



API REFERENCE · API V1

Webhooks

SMSRay POSTs a signed JSON event to your webhook URL whenever a message changes status or a reply arrives.

Updated 2026-10-10 · generated 2026-10-10 · smsray.com/docs/webhooks

Setting up

- In the dashboard, set a webhook URL on your API client. It must use `https://`.
- Copy the signing secret (`whsec_...`) shown when the client was created or the secret was rotated.
- Each event is a POST with `content-type: application/json` and a single event object as the body.
- Respond with any 2xx within 10 seconds. Do slow work after you respond.

Request headers

HEADER	VALUE
<code>x-lacsapace-signature</code>	<code>t=<unix seconds>,v1=<hex></code> — HMAC-SHA256 of <code><t>.<raw body></code> . A second <code>v1=</code> appears during secret rotation.
<code>x-lacsapace-event</code>	<code>message.status</code> or <code>message.inbound</code>
<code>x-lacsapace-delivery</code>	Unique id of this delivery. Retries of the same event reuse it — use it to deduplicate.
<code>user-agent</code>	<code>lacsapace-sms-webhooks/1</code>
<code>content-type</code>	<code>application/json</code>

message.status

Sent when a message you sent reaches `sent`, `delivered`, `undelivered` or `failed`.

message.status

```
{
  "type": "message.status",
  "id": "2b1f6c1e-8d4f-4c55-9a51-0d2b0b7f3a11",
  "status": "delivered",
  "to": "+9779801234567",
  "at": "2026-10-10T04:15:07.000Z",
  "messageType": "transactional",
  "segments": 1
}
```

- Never sent on accept — the API reply already told you **queued**.
- **queued** is sent only when a message goes back to **queued**, for example when delivery is deferred or retried.
- Never sent twice for the same message and status, even if a late duplicate report arrives.
- **error** is included only when the status is **failed**, **undelivered** or **queued** and a reason is known.
- **at** is the time of the transition (delivered, sent or failed time), in ISO 8601 UTC.
- Events can arrive out of order after retries. Compare **at** before overwriting a later status.

Status mapping

The API has more statuses than the webhook. They map like this:

MESSAGE STATUS (GET /SMS/MESSAGES/:ID)	WEBHOOK STATUS
submitted, sent	sent
delivered	delivered
undelivered	undelivered
failed, rejected	failed
anything else	queued

message.inbound

Sent when a recipient replies to a message you sent and the reply is attributed to your API client. **inReplyTo** links it to the message they answered when known.

message.inbound

```
{
  "type": "message.inbound",
  "id": "6705a1c2e4b0f3a9d8c71b25",
  "from": "+9779801234567",
  "text": "Thanks",
  "at": "2026-10-10T04:20:00.000Z",
  "inReplyTo": "2b1f6c1e-8d4f-4c55-9a51-0d2b0b7f3a11"
}
```

Verifying signatures

Always verify before trusting a webhook. Recompute HMAC-SHA256 over `<t>.<raw body>` with your secret, compare in constant time against each **v1**, and reject timestamps more than 300 seconds from your clock. Each retry is re-signed with a fresh **t**, so a slow retry still verifies.

Node.js

```
import crypto from "node:crypto";

// x-lacsapace-signature: t=<unix>,v1=<hex>[,v1=<hex during secret rotation>]
export function verifySmsrayWebhook(rawBody, header, secret, toleranceSec = 300) {
  const pairs = header.split(",").map((p) => p.split("="));
  const t = Number(pairs.find(([k]) => k === "t")?.[1]);
  if (!t || Math.abs(Date.now() / 1000 - t) > toleranceSec) return false;
  const expected = Buffer.from(
```

```

    crypto.createHmac("sha256", secret).update(`${t}.${rawBody}`).digest("hex"), "hex");
return pairs
    .filter(([k]) => k === "v1")
    .some([, v]) => {
        const sig = Buffer.from(v, "hex");
        return sig.length === expected.length && crypto.timingSafeEqual(sig, expected);
    });
}

```

Python

```

import hmac, hashlib, time

def verify_smsray_webhook(raw_body: bytes, header: str, secret: str, tolerance=300) -> bool:
    pairs = [p.split("=", 1) for p in header.split(",")]
    t = next((int(v) for k, v in pairs if k == "t"), 0)
    if not t or abs(time.time() - t) > tolerance:
        return False
    expected = hmac.new(secret.encode(), f"{t}".encode() + raw_body, hashlib.sha256).hexdigest()
    return any(hmac.compare_digest(v, expected) for k, v in pairs if k == "v1")

```

PHP

```

function verify_smsray_webhook(string $rawBody, string $header, string $secret, int $tolerance =
» 300): bool {
    $t = 0;
    $sigs = [];
    foreach (explode(",", $header) as $part) {
        [$k, $v] = array_pad(explode("=", $part, 2), 2, "");
        if ($k === "t") $t = (int) $v;
        if ($k === "v1") $sigs[] = $v;
    }
    if (!$t || abs(time() - $t) > $tolerance) return false;
    $expected = hash_hmac("sha256", $t . "." . $rawBody, $secret);
    foreach ($sigs as $sig) {
        if (hash_equals($expected, $sig)) return true;
    }
    return false;
}

// Use the raw body, before any JSON parsing:
// $raw = file_get_contents("php://input");
// $ok = verify_smsray_webhook($raw, $_SERVER["HTTP_X_LACSPACE_SIGNATURE"] ?? "",
» getenv("SMSRAY_WEBHOOK_SECRET"));

