPay support has whitelisted, and `GET /ping` succeeds from each one, over IPv6 too if the server has IPv6.
- [ ] Server moves, new NAT gateways and other IP changes are agreed with MekaPay support before traffic switches over.
- [ ] Your servers keep time with NTP and send `X-MP-Timestamp` in seconds.
- [ ] You sign the exact body bytes you send, and the path with its query string.
- [ ] Every payment attempt has its own `merchantOrderId`, and retries reuse it.
- [ ] The callback URL is HTTPS with a valid certificate, public, without redirects, and answers `2xx` within 10 seconds.
- [ ] Callbacks are verified over the raw body, with a timestamp check and a constant-time comparison.
- [ ] `eventId` is stored with a unique constraint; each order is credited at most once.
- [ ] `payment.paid` after `payment.expired`, or for an order you cancelled, is credited.
- [ ] `deposit.unmatched` events reach a person who can resolve them.
- [ ] If you show payment details yourself, the network and token are named plainly with a wrong-network warning.
- [ ] `429`, `503` and network errors are retried with back-off.
- [ ] `403 IP_WHITELIST_REQUIRED` and `IP_NOT_WHITELISTED` are not retried; they alert someone, with the `ip` from the response.
- [ ] Someone checks failed callbacks in the portal and reconciles deposits daily.

Support

Contact your MekaPay account manager. For a payment question include the order `reference` (`MP…`) or `orderId`, and the transaction hash if the user has one. Never send your API Secret to anyone, MekaPay staff included.

GO LIVE

Code samples

Complete, dependency-free programs in Node.js, PHP and Python: create an order with a signed request, and receive and verify callbacks.

Requirements and configuration

LANGUAGE	VERSION	USES
Node.js	18 or later	Built-in `crypto`, `fetch` and `http`
PHP	7.4 or later	`hash_hmac`, `hash_equals` and the `curl` extension
Python	3.8 or later	Standard library only (`hmac`, `urllib`, `http.server`)

ENVIRONMENT VARIABLE	MEANING
`MEKAPAY_CLIENT_KEY`	Your API Client Key (`mpk_…`).
`MEKAPAY_API_SECRET`	Your API Secret (`mps_…`).
`MEKAPAY_API_BASE`	Optional. Defaults to `https://api.mekapay.online/api/gateway/v1`.
`PORT`, `HOST`	Optional, for the callback receivers. Default port 3000 on all interfaces.

Run the order programs on a server whose IP is on your whitelist; from anywhere else they get `403 IP_NOT_WHITELISTED`. The whitelist does not apply to callbacks, so it does not affect the receivers.

Create an order

Each program creates the fixed-amount order from the worked example, prints the address and the payment link, and reads the order back by `merchantOrderId`. Replace the order details with your own.

NODE.JS

JAVASCRIPT create-order.mjs

```javascript
// create-order.mjs: create a MekaPay payment order (Node.js 18+, no dependencies)
//   MEKAPAY_CLIENT_KEY=mpk_... MEKAPAY_API_SECRET=mps_... node create-order.mjs
import crypto from 'node:crypto';

const API_BASE = process.env.MEKAPAY_API_BASE || 'https://api.mekapay.online/api/gateway/v1';
const CLIENT_KEY = process.env.MEKAPAY_CLIENT_KEY;
const API_SECRET = process.env.MEKAPAY_API_SECRET; // server-side only, never in a browser or app

// X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp \n METHOD \n pathWithQuery \n rawBody)
function sign(secret, timestamp, method, pathWithQuery, rawBody) {
  const stringToSign = `${timestamp}\n${method}\n${pathWithQuery}\n${rawBody}`;
  return crypto.createHmac('sha256', secret).update(stringToSign, 'utf8').digest('hex');
}

async function mekapayRequest(method, path, payload) {
  const url = new URL(API_BASE + path);
```

```
  const rawBody = payload === undefined ? '' : JSON.stringify(payload);
  const timestamp = String(Math.floor(Date.now() / 1000));
  const res = await fetch(url, {
    method,
    headers: {
      'Content-Type': 'application/json',
      'X-MP-Client-Key': CLIENT_KEY,
      'X-MP-Timestamp': timestamp,
      'X-MP-Signature': sign(API_SECRET, timestamp, method, url.pathname + url.search, rawBody),
    },
    body: rawBody === '' ? undefined : rawBody, // send exactly the bytes you signed
  });
  const json = await res.json().catch(() => null);
  if (!json || !json.success) {
    throw new Error(`MekaPay ${res.status} ${json?.code || ''}: ${json?.message || 'unexpected response'}`);
  }
  return json;
}

// 1. Create (or idempotently re-read) a fixed-amount order for one of your users.
const created = await mekapayRequest('POST', '/orders', {
  merchantOrderId: 'INV-10023',
  userId: 'player_123',
  mode: 'fixed',
  chain: 'bsc',
  currency: 'USDT',
  amount: '100.50',
  expiresInMinutes: 30,
  returnUrl: 'https://merchant.example/deposit/done',
  metadata: { cartId: 'C-778' },
});
const order = created.data;
console.log(created.created ? 'Created order' : 'Order already existed', order.orderId, order.reference);
console.log(`Ask the user to send ${order.amount} ${order.currency} (${order.network}) to ${order.address}`);
console.log('or send them to the payment page:', order.paymentUrl);

// 2. Read it back by your own id (a GET has an empty body).
const found = await mekapayRequest('GET', `/orders?merchantOrderId=${encodeURIComponent(order.merchantOrderId)}`);
console.log('Status:', found.data.status);
```

