

ScoreSay booking integration

API version: 2026-10-01 · **Audience:** developers of the club's booking system

This integration connects a club's booking system to the ScoreSay court tablets:

1. **You send us bookings:** court, start and end time, player names. Send every change and cancellation as it happens.
2. **The court tablet shows the names on time.** From 10 minutes before the booking starts, the tablet on that court shows a card with the players and the booking time. Players press **START MATCH** and keep score on the tablet.
3. **We send you the result.** When a match finishes, we `POST` a signed `match.finished` event to your webhook URL: winner and set scores.

Instead of sending bookings to the API you can have us **read them from your own service**: see [10. Pull mode](#).

The integration is the same for every club and every booking system. A booking system that serves several clubs uses a separate API key for each club.

1. Setup (one time per club)

The club does this in the ScoreSay club panel (skortabelasi.com/panel/ → **Integration** tab). The club manager or owner gives you:

What	Where it comes from	What you do with it
API base URL	Panel, top of the Integration tab	Prefix for every request
API key (<code>ssk_live_...</code>)	Panel → <i>API key</i> → <i>Generate key</i> (shown only once)	Send it as <code>Authorization: Bearer <key></code>
Court numbers	Panel → <i>Court numbers</i> , or <code>GET /v1/club</code>	Map your court IDs to these numbers
Webhook signing secret (<code>whsec_...</code>)	Panel → <i>Result URL</i> → <i>Show signing secret</i>	Verify the <code>ScoreSay-Signature</code> header

You give the club your **webhook URL** (https). The manager enters it in the panel. Clicking **Send test event** sends a sample result to your URL immediately.

The API base URL is `https://scoresay.com/api/club/v1`. The same path also works on `https://skortabelasi.com/api/club/v1`.

Checklist

- ☐ `GET /v1/club` returns the club and its courts (the key works).
- ☐ Your court IDs are mapped to ScoreSay court numbers.
- ☐ Creating, changing and cancelling a booking in your system calls `POST /v1/bookings`.
- ☐ Your webhook verifies the signature, answers `2xx` within 10 seconds and ignores duplicate event IDs.
- ☐ The test event from the panel (or `POST /v1/webhooks/test`) arrives and verifies.

Recommended setup

For a reliable and secure integration we recommend:

1. **Send teams:** when you know who plays with whom, use `teams` (`[[A1, A2], [B1, B2]]`). Send names as "First Last" or "First L."; add your member ID (`id`) to every player.

2. **Send the full state on every change:** when a booking is created, changed (court, time, players) or cancelled, send the **whole** booking with `POST /v1/bookings`. Include the booking's last-modified time in your system as `updatedAt`, so an older request arriving late is ignored automatically. If a request fails (network error, `5xx`, `429`), retry with the same body.
3. **Receive results by webhook, complete them with the event feed:** the signed https webhook is the primary path (arrives about 2 minutes after the match ends). Also call `GET /v1/events?after=<last cursor>` once an hour, so no result is lost even if your endpoint is unreachable for a while. De-duplicate events arriving both ways by `id`.
4. **Initial load:** when going live, load the upcoming days' bookings once with `POST /v1/bookings/batch`.

2. Conventions

- **Authentication:** every request carries `Authorization: Bearer ssk_live_...`. The key belongs to one club. Up to two keys can be active at once, so a key can be rotated without downtime.
- **Format:** JSON in and out, UTF-8. Successful responses have `"ok": true`.
- **Errors:** `{"ok": false, "error": "<code>", ...details}` with an HTTP status (see [Errors](#)).
- **Times:** ISO 8601 with an explicit offset, for example `2026-10-07T18:00:00+03:00` or `2026-10-07T15:00:00Z`. Times without an offset are rejected (`bad_time`), because the time zone would be ambiguous. Responses echo times in the offset you sent.
- **Rate limit:** 120 requests per minute per key. More gives `429 rate_limited` with `retryAfter` (seconds). Repeated failed authentication from one IP is also limited.
- **Unknown fields** in our responses and events may appear in later versions. Ignore fields you do not know.

