GUÍA PASO A PASO

Verificar el HMAC de los webhooks de Shopify

El código en Node para comprobar que cada webhook viene de Shopify, en Express y en un route handler de Next.js: cuerpo crudo, comparación en tiempo constante, 401 si la firma falla, respuesta rápida y duplicados fuera.

PARA	Apps de Shopify y endpoints de webhooks en Node.js
PROBADO CON	Node 22.22.3 · Express 5.3.0 · curl 8.19 · OpenSSL 3.5.6 · Windows 11
ACTUALIZADA	3 de noviembre de 2026
DOCUMENTACIÓN	shopify.dev/docs/apps/build/webhooks/verify-deliveries
ARTÍCULO	ecomlabs.dev/g/hmac
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	La función de verificación
Paso 02	Express
Paso 03	Route handler de Next.js
Paso 04	La cola y los duplicados
Paso 05	Pruébalo en local con curl
Paso 06	Antes de llevarlo a producción
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Un endpoint que recibe los webhooks de Shopify, calcula el HMAC-SHA256 del cuerpo crudo con el client secret de tu app y lo compara en tiempo constante con el encabezado `X-Shopify-Hmac-Sha256`. Si coincide, encola el trabajo y responde `200`; si no, responde `401` y no procesa nada. Va en dos versiones: Express y un route handler de Next.js.

Lo probamos el 2 de noviembre de 2026 con Node 22.22.3, Express 5.3.0, curl 8.19 y OpenSSL 3.5.6 en Windows 11. El route handler lo probamos llamándolo con un `Request` de Node 22, no en un build de Next.js. `test_secret_do_not_use` es un secreto de prueba: el tuyo va en una variable de entorno.

PASO 01

La función de verificación

Guárdala como `verify.mjs` (en Next.js, como `lib/verify-shopify-webhook.js`). No tiene dependencias:

```
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);
}
```

Tres decisiones que importan. Sin secreto devuelve `false`, porque un HMAC con clave vacía lo puede firmar cualquiera. Si el cuerpo no es `Buffer` ni string, alguien lo parseó antes y la firma ya no se puede comprobar. Y compara el largo antes de `timingSafeEqual`, que lanza un `RangeError` si los largos difieren: sin esa línea, un encabezado corto o ausente da 500 en lugar de 401.

PASO 02

Express

Guarda este archivo como `server.mjs`. `express.raw()` va solo en la ruta del webhook, para que `req.body` sea el `Buffer` original; `express.json()` queda después, para el resto de tu 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());
```

Para arrancarlo, un `start.mjs` que importe `app` y llame a `app.listen(8798)`. Si `enqueueWebhook` falla, Express 5 responde 500 y Shopify reintenta la entrega.

PASO 03

Route handler de Next.js

Crea `app/api/webhooks/shopify/route.js`. Lee el cuerpo una sola vez, como bytes, antes de cualquier `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 });
}
```

Si tu proyecto no tiene el alias `@/`, usa rutas relativas. El handler usa `Buffer` y `node:crypto`, que están en el runtime `nodejs`, el predeterminado de Next.js.

PASO 04

La cola y los duplicados

Shopify espera un `200` en cinco segundos (un segundo para conectar) y cuenta como error todo lo que no esté en el rango 200, incluidas las 3XX. Si falla, reintenta 8 veces en 4 horas, y la misma entrega puede llegar más de una vez. Por eso el endpoint solo verifica, encola y responde. Esta es la cola mínima de las pruebas, `webhook-queue.mjs` (en 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;
}
```

Descarta repetidos por `X-Shopify-Webhook-Id`, no por `X-Shopify-Event-Id`: con dos suscripciones al mismo tema, cada entrega trae su propio Webhook-Id y comparten el Event-Id. Si el id ya está, el endpoint responde `200` igual, como pide Shopify.

PASO 05

Pruébalo en local con curl

Arranca el servidor con `SHOPIFY_API_SECRET=test_secret_do_not_use node start.mjs`. En otra terminal con bash (en Windows, Git Bash, que trae `openssl`), guarda y corre `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"
```

Esta es la salida que obtuvimos. El segundo `200` es el duplicado: responde éxito, pero la cola no lo vuelve a encolar.

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

PASO 06

Antes de llevarlo a producción

- **Duplicados en un almacén compartido.** El `Set` vive en un proceso. Con varias instancias o reinicios, guarda el id de la entrega en una tabla con clave primaria, y haz idempotente el proceso que lee la cola.
- **Rotación del client secret.** Mientras el secreto viejo no esté revocado, Shopify sigue firmando con él, y puede tardar hasta una hora en usar el nuevo. Pon el nuevo en `SHOPIFY_API_SECRET`, el viejo en `SHOPIFY_API_SECRET_OLD`, y quita el viejo solo después de revocarlo.
- **Webhooks de cumplimiento.** Si publicas en la App Store, `customers/data_request`, `customers/redact` y `shop/redact` tienen que verificar el HMAC y devolver `401 Unauthorized` si es inválido.
- **Tamaño del cuerpo.** `express.raw()` acepta 100 KB por defecto; si tus cuerpos pueden ser más grandes, sube `limit`.

> **CUIDADO**
>
> No parsees el cuerpo antes de verificarlo. Un `app.use(express.json())` global antes de la ruta, o un `request.json()` en Next.js, te dejan sin los bytes originales, y `JSON.stringify` no los reconstruye: en nuestra prueba quitó espacios y redondeó un id mayor que `Number.MAX_SAFE_INTEGER`, y la firma dejó de coincidir.

AL TERMINAR

Checklist

- [ ] La ruta del webhook recibe el cuerpo crudo: `express.raw()` solo en esa ruta, o `arrayBuffer()` en Next.js.
- [ ] Calculas HMAC-SHA256 del cuerpo crudo con el client secret, en base64.
- [ ] Comparas el largo y después con `timingSafeEqual`; sin secreto o sin encabezado, falla sin lanzar errores.
- [ ] Firma inválida o ausente: `401`. Firma válida: encolas y respondes `200` en menos de cinco segundos.
- [ ] Descartas duplicados por `X-Shopify-Webhook-Id` en un almacén compartido.
- [ ] Al rotar el client secret, aceptas el viejo y el nuevo hasta revocar el viejo.
- [ ] `curl.sh` da `200`, `200`, `401` y `401`.