flot

V2.0

PAYMENT GATEWAY · INTEGRATION GUIDE

Accept payments through the Flot network.

A complete instruction set for merchants to integrate the Flot payment gateway from KYC and key generation to payment links and webhooks.

AUDIENCE
Flot Merchants

ENVIRONMENT
Production API

RELEASED
June 2026

INTRODUCTION

Flot as your payment processor.

Flot can act as a payment processor. The Flot API integrates directly into your merchant website so payment links can be generated on demand and paid through the Flot app or by credit card.

Contents

01 **Technical requirements** p. 03
Order identifiers and the webhook endpoint your application must expose.

02 **Steps for the merchant** p. 04
Registration, KYC, RSA key generation, and merchant onboarding with Flot Staff.

03 **Payment link generation** p. 06
Production endpoint, order status tracking, and the signature pipeline.

04 **Signature code examples** p. 07
Node.js and Go reference implementations for body-present and body-less requests.

05 **Outgoing webhooks** p. 08
Authentication, payload structure, and retry policy.

SECTION 01

Technical requirements for the merchant application.

Before integrating with Flot, your application must satisfy two requirements: every order needs a stable identifier, and you must expose a webhook endpoint that Flot can notify when an order is processed.

REQ · 01

Each order needs an ID

Every order must carry a unique identifier in your system. This identifier is what Flot references throughout the lifecycle.

REQ · 02

Webhook endpoint to track status

Implement an endpoint that receives webhook notifications so order status can be tracked once Flot finishes processing.

Webhook endpoint specification

- The endpoint must accept the `POST` method.
- The endpoint must be protected with Basic authorization.
- The endpoint must be highly available — Flot Backend sends a webhook notification only once after order processing.

One-shot delivery.
Because Flot does not retry webhooks, treat the endpoint as a critical path. Queue the request immediately on receipt and acknowledge with a 2xx response before any heavy processing.

What good looks like

An endpoint behind a load balancer with redundancy, request logging, and idempotent handling of repeated orderId + flotRequestId combinations. Treat the webhook as the source of truth that flips your order from pending to a terminal state.

SECTION 02

Steps for the merchant to follow.

01 **Register as a user** in the Flot application.

02 **Upgrade KYC level** to the maximum level possible. This increases the maximum possible order amount available to your merchant account.

03 **Generate a pair of RSA-4096 keys** (via OpenSSL).

a. Generate a private key:

```bash
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:4096
```

b. Extract the matching public key:

```bash
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

c. Store the private key securely — it signs your payment-link requests.

d. Prepare a Base64-encoded version of the public key:

```bash
cat public_key.pem | base64
```

Continued on the next page → Step 04: contacting Flot Staff.

SECTION 02 · CONTINUED

Contact Flot Staff to finish onboarding.

The final onboarding step requires direct coordination with Flot. Reach out using either of the numbers below to have your merchant account created and your destination wallets assigned.

04

Contact Flot Staff at `+232 80 800100` or `+232 99 800100` to create a merchant ID and assign destination wallets.

a. By default, orders can be paid in the Flot app.

b. If you want orders to also be payable by credit card, specify this when contacting Flot Staff.

c. Provide Flot Staff with: merchant name, webhook URL, webhook basic-auth credentials, and the Base64-encoded RSA-4096 public key.

d. Ask Flot Staff to provide your Merchant ID — required when requesting payment links.

Keep your Merchant ID safe.
You'll reference it on every payment-link request, alongside the signature generated with your private key.

SECTION 03

Payment link generation.

Payment links are generated by making a request to the payment-link generation endpoint. Refer to flot-merchant-api.yaml for the full request and response schema.

PRODUCTION BASE URL

```
BASE  https://api.app.flotme.ai
```

PAYMENT LINK ENDPOINT

```
POST  /merchants/private/v1/payment-links
```

Tracking order status

Once a payment link is created, the submitted orderId and the internally created order ID (payment attempt ID) can be used to track the payment attempt via:

```
GET  /merchants/private/v1/external-orders/{externalOrderId}/payment-attempts/{internalOrderId}
```

Signature computation

The payment-link endpoint requires the `X-Flot-Merchant-Signature` header. Compute its value from the request-related string using the following pipeline:

STEP 1	STEP 2	STEP 3	STEP 4
RSA-4096	**SHA-512**	**PSS Padding**	**Base64**
Sign with private key	Hash the string	Probabilistic signature	Encode the signature

The request-related string is the stringified request body when a body is present; otherwise it is the canonical request string {METHOD}\n{PATH}. Body-less requests must also include the X-Flot-Merchant-Id header.