3. Bookings

3.1 Create or update — `POST /v1/bookings`

One endpoint for create, update and cancel: always send the **full current state** of the booking. The booking is identified by **your** `id`. Sending the same state again is harmless (`"result": "unchanged"`).

```
POST /api/club/v1/bookings
Authorization: Bearer ssk_live_...
Content-Type: application/json

{
  "id": "R-2026-10-07-0042",
  "court": 2,
  "startsAt": "2026-10-07T18:00:00+03:00",
  "endsAt": "2026-10-07T19:30:00+03:00",
  "status": "confirmed",
  "label": "Sunday league",
  "teams": [
    [ { "name": "Ali Kaya", "id": "member-118" }, { "name": "Veli Demir", "id": "member-204" } ],
    [ { "name": "Can Yıldız", "id": "member-77" }, { "name": "Ece Ak" } ]
  ],
  "updatedAt": "2026-10-07T11:12:40+03:00"
}
```

Field	Required	Description
<code>id</code>	yes	Your booking ID. String <code>[A-Za-z0-9._:-]</code> , 1-100 characters (a number is also accepted). Unique per club.
<code>court</code>	yes	ScoreSay court number (see <code>GET /v1/club</code>).

Field	Required	Description
startsAt , endsAt	yes	ISO 8601 with offset. End after start; at most 12 hours; start between 7 days ago and 180 days ahead.
status	no	confirmed (default) or cancelled .
teams	no	Two sides, each with 0–2 players: [[player, player], [player, player]] . The response shape [{ "side": "A", "players": [...] }, ...] is also accepted.
players	no	Instead of teams : a flat list of up to 4 players when you do not know the sides. 2 players → 1 vs 1; 3 → first two vs third; 4 → first two vs last two. Players can rearrange names on the tablet before starting.
label	no	Short text shown on the card (up to 60 characters), for example <i>Lesson</i> or <i>League</i> .
sport	no	padel , tennis , pickleball or padbol . The court's configured sport takes priority; this is used only for a court without one.
version	no	Increasing integer (or updatedAt , ISO with offset). If a request arrives with an older version than the one stored, it is ignored ("result": "stale"). This protects against retries arriving out of order. Without either field the last write wins.

A **player** is { "name": "Ali Kaya", "id": "member-118" } or just the name string. name is 1–40 characters. id is optional (your member ID, up to 100 characters). It is returned unchanged in booking responses.

Names on the tablet: the card shows up to 24 characters per name. Longer names are shortened to first name + initial ("Ahmet Mehmet Yılmazoğlu" → "Ahmet Y."). Two identical names in one booking become "Ali" and "Ali 2". Send display names your players recognise; full legal names are not needed.

Response

Status	result	Meaning
201	created	New booking.
200	updated	Something changed.
200	unchanged	Same state as stored.
200	stale	Older version / updatedAt than stored; ignored. The stored booking is returned.

```
{
  "ok": true,
  "result": "created",
  "booking": {
    "id": "R-2026-10-07-0042",
    "court": 2,
    "startsAt": "2026-10-07T18:00:00+03:00",
    "endsAt": "2026-10-07T19:30:00+03:00",
    "status": "confirmed",
    "sport": null,
    "label": "Sunday league",
    "teams": [
      { "side": "A", "players": [ { "name": "Ali Kaya", "id": "member-118" }, { "name": "Veli Demir", "id": "member-204" } ] },
      { "side": "B", "players": [ { "name": "Can Yıldız", "id": "member-77" }, { "name": "Ece Ak", "id": null } ] }
    ],
    "version": null,
    "createdAt": "2026-10-07T08:12:40.118Z",
    "updatedAt": "2026-10-07T08:12:40.118Z"
  }
}
```

If the court is outside the club's current ScoreSay plan, the booking is still stored and the response contains `"warnings": ["court_locked"]`. The tablet on that court cannot start matches until the club's plan covers it.

3.2 Cancel — DELETE /v1/bookings/{id}

Same as sending `"status": "cancelled"`. Idempotent: cancelling twice returns `"result": "unchanged"`; an unknown ID returns 200 with `"existed": false`. The card disappears from the tablet within seconds.