PHP

PHP create-order.php

```
<?php
// create-order.php: create a MekaPay payment order (PHP 7.4+ with the curl extension)
//   MEKAPAY_CLIENT_KEY=mpk_... MEKAPAY_API_SECRET=mps_... php create-order.php

define('MEKAPAY_API_BASE', getenv('MEKAPAY_API_BASE') ?: 'https://api.mekapay.online/api/gateway/v1');
define('MEKAPAY_CLIENT_KEY', (string) getenv('MEKAPAY_CLIENT_KEY'));
define('MEKAPAY_API_SECRET', (string) getenv('MEKAPAY_API_SECRET')); // server-side only

// X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp \n METHOD \n pathWithQuery \n rawBody)
function mekapay_sign(string $secret, string $timestamp, string $method, string $pathWithQuery, string $rawBody):
string
{
    return hash_hmac('sha256', $timestamp . "\n" . $method . "\n" . $pathWithQuery . "\n" . $rawBody, $secret);
}

function mekapay_request(string $method, string $path, ?array $payload = null): array
{
    $url = MEKAPAY_API_BASE . $path;
    $parts = parse_url($url);
    $pathWithQuery = $parts['path'] . (isset($parts['query']) ? '?' . $parts['query'] : '');
```

```php
    $rawBody = $payload === null ? '' : json_encode($payload, JSON_UNESCAPED_SLASHES | JSON_THROW_ON_ERROR);
    $timestamp = (string) time();

    $ch = curl_init($url);
    curl_setopt_array($ch, [
        CURLOPT_CUSTOMREQUEST => $method,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_TIMEOUT => 15,
        CURLOPT_HTTPHEADER => [
            'Content-Type: application/json',
            'X-MP-Client-Key: ' . MEKAPAY_CLIENT_KEY,
            'X-MP-Timestamp: ' . $timestamp,
            'X-MP-Signature: ' . mekapay_sign(MEKAPAY_API_SECRET, $timestamp, $method, $pathWithQuery, $rawBody),
        ],
    ]);
    if ($method !== 'GET') {
        curl_setopt($ch, CURLOPT_POSTFIELDS, $rawBody); // send exactly the bytes you signed
    }
    $response = curl_exec($ch);
    if ($response === false) {
        throw new RuntimeException('MekaPay request failed: ' . curl_error($ch));
    }
    $status = curl_getinfo($ch, CURLINFO_RESPONSE_CODE);
    $json = json_decode($response, true);
    if (!is_array($json) || empty($json['success'])) {
        throw new RuntimeException(sprintf('MekaPay %d %s: %s', $status, $json['code'] ?? '', $json['message'] ??
'unexpected response'));
    }
    return $json;
}

// 1. Create (or idempotently re-read) a fixed-amount order for one of your users.
$created = mekapay_request('POST', '/orders', [
    'merchantOrderId' => 'INV-10023',
    'userId' => 'player_123',
    'mode' => 'fixed',
    'chain' => 'bsc',
    'currency' => 'USDT',
    'amount' => '100.50',
    'expiresInMinutes' => 30,
    'returnUrl' => 'https://merchant.example/deposit/done',
    'metadata' => ['cartId' => 'C-778'],
]);
$order = $created['data'];
echo ($created['created'] ? 'Created order ' : 'Order already existed ') . $order['orderId'] . ' ' .
$order['reference'] . PHP_EOL;
echo "Ask the user to send {$order['amount']} {$order['currency']} ({$order['network']}) to {$order['address']}" .
PHP_EOL;
echo 'or send them to the payment page: ' . $order['paymentUrl'] . PHP_EOL;

// 2. Read it back by your own id (a GET has an empty body).
$found = mekapay_request('GET', '/orders?merchantOrderId=' . rawurlencode($order['merchantOrderId']));
echo 'Status: ' . $found['data']['status'] . PHP_EOL;
```

