EcomLabs

STEP-BY-STEP GUIDE

Verify Shopify webhook HMACs in Node

The Node code that proves every webhook came from Shopify, for Express and a Next.js route handler: raw body, constant-time comparison, 401 on a bad signature, a fast answer and no duplicates.

FOR	Shopify apps and webhook endpoints in Node.js
TESTED WITH	Node 22.22.3 · Express 5.3.0 · curl 8.19 · OpenSSL 3.5.6 · Windows 11
UPDATED	November 3, 2026
DOCS	shopify.dev/docs/apps/build/webhooks/verify-deliveries
ARTICLE	ecomlabs.dev/g/hmac-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will build
Step 01	The verification function
Step 02	Express
Step 03	Next.js route handler
Step 04	The queue and duplicates
Step 05	Try it locally with curl
Step 06	Before you ship it
When you finish	Checklist

BEFORE YOU START

What you will build

An endpoint that receives Shopify webhooks, computes an HMAC-SHA256 of the raw body with your app's client secret and compares it in constant time with the `X-Shopify-Hmac-Sha256` header. If it matches, it queues the work and returns `200`; if not, it returns `401` and processes nothing. There are two versions: Express and a Next.js route handler.

We tested it on November 2, 2026 with Node 22.22.3, Express 5.3.0, curl 8.19 and OpenSSL 3.5.6 on Windows 11. The route handler was tested by calling it with a Node 22 `Request`, not in a Next.js build. `test_secret_do_not_use` is a placeholder: your secret belongs in an environment variable.

STEP 01

The verification function

Save it as `verify.mjs` (in Next.js, as `lib/verify-shopify-webhook.js`). No dependencies:

```
import { createHmac, timingSafeEqual } from "node:crypto";

export function verifyShopifyWebhook(rawBody, hmacHeader, secret) {
  if (!secret || typeof hmacHeader !== "string") return false;
  if (typeof rawBody !== "string" && !Buffer.isBuffer(rawBody)) return false;

  const expected = Buffer.from(createHmac("sha256", secret).update(rawBody).digest("base64"));
  const received = Buffer.from(hmacHeader);

  return expected.length === received.length && timingSafeEqual(expected, received);
}
```

Three decisions that matter. With no secret it returns `false`, because anyone can sign an HMAC made with an empty key. If the body is neither a `Buffer` nor a string, something parsed it first and the signature can no longer be checked. And it compares lengths before `timingSafeEqual`, which throws a `RangeError` on different lengths: without that line, a short or missing header gives a 500 instead of a 401.

STEP 02

Express

Save this file as `server.mjs`. `express.raw()` runs on the webhook route only, so `req.body` is the original `Buffer`; `express.json()` comes after, for the rest of your API:

```
import express from "express";
import { verifyShopifyWebhook } from "./verify.mjs";
import { enqueueWebhook } from "./webhook-queue.mjs";

const secrets = [process.env.SHOPIFY_API_SECRET, process.env.SHOPIFY_API_SECRET_OLD].filter(Boolean);
```

```
export const app = express();

app.post("/webhooks/shopify", express.raw({ type: "application/json" }), async (req, res) => {
  const hmac = req.get("x-shopify-hmac-sha256");
  if (!secrets.some((secret) => verifyShopifyWebhook(req.body, hmac, secret))) {
    return res.status(401).send("invalid hmac");
  }

  await enqueueWebhook({
    id: req.get("x-shopify-webhook-id"),
    topic: req.get("x-shopify-topic"),
    shop: req.get("x-shopify-shop-domain"),
    body: req.body.toString("utf8"),
  });
  res.sendStatus(200);
});

app.use(express.json());
```

To start it, a `start.mjs` that imports `app` and calls `app.listen(8798)`. If `enqueueWebhook` throws, Express 5 answers 500 and Shopify retries the delivery.

STEP 03

Next.js route handler

Create `app/api/webhooks/shopify/route.js`. It reads the body once, as bytes, before any `request.json()`:

```
import { verifyShopifyWebhook } from "@/lib/verify-shopify-webhook";
import { enqueueWebhook } from "@/lib/webhook-queue";

const secrets = [process.env.SHOPIFY_API_SECRET, process.env.SHOPIFY_API_SECRET_OLD].filter(Boolean);

export async function POST(request) {
  const rawBody = Buffer.from(await request.arrayBuffer());
  const hmac = request.headers.get("x-shopify-hmac-sha256");
  if (!secrets.some((secret) => verifyShopifyWebhook(rawBody, hmac, secret))) {
    return new Response("invalid hmac", { status: 401 });
  }

  await enqueueWebhook({
    id: request.headers.get("x-shopify-webhook-id"),
    topic: request.headers.get("x-shopify-topic"),
    shop: request.headers.get("x-shopify-shop-domain"),
    body: rawBody.toString("utf8"),
  });
  return new Response(null, { status: 200 });
}
```

If your project has no `@/` alias, use relative paths. The handler uses `Buffer` and `node:crypto`, which are available in the `nodejs` runtime, the Next.js default.