3.3 Initial load — POST /v1/bookings/batch

`{ "bookings": [ ...up to 200 bookings... ] }`. Each item is processed like `POST /v1/bookings`; an invalid item does not stop the others.

```
{ "ok": true, "results": [
  { "id": "R-1", "ok": true, "result": "created" },
  { "id": "R-2", "ok": false, "error": "unknown_court", "court": 7 }
] }
```

3.4 Read — GET /v1/bookings and GET /v1/bookings/{id}

- `GET /v1/bookings?from=...&to=...&court=2&status=confirmed`: bookings overlapping the range. `from` / `to` are ISO with offset; the default is the last 24 hours to 7 days ahead; the range is at most 31 days.
- `GET /v1/bookings/{id}`: the booking plus `matches`, the result sent to you in the same shape as the webhook's `data.match` (empty if no result has been sent yet). Use this to reconcile if you missed a webhook.

3.5 Club and courts — GET /v1/club

```
{
  "ok": true,
  "apiVersion": "2026-10-01",
  "club": { "id": "cl_...", "slug": "sports-international", "name": "Sports International" },
  "courts": [
    { "court": 1, "name": "Glass Court", "sport": "padel", "locked": false },
    { "court": 2, "name": "Center Court", "sport": "tennis", "locked": false }
  ],
  "webhook": { "url": "https://booking.example.com/scoresay/webhook", "enabled": true }
}
```

Use `club.id` to identify the club; the `slug` can change.

4. What happens on the court

- The tablet receives its own court's bookings for the next 24 hours and keeps them, so the card also works without internet.
- ****From 10 minutes before `startsAt` until `endsAt` **, when no match or tournament occupies the court, the tablet shows the booking card. If two bookings overlap (back-to-back slots), the later one is shown.**
- **START MATCH** opens the match setup with the names filled in. Players can change names and rules (number of sets, golden point, ...) and start.
- A match started from the card is linked to the booking. A match started on that court in the booking's time window without the card (from a phone via the court's QR code, or on an older tablet app) is linked by time.
- A result is sent **once** per booking: about 2 minutes after the match ends, in its final state. Players can correct the score on the tablet during that time; the corrected score is what you receive.

- Corrections made after the result was sent, and other matches played later in the same booking, are not sent.
- If nobody plays, nothing is sent.
- A result is sent when the match ends by its rules or the players press **End match**. A match that is left without being ended (**Cancel match** on the tablet, or a new match started over the open one) is not sent; the same goes for players who do not want to keep score.
- Changes and cancellations reach the tablet within seconds (at most one minute).
- Tournaments organised in ScoreSay take priority: while a tournament holds the court, the card is not shown.

5. Results — webhook

When a match linked to a booking finishes, we `POST` an event to your webhook URL about 2 minutes later. A result is sent once per booking.

5.1 Request

```
POST /scoresay/webhook
Content-Type: application/json
User-Agent: ScoreSay-Webhooks/1
ScoreSay-Event-Id: evt_3J9xkq2Lw8Q
ScoreSay-Event-Type: match.finished
ScoreSay-Signature: t=1791399000,v1=5b1f0d...e2a9

{
  "id": "evt_3J9xkq2Lw8Q",
  "type": "match.finished",
  "test": false,
  "data": {
    "booking": { "id": "R-2026-10-07-0042" },
    "match": {
      "id": "m_Q2b8Xk1dPzE",
      "revision": 1,
      "court": 2,
      "winner": "A",
      "score": {
        "sets": [ { "a": 6, "b": 4 }, { "a": 6, "b": 7 }, { "a": 10, "b": 8 } ],
        "setsWon": { "a": 2, "b": 1 }
      }
    }
  }
}
```

The result is kept minimal on purpose; fields may be added later. Ignore fields you do not know.

Field	Description
id	Event ID (= <code>ScoreSay-Event-Id</code>). The same event may arrive more than once; de-duplicate by it.
type	Only <code>match.finished</code> for now.
test	<code>true</code> for a test event sent from the panel or with <code>POST /v1/webhooks/test</code> ; do not store it.
data.booking.id	Your booking ID. <code>data.booking</code> is <code>null</code> for a test event sent without a booking.
data.match.id	ScoreSay match ID.