PYTHON

PYTHON create_order.py

```python
# create_order.py: create a MekaPay payment order (Python 3.8+, standard library only)
#   MEKAPAY_CLIENT_KEY=mpk_... MEKAPAY_API_SECRET=mps_... python create_order.py
import hashlib
import hmac
import json
import os
```

```python
import time
import urllib.error
import urllib.parse
import urllib.request

API_BASE = os.environ.get("MEKAPAY_API_BASE", "https://api.mekapay.online/api/gateway/v1")
CLIENT_KEY = os.environ["MEKAPAY_CLIENT_KEY"]
API_SECRET = os.environ["MEKAPAY_API_SECRET"]  # server-side only, never in a browser or app


# X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp \n METHOD \n pathWithQuery \n rawBody)
def sign(secret, timestamp, method, path_with_query, raw_body):
    string_to_sign = f"{timestamp}\n{method}\n{path_with_query}\n{raw_body}"
    return hmac.new(secret.encode("utf-8"), string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()


def mekapay_request(method, path, payload=None):
    url = API_BASE + path
    parts = urllib.parse.urlsplit(url)
    path_with_query = parts.path + ("?" + parts.query if parts.query else "")
    raw_body = "" if payload is None else json.dumps(payload, separators=(",", ":"))
    timestamp = str(int(time.time()))
    request = urllib.request.Request(
        url,
        method=method,
        data=None if method == "GET" else raw_body.encode("utf-8"),  # send exactly the bytes you signed
        headers={
            "Content-Type": "application/json",
            "X-MP-Client-Key": CLIENT_KEY,
            "X-MP-Timestamp": timestamp,
            "X-MP-Signature": sign(API_SECRET, timestamp, method, path_with_query, raw_body),
        },
    )
    try:
        with urllib.request.urlopen(request, timeout=15) as response:
            return json.loads(response.read())
    except urllib.error.HTTPError as error:
        try:
            body = json.loads(error.read())
        except ValueError:
            body = {}
        message = body.get("message", "unexpected response")
        raise RuntimeError(f"MekaPay {error.code} {body.get('code', '')}: {message}") from None


if __name__ == "__main__":
    # 1. Create (or idempotently re-read) a fixed-amount order for one of your users.
    created = mekapay_request("POST", "/orders", {
        "merchantOrderId": "INV-10023",
        "userId": "player_123",
        "mode": "fixed",
        "chain": "bsc",
        "currency": "USDT",
        "amount": "100.50",
        "expiresInMinutes": 30,
        "returnUrl": "https://merchant.example/deposit/done",
        "metadata": {"cartId": "C-778"},
    })
    order = created["data"]
    print("Created order" if created["created"] else "Order already existed", order["orderId"], order["reference"])
    print(f"Ask the user to send {order['amount']} {order['currency']} ({order['network']}) to {order['address']}")
    print("or send them to the payment page:", order["paymentUrl"])

    # 2. Read it back by your own id (a GET has an empty body).
    found = mekapay_request("GET", "/orders?merchantOrderId=" + urllib.parse.quote(order["merchantOrderId"],
```

```
safe=""))
    print("Status:", found["data"]["status"])
```

Receive and verify callbacks

Each receiver accepts `POST /mekapay/callback`, verifies the signature and timestamp over the raw body, ignores repeated `eventId`s and credits each paid order once. The in-memory sets stand in for database tables with unique constraints.