```

Go

```

import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "strconv"
    "strings"
    "time"
)

// VerifySmsrayWebhook checks x-lacsapace-signature against the raw request body.
func VerifySmsrayWebhook(rawBody []byte, header, secret string, tolerance int64) bool {

```

```

var t int64
var sigs []string
for _, part := range strings.Split(header, ",") {
    k, v, _ := strings.Cut(part, "=")
    switch k {
    case "t":
        t, _ = strconv.ParseInt(v, 10, 64)
    case "v1":
        sigs = append(sigs, v)
    }
}
age := time.Now().Unix() - t
if t == 0 || age > tolerance || age < -tolerance {
    return false
}
mac := hmac.New(sha256.New, []byte(secret))
mac.Write([]byte(strconv.FormatInt(t, 10) + "."))
mac.Write(rawBody)
expected := mac.Sum(nil)
for _, s := range sigs {
    got, err := hex.DecodeString(s)
    if err == nil && hmac.Equal(got, expected) {
        return true
    }
}
return false
}

```

USE THE RAW BODY

Verify the exact bytes you received. Parsing and re-serialising JSON changes spacing and key order and the signature will not match.

Node.js (Express)

```

import express from "express";
import { verifySmsrayWebhook } from "../verify.js"; // the function above

const app = express();

// Verify against the RAW body – parse JSON only after the check
app.post("/hooks/sms", express.raw({ type: "application/json" })), async (req, res) => {
    const raw = req.body.toString("utf8");
    const ok = verifySmsrayWebhook(raw, req.get("x-lacsapace-signature") ?? "",
    » process.env.SMSRAY_WEBHOOK_SECRET);
    if (!ok) return res.status(400).send("bad signature");

    res.sendStatus(200); // acknowledge fast, work after

    const event = JSON.parse(raw);
    const deliveryId = req.get("x-lacsapace-delivery"); // dedupe on this
    if (event.type === "message.status") await markStatus(event.id, event.status, event.at);
    if (event.type === "message.inbound") await saveReply(event.from, event.text, event.inReplyTo);
});

```

Python (Flask)

```

from flask import Flask, request

app = Flask(__name__)

@app.post("/hooks/sms")
def sms_hook():
    raw = request.get_data() # raw bytes, before JSON parsing
    if not verify_smsray_webhook(raw, request.headers.get("x-lacsapace-signature", ""), SECRET):
        return "bad signature", 400
    event = request.get_json()
    enqueue(event, delivery_id=request.headers.get("x-lacsapace-delivery")) # work after
    » responding
    return "", 200

```

PHP

```

$raw = file_get_contents("php://input");
$sig = $_SERVER["HTTP_X_LACSPACE_SIGNATURE"] ?? "";
if (!verify_smsray_webhook($raw, $sig, getenv("SMSRAY_WEBHOOK_SECRET"))) {
    http_response_code(400);
    exit("bad signature");
}
http_response_code(200);
$event = json_decode($raw, true);
$deliveryId = $_SERVER["HTTP_X_LACSPACE_DELIVERY"] ?? ""; // dedupe on this
// queue $event for processing

```

Go (net/http)

```

http.HandleFunc("/hooks/sms", func(w http.ResponseWriter, r *http.Request) {
    raw, err := io.ReadAll(r.Body) // raw bytes, before JSON parsing
    if err != nil {
        http.Error(w, "bad body", http.StatusBadRequest)
        return
    }
    if !VerifySmsrayWebhook(raw, r.Header.Get("x-lacsapace-signature"),
    » os.Getenv("SMSRAY_WEBHOOK_SECRET"), 300) {
        http.Error(w, "bad signature", http.StatusBadRequest)
        return
    }
    w.WriteHeader(http.StatusOK) // acknowledge fast, work after
    go handleEvent(raw, r.Header.Get("x-lacsapace-delivery"))
})

```

Secret rotation

Workspace owners can rotate the webhook secret from the dashboard and choose a grace period of 0–168 hours (24 h by default). During the grace period, every delivery carries **two** signatures: one **v1=** made with the new secret and one with the previous secret.

During rotation

```
x-lacsapace-signature: t=1760069707,v1=5f2c...(new secret),v1=a91e...(previous secret)
```

- Because the verifiers above accept any matching **v1**, your handler keeps working with the old secret while

you deploy the new one.

- Deploy the new secret within the grace period. After it ends, only the new signature is sent.
- Choose a grace period of 0 if the old secret may have leaked — it stops working at once.

Retries and timeouts

A delivery that does not get a 2xx within 10 seconds is retried with increasing delays, up to 8 attempts in total.

ATTEMPT	WHEN	WAIT
1	Right after the event	—
2	10 s after attempt 1	10 s
3	30 s after attempt 2	30 s
4	2 min after attempt 3	2 min
5	10 min after attempt 4	10 min
6	30 min after attempt 5	30 min
7	1 h after attempt 6	1 h
8	3 h after attempt 7	3 h

- The first attempt is made right after the event happens.
- Delays grow along the schedule above and are capped at 6 hours.
- After 8 attempts — or 3 attempts if your endpoint answers with a permanent 4xx (anything 400–499 except 408 and 429) — the delivery is marked dead.
- Delivery history is kept for 7 days.
- The event body never changes between attempts; only the signature timestamp does.

Handler checklist

- Verify the signature on the raw body and reject old timestamps.
- Return 2xx quickly; queue heavy work.
- Deduplicate on `x-lacspace-delivery`, and treat status updates as idempotent.
- Return 5xx (or time out) for temporary problems so we retry; return 2xx for events you choose to ignore.

The web page is always the current version: smsray.com/docs/webhooks

Full API reference PDF: smsray.com/docs/pdf · Your keys and live examples: app.smsray.com/developers