Field	Description
<code>data.match.revision</code>	Always 1 for now. If a result for the same booking is ever sent again it increases; the highest one wins.
<code>data.match.court</code>	The court the match was played on (ScoreSay court number).
<code>data.match.winner</code>	"A", "B" or <code>null</code> (no winner). Only completed sets decide the winner: when the players press End match with the sets level, or before any set is completed, it is <code>null</code> ; games in an unfinished set do not count (3-6, 6-0 is 1-1, no winner). Side A is always the first team of the booking : even if the players swap the teams on the tablet, the score and <code>winner</code> follow your order.
<code>data.match.score.sets</code>	One entry per set: <code>a</code> is side A's score, <code>b</code> side B's. Games; points in a match tie-break and in rally-scored sports (pickleball, padbol). If the players pressed End match in the middle of a set, the last set is sent as it stood.
<code>data.match.score.setswon</code>	Sets won by side A and B.

5.2 Verifying the signature

ScoreSay-Signature: `t=<unix seconds>,v1=<hex>[,v1=<hex>]`

1. Take the **raw request body** exactly as received (do not re-serialise the JSON).
2. Compute `HMAC-SHA256(key = signing secret, message = t + "." + rawBody)` as lowercase hex.
3. Accept if it equals **any** `v1` value (constant-time comparison).
4. Reject if `t` is more than 5 minutes away from your clock (replay protection).

When the club rotates the signing secret, for 24 hours each event carries two `v1` values: one with the new secret, one with the previous one. Examples in Node.js, PHP and C# are in [[examples/](#)](examples/).

5.3 Delivery rules

- Answer with any `2xx` within **10 seconds**. Do the heavy work after answering.
- Anything else (non-`2xx`, timeout, connection error, redirect) is a failure. **Redirects are not followed**.
- Failed events are retried after 30 s, 2 min, 10 min, 30 min, 1 h, 2 h, 4 h, 8 h, 12 h, 24 h and 24 h (12 attempts over about 3 days). Then the event is marked failed; the club can resend it from the panel.
- Delivery is **at least once**: the same event can arrive more than once. De-duplicate by `ScoreSay-Event-Id` (`= id`).
- The webhook URL must be `https` and reachable from the internet.

5.4 Without a webhook — GET /v1/events

If you cannot receive webhooks, or to fill gaps, read the same events from the API:

```
GET /v1/events?after=<cursor>&limit=100
→ { "ok": true, "events": [ { "cursor": "41", ...same JSON as the webhook body... } ], "next": "57" }
```

Store `next` and pass it as `after` on the next call. Events are kept for 30 days.

5.5 Test event — POST /v1/webhooks/test

Sends a sample `match.finished` with `"test": true` to the configured webhook and waits for the attempt: `{ "ok": true, "eventId": "evt_...", "delivered": true, "status": 200 }`. With `{ "bookingId": "<your booking id>" }` the sample uses that booking's ID and court. Test events are not retried. At most 10 per hour.

6. Edge cases

Situation	Behaviour
The booking arrives after the match was played	The match is linked when the booking arrives (within 8 days) and the result is sent.
The booking is moved to another court or time	The tablets update within seconds. Matches already linked stay linked.
Back-to-back bookings on one court	Each match belongs to the booking whose window contains its start; when both do (the 10-minute early window), the later booking wins.
Players arrive early	The card opens 10 minutes before <code>startsAt</code> ; a match started in that window belongs to the booking.
The court has no internet	The card works from the stored list; the result is uploaded and sent when the connection returns (up to 7 days later).
Matches that finished before the integration was set up	Not sent.
Match left without being ended	Not sent; it does not appear in <code>GET /v1/bookings/{id}</code> either.