NODE.JS

JAVASCRIPT callback-server.mjs

```
// callback-server.mjs: receive and verify MekaPay callbacks (Node.js 18+, no dependencies)
//   MEKAPAY_API_SECRET=mps_... PORT=3000 node callback-server.mjs
import http from 'node:http';
import crypto from 'node:crypto';

const API_SECRET = process.env.MEKAPAY_API_SECRET;
const PORT = Number(process.env.PORT || 3000);
const TOLERANCE_SEC = 300;

// X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp + "." + rawBody)
function isValidSignature(rawBody, timestamp, signature) {
  if (!/^\d{1,12}$/.test(timestamp || '') || !/^[0-9a-f]{64}$/i.test(signature || '')) return false;
  if (Math.abs(Date.now() / 1000 - Number(timestamp)) > TOLERANCE_SEC) return false;
  const expected = crypto.createHmac('sha256', API_SECRET).update(`${timestamp}.${rawBody}`, 'utf8').digest();
  return crypto.timingSafeEqual(expected, Buffer.from(signature, 'hex'));
}

// Replace these with database tables that have a UNIQUE constraint.
const processedEventIds = new Set();
const creditedOrderIds = new Set();

function handleEvent(event) {
  const { order, deposit } = event.data;
  if (event.event === 'payment.paid') {
    if (creditedOrderIds.has(order.orderId)) return; // credit each order at most once
    creditedOrderIds.add(order.orderId);
    console.log(`Credit ${order.amountReceived} ${order.currency} to ${order.userId} for ${order.merchantOrderId}
(${order.orderId})`);
  } else if (event.event === 'payment.expired') {
    console.log(`Order ${order.orderId} expired; received ${order.amountReceived} ${order.currency}`);
  } else if (event.event === 'deposit.unmatched') {
    console.log(`Unmatched deposit of ${deposit.amount} USDT from ${deposit.userId} (${deposit.unmatchedReason}):
${deposit.explorerUrl}`);
  }
}

http.createServer((req, res) => {
  if (req.method !== 'POST' || req.url !== '/mekapay/callback') {
    res.writeHead(404).end();
    return;
  }
  const chunks = [];
  req.on('data', chunk => chunks.push(chunk));
  req.on('end', () => {
    const rawBody = Buffer.concat(chunks).toString('utf8'); // verify the raw bytes, not re-serialised JSON
    if (!isValidSignature(rawBody, req.headers['x-mp-timestamp'], req.headers['x-mp-signature'])) {
      res.writeHead(401).end('invalid signature');
      return;
    }
```

```
    const event = JSON.parse(rawBody);
    if (!processedEventIds.has(event.eventId)) {
      processedEventIds.add(event.eventId);
      handleEvent(event);
    }
    res.writeHead(200, { 'Content-Type': 'text/plain' }).end('ok'); // answer 2xx within 10 s
  });
}).listen(PORT, process.env.HOST, () => console.log(`Listening on http://localhost:${PORT}/mekapay/callback`));
```

PHP

PHP callback.php

```php
<?php
// callback.php: receive and verify MekaPay callbacks (PHP 7.4+). Point your callback URL at this script.

const MEKAPAY_TOLERANCE_SEC = 300;

// X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp + "." + rawBody)
function mekapay_verify_callback(string $secret, string $rawBody, string $timestamp, string $signature): bool
{
    if (!preg_match('/\A\d{1,12}\z/', $timestamp) || !preg_match('/\A[0-9a-fA-F]{64}\z/', $signature)) {
        return false;
    }
    if (abs(time() - (int) $timestamp) > MEKAPAY_TOLERANCE_SEC) {
        return false;
    }
    $expected = hash_hmac('sha256', $timestamp . '.' . $rawBody, $secret);
    return hash_equals($expected, strtolower($signature));
}

$apiSecret = (string) getenv('MEKAPAY_API_SECRET');          // server-side only
$rawBody = (string) file_get_contents('php://input');       // verify the raw bytes, not re-encoded JSON
$timestamp = $_SERVER['HTTP_X_MP_TIMESTAMP'] ?? '';
$signature = $_SERVER['HTTP_X_MP_SIGNATURE'] ?? '';

if (!mekapay_verify_callback($apiSecret, $rawBody, $timestamp, $signature)) {
    http_response_code(401);
    exit('invalid signature');
}

$event = json_decode($rawBody, true);
$order = $event['data']['order'] ?? null;
$deposit = $event['data']['deposit'] ?? null;

// Insert $event['eventId'] into a table with a UNIQUE constraint first. If it is
// already there, this is a retry or a resend: answer 200 and stop.
switch ($event['event']) {
    case 'payment.paid':
        // Credit $order['amountReceived'] USDT to $order['userId'] once per $order['orderId'].
        break;
    case 'payment.expired':
        // Mark $order['merchantOrderId'] as expired. It can still become paid later.
        break;
    case 'deposit.unmatched':
        // Review $deposit['amount'] USDT from $deposit['userId'] ($deposit['unmatchedReason']).
        break;
}

http_response_code(200); // answer 2xx within 10 s
echo 'ok';
```

