



Gasless Relayer > API Reference

Gasless Relayer

API Reference

Complete API documentation for the gasless relayer system

Base URL

`https://api.plasma.to`



All endpoints are **internal-only** and require authentication via shared secret. External traffic is blocked at the middleware level.

Authentication



These examples are for server-side code only!

Never expose the `INTERNAL_SECRET` in client-side code. All API calls must be made from your backend (Next.js API routes, Server Actions, etc.)

All requests must include:

`X-Internal-Secret: <shared-secret>`

`X-User-IP: <forwarded-user-ip> # Set by dashboard backend, never by client`



Endpoints

Submit Authorization

Submit a signed authorization for gasless transfer.

POST /submit Server-Side Example



```
// Headers
{
  >
  "X-Internal-Secret": "your-secret",
  "X-User-IP": "192.168.1.1"
}

// Request Body
{
  "authorization": {
    "from": "0x123...",
    "to": "0x456...",
    "value": "1000000", // 1 USDT0 (6 decimals)
    "validAfter": "0",
    "validBefore": "1234567890",
    "nonce": "0xabc123..."
  },
  "signature": "0xdef456..."
}

// Response
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "queued"
}
```

Check Status

Get the status of a submitted transfer.

GET /status/:id Example



```
// Headers
{
  >
  "X-Internal-Secret": "your-secret"
}

// Response (Success)
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "confirmed",
  "txHash": "0x789...",
  "gasUsed": "50000"
}

// Response (Failed)
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "status": "failed",
  "error": "Insufficient balance"
}
```

Get Rate Limits

Check current rate limit usage for an address and IP.

GET /limits Example



```
// Headers
{
  >
  "X-Internal-Secret": "your-secret",
  "X-User-IP": "forwarded-user-ip"
}

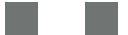
// Query Parameters
{
  "address": "0x123..."
}

// Response
{
  "addressLimits": {
    "dailyTransferCount": 3,
    "dailyTransferLimit": 10,
    "dailyVolume": "30000000000", // 3000 USDT0
    "dailyVolumeLimit": "100000000000" // 10000 USDT0
  },
  "ipLimits": {
    "dailyTransferCount": 5,
    "dailyTransferLimit": 20
  },
  "resetsAt": "2024-01-01T00:00:00Z"
}
```

Status Codes

Code	Description
200	Success
400	Invalid request data
401	Invalid or missing authentication
429	Rate limit exceeded
500	Internal server error
503	Service unavailable (emergency stop)

Rate Limits



Per Address

- **10 transfers** per day
- **10,000 USDT0** daily volume
- Resets at 00:00 UTC

Per IP

- **20 transfers** per day (across all addresses)
- Resets at 00:00 UTC

Error Responses

All errors follow this format:

```
{
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Daily transfer limit exceeded for address",
    "details": {
      "limit": 10,
      "used": 10,
      "resetsAt": "2024-01-01T00:00:00Z"
    }
  }
}
```











Error Codes

Code	Description
INVALID_SIGNATURE	Signature validation failed
INVALID_AUTHORIZATION	Authorization data invalid
NONCE_ALREADY_USED	Nonce has been used
INSUFFICIENT_BALANCE	User has insufficient tokens