7. Errors

HTTP	error	Meaning
400	<code>bad_json</code>	The body is not valid JSON.
400	<code>bad_request</code>	Missing or malformed field (see <code>field</code>).
400	<code>bad_id</code>	<code>id</code> missing or not <code>[A-Za-z0-9._:-]{1,100}</code> .
400	<code>bad_time</code>	Time without offset, end before start, longer than 12 hours, or out of range (see <code>field</code> , <code>reason</code>).
400	<code>bad_players</code>	More than 2 per side, more than 4 players, name empty or over 40 characters, both <code>teams</code> and <code>players</code> sent.
400	<code>bad_status</code>	<code>status</code> is not <code>confirmed</code> / <code>cancelled</code> .
400	<code>bad_sport</code>	Unknown <code>sport</code> .
401	<code>unauthorized</code>	Missing, wrong or revoked key.
404	<code>not_found</code>	Unknown booking (<code>GET /v1/bookings/{id}</code> , test with <code>bookingId</code>).
409	<code>too_many_bookings</code>	More than 5,000 future bookings for the club.
409	<code>no_webhook</code>	Test event requested but no webhook URL is set.
422	<code>unknown_court</code>	The club has no court with this number.
429	<code>rate_limited</code>	Too many requests; retry after <code>retryAfter</code> seconds.

`5xx` and network errors are safe to retry: every write in this API is idempotent.

8. Privacy (KVKK / GDPR)

- The club is the data controller for its booking data; ScoreSay processes player names only to show them on the club's court tablet and to return results to the club's booking system.
- Send display names and optional member IDs only. Phone numbers, e-mail addresses and other personal data are not needed and must not be sent.

- Bookings are deleted 30 days after they end; events after 30 days. Match results stay in the club's match history in ScoreSay, as for any match played on the club's courts.
- Booking names are not published on the club's public screens before play; they only go to the tablet of the booked court and to the phone of a player who scans the QR code on that tablet to start the match from their own phone. Once players start a match, it appears on the club's screens like any other match.

9. Versioning

The API is versioned in the path (`/v1`). Within `v1` we only add fields, endpoints and event types; we do not remove or rename them. Ignore unknown fields and event types.

10. Pull mode

Instead of sending bookings with `POST /v1/bookings`, you can have us read them from your own service. The club enters your service's address and key in the panel, **Integration → Pull from your booking system** (format: *Standard*). Results arrive as in [section 5](#); `data.booking.id` is your booking ID.

10.1 Our request

```
GET https://booking.example.com/scoresay/bookings?from=2026-10-09T21:00:00Z&to=2026-10-11T21:00:00Z
Authorization: Bearer <per-club key>
Accept: application/json
User-Agent: ScoreSay-BookingPull/1
```

- You choose the address; it may contain your own parameters (for example `?club=1`). The key cannot be in the address: it travels in the `Authorization` header.
- `from` and `to` are UTC (ending in `z`): from 00:00 of the club's today to 48 hours later. You may ignore both and return the bookings of the next 48 hours; we drop anything outside the range.
- We wait 10 seconds for the response; it can be at most 256 KB. Redirects are not followed.

10.2 Your response

200, `application/json`, UTF-8. The booking object is the same as in `POST /v1/bookings`.

```
{
  "bookings": [
    {
      "id": "R-2026-10-10-0042",
      "court": 1,
      "startsAt": "2026-10-10T23:00:00+03:00",
      "endsAt": "2026-10-11T00:00:00+03:00",
      "status": "confirmed",
      "teams": [
        [ { "name": "Ali Kaya", "id": "member-118" }, { "name": "Veli Demir", "id": "member-204" } ],
        [ { "name": "Can Yıldız", "id": "member-77" }, { "name": "Ece Ak" } ]
      ]
    }
  ]
}
```

Field	Required	Description
<code>id</code>	yes	Your booking ID (string or number; letters, digits, <code>.</code> , <code>_</code> , <code>-</code> ; at most 64 characters). Returned unchanged as <code>data.booking.id</code> in results.
<code>court</code>	yes	Your court number (1–99). The club maps it to its own court in the panel; no mapping is needed when they are the same.

Field	Required	Description
startsAt	yes	ISO 8601 with offset. A time without an offset is rejected.
endsAt	no	If missing, the match length from the club's panel is used (60 minutes by default).
status	no	confirmed (default) or cancelled.
teams	yes	Two teams with at most 2 players each: { "name": ..., "id": ... } or just the name. The first team is A, the second is B. id is optional (at most 100 characters).
label , sport	no	As in POST /v1/bookings.