PYTHON

PYTHON callback_server.py

```python
# callback_server.py: receive and verify MekaPay callbacks (Python 3.8+, standard library only)
#   MEKAPAY_API_SECRET=mps_... PORT=3000 python callback_server.py
import hashlib
import hmac
import json
import os
import re
import time
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer

API_SECRET = os.environ["MEKAPAY_API_SECRET"]
HOST = os.environ.get("HOST", "")
PORT = int(os.environ.get("PORT", "3000"))
TOLERANCE_SEC = 300

# Replace these with database tables that have a UNIQUE constraint.
processed_event_ids = set()
credited_order_ids = set()


# X-MP-Signature = hex HMAC-SHA256(apiSecret, timestamp + "." + rawBody)
def is_valid_signature(raw_body, timestamp, signature):
    if not re.fullmatch(r"[0-9]{1,12}", timestamp) or not re.fullmatch(r"[0-9a-fA-F]{64}", signature):
        return False
    if abs(time.time() - int(timestamp)) > TOLERANCE_SEC:
        return False
    signed = timestamp.encode("ascii") + b"." + raw_body
    expected = hmac.new(API_SECRET.encode("utf-8"), signed, hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, signature.lower())


def handle_event(event):
    order, deposit = event["data"].get("order"), event["data"].get("deposit")
    if event["event"] == "payment.paid":
        if order["orderId"] in credited_order_ids:
            return  # credit each order at most once
        credited_order_ids.add(order["orderId"])
        print(f"Credit {order['amountReceived']} {order['currency']} to {order['userId']} "
              f"for {order['merchantOrderId']} ({order['orderId']})", flush=True)
    elif event["event"] == "payment.expired":
        print(f"Order {order['orderId']} expired; received {order['amountReceived']} {order['currency']}",
flush=True)
    elif event["event"] == "deposit.unmatched":
        print(f"Unmatched deposit of {deposit['amount']} USDT from {deposit['userId']} "
              f"({deposit['unmatchedReason']}): {deposit['explorerUrl']}", flush=True)


class CallbackHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        if self.path != "/mekapay/callback":
            self.send_error(404)
            return
        raw_body = self.rfile.read(int(self.headers.get("Content-Length", 0)))  # the raw bytes
        timestamp = self.headers.get("X-MP-Timestamp", "")
        signature = self.headers.get("X-MP-Signature", "")
        if not is_valid_signature(raw_body, timestamp, signature):
            self.send_error(401, "invalid signature")
            return
        event = json.loads(raw_body)
        if event["eventId"] not in processed_event_ids:
            processed_event_ids.add(event["eventId"])
```

```
            handle_event(event)
        self.send_response(200)  # answer 2xx within 10 s
        self.send_header("Content-Type", "text/plain")
        self.end_headers()
        self.wfile.write(b"ok")


if __name__ == "__main__":
    server = ThreadingHTTPServer((HOST, PORT), CallbackHandler)
    print(f"Listening on http://localhost:{PORT}/mekapay/callback", flush=True)
    server.serve_forever()
```

Self-test values

Run your own signing code with the test secret `mps_AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8` and compare:

INPUT	EXPECTED `X-MP-SIGNATURE`
`POST /orders` at `1794391200`	`59e0d2ed6ce46e58fe8f87885dc1100c3c71c36fe4e7d4ecde00ff503c77a394`
`GET /orders?merchantOrderId=INV-10023` at `1794391200`	`a1b80522211ffc539ab2c081439d6c6ed9fe122a007d023a69502d3bfea31f93`
Callback body at `1794391512`	`c10c65cc2f03d52438d090c3823914dcc18df2f2886f3af11368376d27ad775f`

The create-order programs above send exactly the worked-example body. Run at timestamp `1794391200` with the test secret, they produce the first two signatures.