```
HEADER  X-Flot-Merchant-Signature: <base64( RSA-4096-PSS( SHA-512( requestString ) ) )>
```

SECTION 03 · CONTINUED

Signature code examples.

Reference implementations in Node.js and Go. Use the same private key whose public counterpart you submitted during onboarding — any mismatch causes Flot to reject the request.

Body-present requests — sign the stringified JSON body

flot-signature-gen.js

```
const crypto = require('crypto');
const fs = require('fs');
const privateKey =
  fs.readFileSync('private_key.pem', 'utf8');
const requestBody = {
  merchantId: "6efaa63d-c1eb-49eb-bdb7-ca24a6f3637e",
  type: "in-app",
  payload: { orderId: "some-test-order-id-12345",
             currency: "SLE", amount: "10" }
};
const signer = crypto.createSign('RSA-SHA512');
signer.update(JSON.stringify(requestBody));
const signature = signer.sign({
  key: privateKey,
  padding: crypto.constants.RSA_PKCS1_PSS_PADDING,
  saltLength: crypto.constants.RSA_PSS_SALTLEN_DIGEST,
}, 'base64');
```

flot-signature-gen.go

```
hashed := sha512.Sum512(payload)
signature, err := rsa.SignPSS(
  rand.Reader, privateKey, crypto.SHA512,
  hashed[:], &rsa.PSSOptions{
    SaltLength: rsa.PSSSaltLengthEqualsHash,
    Hash:       crypto.SHA512,
  })
if err != nil { return "", err }
sig := base64.StdEncoding.
  EncodeToString(signature)

// payload = json.Marshal(requestBody)
// struct fields: merchantId, type,
//   payload{ orderId, currency, amount }
return sig, nil
```

Body-less requests — sign the canonical {METHOD}\n{PATH} string

flot-signature-gen-bodyless.js

```
const crypto = require('crypto');
const fs = require('fs');
const privateKey =
  fs.readFileSync('private_key.pem', 'utf8');
const canonical =
  "GET\n/merchants/private/v1/external-
   orders/some-test-order-id-62342352/
   payment-attempts/d8cd7824-...090c";
const signer = crypto.createSign('RSA-SHA512');
signer.update(canonical);
const signature = signer.sign({
  key: privateKey,
  padding: crypto.constants.RSA_PKCS1_PSS_PADDING,
  saltLength: crypto.constants.RSA_PSS_SALTLEN_DIGEST,
}, 'base64');
```

flot-signature-gen-bodyless.go

```
func canonicalRequestString(
    method, path string) string {
  return fmt.Sprintf("%s\n%s",
    method, path)
}
method := "GET"
path := "/merchants/private/v1/external-
  orders/my-another-order-7342526/
  payment-attempts/19544284-...0b68"
dataToSign :=
  canonicalRequestString(method, path)
sig, err := signPayload(privateKey,
  []byte(dataToSign))
// remember X-Flot-Merchant-Id header
fmt.Println(sig)
```

X-Flot-Merchant-Id is required for body-less requests.
Because there is no body to carry the merchant context, the signature is computed over the canonical {METHOD}\n{PATH} string and the merchant is identified by this header instead.

SECTION 04

Outgoing webhooks.

Authentication

Basic authentication should be used on the merchant side. Flot includes the credentials you provided at onboarding in every webhook request.

Payload structure

APPLICATION/JSON

```
{
  "orderId": "order-123",
  "flotRequestId": "123456",
  "status": "completed"
}
```

FIELD	TYPE	DESCRIPTION
orderId	string	Order ID in the merchant's system.
flotRequestId	string	Unique transfer request ID in Flot.
status	enum	One of completed or failed.

Handling failed status.
If a webhook arrives with status = failed, the end-user experienced a payment error while paying with credit card. They can retry later — when your backend receives this status, leave the order in pending state rather than marking it failed.

Retry policy

As of now, Flot Backend sends a webhook notification only once. Design your endpoint for high availability and acknowledge requests promptly with a 2xx response, queuing any heavy work asynchronously.

NEED HELP?

Reach Flot Staff for onboarding & support.

+232 80 800100 · +232 99 800100