- **The card appears when the teams are complete:** both teams filled and equal (1 vs 1 or 2 vs 2); 2 vs 2 on padel and padbol courts. A booking with missing players may stay in the list; when the names are in, return its current state with the same id.
- **Cancellation:** send status: "cancelled" or remove the booking from the list. A booking that has not started and is missing from the list counts as cancelled.
- **Changes:** if the court, time or players change, return the new state with the same id; the card updates by itself.
- **Errors:** on a non-2xx response or a timeout we leave the existing cards alone and ask again 5 minutes later.
- **Names:** a name written entirely in capitals is fixed; on the tablet a name is shown as "First L".

10.3 How often we ask

When	How often
Regularly	Every 15 minutes
Before each match	Once, 5 minutes before the start
Booking with missing players	From 5 minutes before the start until 10 minutes after it, once a minute
When a match ends on the court	Once, immediately (for the next match's card)
After an error	We wait 5 minutes

11. Security summary

Topic	Rule
Transport	HTTPS (443) only, in both directions. Plain http and redirects are not accepted.
Our server's IP address	Our pull requests and result deliveries come from one fixed address: 147.93.84.183. Allow it in your firewall or rate limiter.
API key (ssk_live_...)	Belongs to one club; only its hash is stored; at most two keys are active at a time (rotation without downtime).
Pull key	Should be unique per club; at least 16 characters are accepted, we recommend 32 random characters. It travels in the Authorization header (a key in the address ends up in server logs). It is only sent to the domain it was saved for.
Result signature	Every result request is signed (5.2). Do not process a request whose signature does not match.
Replay	Reject a signature whose t is older than 5 minutes; do not process the same event id twice.
Personal data	Display name and optional member ID only (section 8).

Support: info@scoresay.com

Appendix — Signature verification examples

Node.js · examples/verify-signature.mjs

```
// ScoreSay webhook signature check – Node.js (no dependencies).
//
// Header: ScoreSay-Signature: t=<unix seconds>,v1=<hex>[,v1=<hex>]
// v1 = HMAC-SHA256(signing secret, t + "." + raw request body), lowercase hex.
// During a secret rotation the header carries two v1 values; accept if any matches.
import crypto from 'node:crypto';

/**
 * @param {string} header      value of the ScoreSay-Signature header
 * @param {string|Buffer} rawBody the request body exactly as received (do not re-serialise the JSON)
 * @param {string} secret      the club's signing secret (whsec_...)
 * @param {number} [toleranceSec=300] maximum clock difference (replay protection)
 * @param {number} [nowSec]      current unix time in seconds (for tests)
 * @returns {boolean}
 */
export function verifyScoreSaySignature(header, rawBody, secret, toleranceSec = 300, nowSec =
Math.floor(Date.now() / 1000)) {
  if (typeof header !== 'string' || !secret) return false;
  let t = null;
  const sigs = [];
  for (const part of header.split(',')) {
    const [k, v] = part.split('=', 2).map((s) => (s || '').trim());
    if (k === 't' && /^d+$/.test(v)) t = Number(v);
    else if (k === 'v1' && /^[0-9a-f]{64}$/.test(v)) sigs.push(v);
  }
  if (t === null || !sigs.length) return false;
  if (Math.abs(nowSec - t) > toleranceSec) return false;
  const body = Buffer.isBuffer(rawBody) ? rawBody.toString('utf8') : String(rawBody);
  const expected = Buffer.from(crypto.createHmac('sha256', secret).update(t + '.' + body).digest('hex'), 'hex');
  return sigs.some((s) => {
    const got = Buffer.from(s, 'hex');
    return got.length === expected.length && crypto.timingSafeEqual(got, expected);
  });
}
```

```
<?php
// ScoreSay webhook signature check – PHP 7.3+ (no dependencies).
//
// Header: ScoreSay-Signature: t=<unix seconds>,v1=<hex>[,v1=<hex>]
// v1 = HMAC-SHA256(signing secret, t + "." + raw request body), lowercase hex.
// During a secret rotation the header carries two v1 values; accept if any matches.

function scoresay_verify_signature(string $header, string $rawBody, string $secret, int $toleranceSec = 300):
bool
{
    $t = null;
    $sigs = [];
    foreach (explode(',', $header) as $part) {
        $kv = explode('=', $part, 2);
        if (count($kv) !== 2) continue;
        $k = trim($kv[0]);
        $v = trim($kv[1]);
        if ($k === 't' && ctype_digit($v)) $t = (int) $v;
        elseif ($k === 'v1' && preg_match('/^[0-9a-f]{64}$/', $v)) $sigs[] = $v;
    }
    if ($t === null || !$sigs || $secret === '') return false;
    if (abs(time() - $t) > $toleranceSec) return false;
    $expected = hash_hmac('sha256', $t . '.' . $rawBody, $secret);
    foreach ($sigs as $s) {
        if (hash_equals($expected, $s)) return true;
    }
    return false;
}

// Usage (plain PHP endpoint):
//
// $raw = file_get_contents('php://input');           // raw body – do not json_decode first
// $sig = $_SERVER['HTTP_SCORESAY_SIGNATURE'] ?? '';
// if (!scoresay_verify_signature($sig, $raw, getenv('SCORESAY_WEBHOOK_SECRET'))) {
//     http_response_code(400);
//     exit;
// }
// $event = json_decode($raw, true);
// // de-duplicate by $event['id'], store the result keyed by $event['data']['match']['id']
// // (keep the highest $event['data']['match']['revision'])
// http_response_code(200);
```