Using a web framework

Verify callbacks over the raw body, before the framework parses JSON:

- **Express**: `app.post('/mekapay/callback', express.raw({ type: 'application/json' }), handler)`, then `req.body.toString('utf8')` is the raw body.
- **Laravel**: `$request->getContent()` and `$request->header('X-MP-Signature')`; exclude the route from CSRF protection.
- **Django**: `request.body` (bytes) and `request.headers['X-MP-Signature']`; mark the view `@csrf_exempt`.
- **Flask**: `request.get_data()` and `request.headers['X-MP-Signature']`.

GO LIVE

Changelog

Changes to the Payment Gateway API and to this documentation.

Balances and settlements · 11 November 2026

- New `GET /balance`: your balance per chain, the confirmed deposits minus your deposit fee and your settlements.
- New settlement addresses: up to 3 of your own wallets per chain, added in the merchant portal with your two-factor code, approved by MekaPay and usable 48 hours after approval (`GET /settlement-addresses`).
- New settlements: `POST /settlements`, `GET /settlements`, `GET /settlements/:settlementId` and `POST /settlements/:settlementId/cancel`. Fee 1 USDT on BSC and 3 USDT on TRON, deducted from the amount; minimum 10 USDT. Off until MekaPay enables it for your account.
- New callback events `settlement.completed` and `settlement.rejected`. Ignore events you do not know: see Events.

Manual approval by staff · 11 November 2026

- MekaPay staff can mark a `pending` or `expired` order as paid by hand, for example for integration tests or after a support decision on an underpaid order. You receive a normal signed `payment.paid` with `deposit: null`: see Manual approval.
- New order field `manualApproval` (`approvedAt`, `amount`, `amountOnChain`), present only on such orders. `amountReceived` is the approved total; credit by the Crediting rule as usual.

IP whitelist required · 11 November 2026

- API requests are accepted only from the server IP addresses whitelisted for your account. Give MekaPay support the static public IPv4 and IPv6 addresses or ranges your servers call from: see IP whitelist.
- New errors `403 IP_WHITELIST_REQUIRED` and `403 IP_NOT_WHITELISTED`, with the caller's address in `ip`.
- The whitelist applies to the older client API (`/api/client/v1`) as well. The payment page and callbacks are not affected.
- On request, merchant portal access can also be restricted to listed IP addresses.

v1.0 · 11 November 2026

First release of the Payment Gateway API.

- Signed merchant requests with `X-MP-Client-Key`, `X-MP-Timestamp` and `X-MP-Signature`.
- Endpoints: `GET /ping`, `POST /orders`, `GET /orders/:orderId`, `GET /orders`, `POST /orders/:orderId/cancel`, `GET /deposits`.
- Fixed-amount and any-amount payment modes.
- USDT on BNB Smart Chain (BEP20) and TRON (TRC20), with a permanent deposit address per user and chain.
- Hosted payment page at `https://mekapay.online/pay/<orderId>` in English and Simplified Chinese.

- Signed callbacks `payment.paid`, `payment.expired` and `deposit.unmatched`, with retries over about 46 hours and resend from the merchant portal.
- The transfer's block time decides which order it belongs to. A transfer sent in time that confirms after the order expired or was cancelled pays it if it completes it, else it is reported as `deposit.unmatched` (`after_expiry` or `order_cancelled`). Every confirmed deposit is reported; credit orders by their running total (`amountReceived` minus what you already credited).
- Orders expire only after MekaPay's chain scanner has caught up with the order window.
- Fair callback delivery across merchants, with slow endpoints deferred; resend guards `409 RESEND_PENDING` and `429 RESEND_LIMIT`.
- Request bodies must be JSON (`415 UNSUPPORTED_MEDIA_TYPE` otherwise).

The older contract-based client API (`/api/client/v1`) is documented separately. Apart from the IP whitelist, it is unchanged.