STEP 04

The queue and duplicates

Shopify expects a `200` within five seconds (one second to connect) and counts anything outside the 200 range as an error, 3XX included. On failure it retries 8 times over 4 hours, and the same delivery can arrive more than once. That is why the endpoint only verifies, queues and responds. This is the minimal queue from the tests, `webhook-queue.mjs` (in Next.js, `lib/webhook-queue.js`):

```
// shortcut: in memory and in one process; in production, a table with the webhook id as its primary key.
const seen = new Set();
export const jobs = [];

export async function enqueueWebhook(job) {
  if (seen.has(job.id)) return false;
  seen.add(job.id);
  jobs.push(job);
  return true;
}
```

Deduplicate on `X-Shopify-Webhook-Id`, not `X-Shopify-Event-Id`: with two subscriptions to the same topic, each delivery has its own Webhook-Id and both share the Event-Id. If the ID is already there, the endpoint still returns `200`, as Shopify asks.

STEP 05

Try it locally with curl

Start the server with `SHOPIFY_API_SECRET=test_secret_do_not_use node start.mjs`. In another bash terminal (on Windows, Git Bash, which ships `openssl`), save and run `bash curl.sh`:

```
SECRET=test_secret_do_not_use # placeholder: never paste your real client secret into a script
BODY='{"id": 820982911946154508, "email": "jon@example.com"}'
HMAC=$(printf '%s' "$BODY" | openssl dgst -sha256 -hmac "$SECRET" -binary | base64)
send() { curl -s -o /dev/null -w "$1: %{http_code}" http://localhost:8798/webhooks/shopify -H "content-type:
application/json" -H "x-shopify-webhook-id: $1" -H "x-shopify-hmac-sha256: $2" -d "$3"; echo; }

send wh-1 "$HMAC" "$BODY"
send wh-1 "$HMAC" "$BODY"
send wh-2 "$HMAC" '{"id": 820982911946154508, "email": "eve@example.com"}'
send wh-3 "not-a-signature" "$BODY"
```

This is the output we got. The second `200` is the duplicate: it gets a success response, but the queue does not take it again.

```
wh-1: 200
wh-1: 200
wh-2: 401
wh-3: 401
```

STEP 06

Before you ship it

- **Duplicates in a shared store.** The `Set` lives in one process. With several instances or restarts, store the delivery ID in a table with a primary key, and make the worker that reads the queue idempotent.
- **Client secret rotation.** As long as the old secret is not revoked, Shopify keeps signing with it, and it can take up to an hour to use the new one. Put the new one in `SHOPIFY_API_SECRET`, the old one in `SHOPIFY_API_SECRET_OLD`, and drop the old one only after revoking it.
- **Compliance webhooks.** If you list on the App Store, `customers/data_request`, `customers/redact` and `shop/redact` must verify the HMAC and return `401 Unauthorized` when it is invalid.
- **Body size.** `express.raw()` accepts 100 KB by default; if your payloads can be bigger, raise `limit`.

> **CAUTION**
>
> Do not parse the body before verifying it. A global `app.use(express.json())` ahead of the route, or a `request.json()` in Next.js, leaves you without the original bytes, and `JSON.stringify` does not rebuild them: in our test it dropped spaces and rounded an ID above `Number.MAX_SAFE_INTEGER`, and the signature stopped matching.

WHEN YOU FINISH

Checklist

- [ ] The webhook route gets the raw body: `express.raw()` on that route only, or `arrayBuffer()` in Next.js.
- [ ] You compute HMAC-SHA256 of the raw body with the client secret, base64-encoded.
- [ ] You check the length, then compare with `timingSafeEqual`; with no secret or no header it fails without throwing.
- [ ] Invalid or missing signature: `401`. Valid signature: you queue the job and return `200` within five seconds.
- [ ] You deduplicate on `X-Shopify-Webhook-Id` in a shared store.
- [ ] When rotating the client secret, you accept the old and the new one until the old one is revoked.
- [ ] `curl.sh` prints `200`, `200`, `401` and `401`.