C# / .NET · examples/VerifySignature.cs

```
// ScoreSay webhook signature check - C# / .NET 6+ (no dependencies).
//
// Header: ScoreSay-Signature: t=<unix seconds>,v1=<hex>[,v1=<hex>]
// v1 = HMAC-SHA256(signing secret, t + "." + raw request body), lowercase hex.
// During a secret rotation the header carries two v1 values; accept if any matches.
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;

public static class ScoreSaySignature
{
    public static bool Verify(string header, string rawBody, string secret, int toleranceSec = 300)
    {
        if (string.IsNullOrEmpty(header) || string.IsNullOrEmpty(secret)) return false;
        long? t = null;
        var sigs = new List<string>();
        foreach (var part in header.Split(','))
        {
            var i = part.IndexOf('=');
            if (i < 0) continue;
            var k = part.Substring(0, i).Trim();
            var v = part.Substring(i + 1).Trim();
            if (k == "t" && long.TryParse(v, out var ts)) t = ts;
            else if (k == "v1" && v.Length == 64) sigs.Add(v);
        }
        if (t == null || sigs.Count == 0) return false;
        if (Math.Abs(DateTimeOffset.UtcNow.ToUnixTimeSeconds() - t.Value) > toleranceSec) return false;

        using var hmac = new HMACSHA256(Encoding.UTF8.GetBytes(secret));
        var expected = hmac.ComputeHash(Encoding.UTF8.GetBytes(t.Value + "." + rawBody));
        foreach (var s in sigs)
        {
            byte[] got;
            try { got = Convert.FromHexString(s); } catch (FormatException) { continue; }
            if (CryptographicOperations.FixedTimeEquals(got, expected)) return true;
        }
        return false;
    }
}

// ASP.NET Core minimal API usage:
//
// app.MapPost("/scoresay/webhook", async (HttpRequest req) => {
//     using var reader = new StreamReader(req.Body, Encoding.UTF8);
//     var raw = await reader.ReadToEndAsync(); // raw body - do not deserialize first
//     var sig = req.Headers["ScoreSay-Signature"].ToString();
//     if (!ScoreSaySignature.Verify(sig, raw, Environment.GetEnvironmentVariable("SCORESAY_WEBHOOK_SECRET")))
//         return Results.BadRequest();
//     // de-duplicate by event id; store the result keyed by data.match.id (keep the highest revision)
//     return Results.Ok();
// });
```

Machine-readable definition: openapi.yaml (same folder).