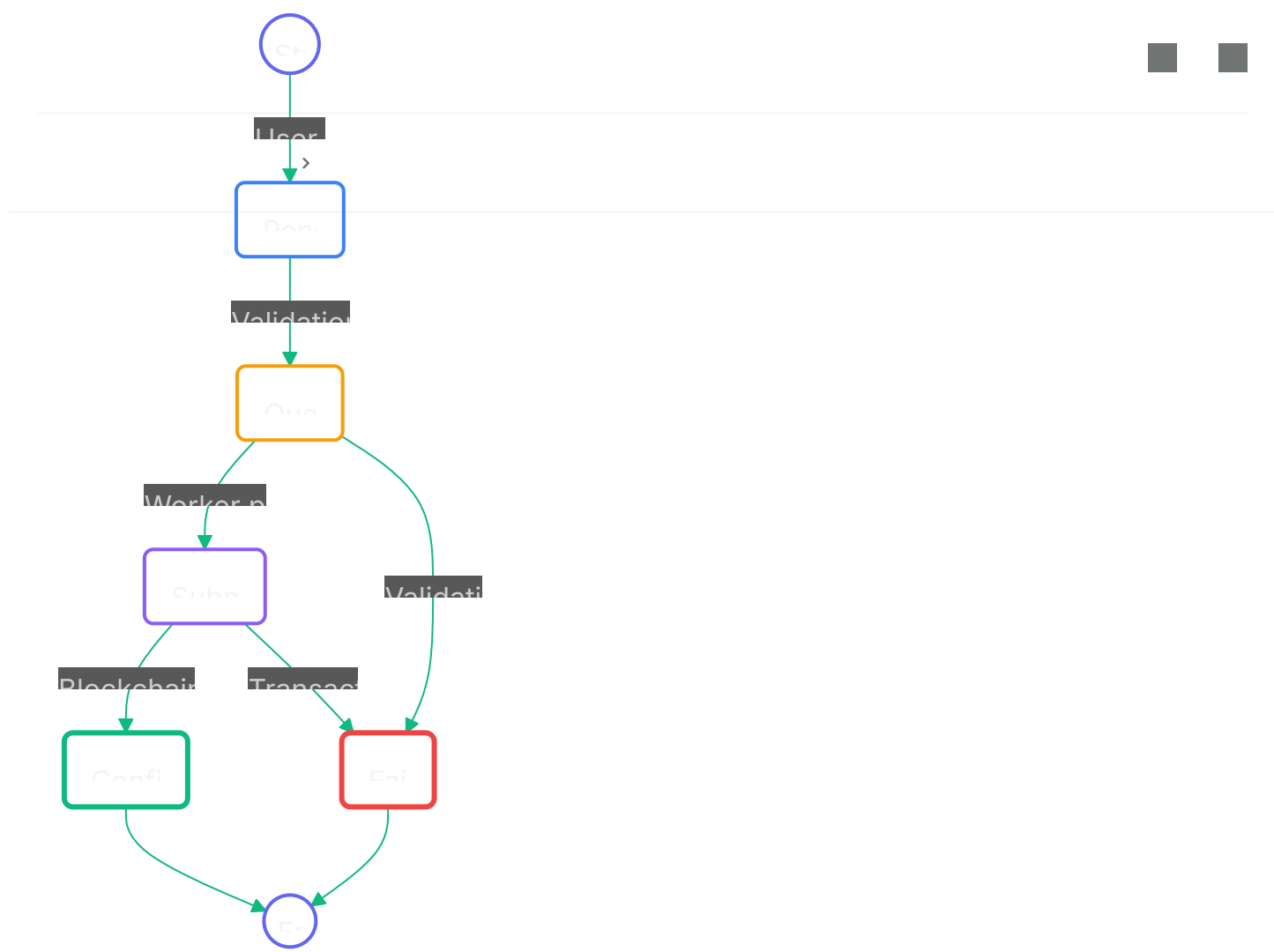
Code	Description
RATE_LIMIT_EXCEEDED	Rate limit hit
AMOUNT_TOO_LOW	Below minimum transfer amount
EMERGENCY_STOP	System is halted
INTERNAL_ERROR	Unexpected error occurred

Validation Rules

 All validation happens server-side in the relayer:

-  Signature must be valid EIP-712
-  Amount must be ≥ 1 USDT0
-  Nonce must be unique (32 bytes)
-  ValidBefore must be 1-24 hours from now
-  From address must have sufficient balance
-  To address must not be zero address
-  Both IP and address within rate limits

Transaction Lifecycle



Status Descriptions

Status	Description
pending	Authorization received, awaiting validation
queued	Validated and queued for processing
submitted	Transaction submitted to blockchain
confirmed	Transaction confirmed on-chain
failed	Processing failed (see error message)

Dashboard Backend Implementation

Your dashboard needs a server-side component to securely communicate with the relay API. Here are two approaches:

Option 1: Next.js Server Action (Recommended)



Server actions provide better security by default with automatic CSRF protection and are harder to abuse from external sources.

Create a server action:



```
'use server'
import { headers } from 'next/headers'

export async function submitToRelayer(
  authorization: any,
  signature: string
) {
  // Get user IP from headers
  const headersList = headers()
  const forwardedFor = headersList.get('x-forwarded-for')
  const userIP = forwardedFor?.split(',')[0] || 'unknown'

  const response = await fetch('https://api.plasma.to/submit', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-Internal-Secret': process.env.INTERNAL_SECRET!,
      'X-User-IP': userIP
    },
    body: JSON.stringify({ authorization, signature })
  })

  if (!response.ok) {
    throw new Error(`Relayer error: ${response.statusText}`)
  }

  return response.json()
}
```

Frontend usage:



```
// In your React component
import { submitToRelayer } from '@app/actions'

const result = await submitToRelayer(authorization, signature)
```

Option 2: Next.js API Route

API routes are publicly accessible endpoints. While they keep secrets secure on the server, they may require additional authentication checks to prevent abuse.

Create `/app/relay/route.ts` :



```
import { NextRequest, NextResponse } from 'next/server'

export async function POST(request: NextRequest) {
  // Get the real user IP
  const forwardedFor = request.headers.get('x-forwarded-for')
  const realIp = request.headers.get('x-real-ip')
  const userIP = forwardedFor?.split(',')[0] || realIp || 'unknown'

  // Get request body
  const body = await request.json()


  // Forward to relayer API with authentication
  const response = await fetch('https://api.plasma.to/submit', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-Internal-Secret': process.env.INTERNAL_SECRET!,
      'X-User-IP': userIP
    },
    body: JSON.stringify(body)
  })

  const data = await response.json()
  return NextResponse.json(data, { status: response.status })
}
```

Frontend usage:

```
// In your React component (client-side)
const response = await fetch('/relay', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ authorization, signature })
})
```



 Both approaches keep the INTERNAL_SECRET secure on the server and properly forward the user's real IP address for rate limiting. **Server actions are recommended** for better default security with built-in CSRF protection.

[< Technical Specification](#)

[Development Guide >](#)

Powered by Mintlify