

Webhooks in Relatel

Relatel · August 2026

Webhooks let your own systems know the moment something happens in Relatel. A call starts, a message arrives, a contact is updated. Instead of polling us on a timer, we push to you as the event happens.

You set them up yourself under **Company administration** → **Webhooks**. There is no developer agreement to sign, and nothing to configure on Relatel's side.

Contents

1. The short version
2. The overview
3. Create a webhook
4. The signing key
5. Verifying what you receive
6. Following your deliveries
7. When a webhook stops by itself
8. Rotating the key
9. Pausing
10. Current limits
11. Questions

The short version

1. You have an HTTPS address that accepts `POST` with JSON.

2. You create a webhook, choose which resource it listens to, and which events you want.
 3. You save the signing key. It is shown **once only**.
 4. You send a test delivery and confirm your system answers `2xx`.
 5. From then on you can follow every delivery, its response time and its status.
-

The overview

The screenshots in this guide come from a fictional example company, Tandklinikken Eksempel, a dental clinic. They have created one webhook per system that needs to hear about events: **Booking**, **Journalsystem** (their patient records), **Patient-SMS**, **Patientkartotek** (their patient directory), **Regnskab** (accounting), **Vagttelefon** (out-of-hours line) and **Ledelsesrapport** (management reporting).

Webhooks

Send events from Relatel to your own systems in real time.

NAME	SCOPE	EVENTS	STATE
Booking https://hooks.tandklinik.example/relatel/1	Main number	1	Active
Journalsystem https://hooks.tandklinik.example/relatel/2	Main number	1	Active
Ledelsesrapport https://hooks.tandklinik.example/relatel/4	Main number	1	Active
Patient-SMS https://hooks.tandklinik.example/relatel/7	Main number	2	Active
Patientkartotek https://hooks.tandklinik.example/relatel/5	Main number	2	Active
Regnskab https://hooks.tandklinik.example/relatel/6	Main number	2	Paused
Vagttelefon https://hooks.tandklinik.example/relatel/3	Main number	1	Stopped

Each row is one webhook: its name, the address we post to, the resource it listens to, its state, and when we last delivered.

In the example, **Regnskab** was paused by the clinic itself, and **Vagttelefon** was stopped automatically because its endpoint kept failing. The rest are delivering normally.

State means:

State	Meaning
Active	We are sending events.
Paused	You paused it. We don't send, and we don't store the events.
Stopped	We stopped it automatically because your endpoint kept failing.

Create a webhook

Basics

Name

A short label so you can tell your webhooks apart.

Description

Endpoint URL

Must start with https://. We send events to this address.

Name is only for you, so you can tell your webhooks apart.

Endpoint URL must start with `https://`. We don't post to `http://`, and we don't post to addresses on private networks. We check that every time we send, not just when you save.

Scope is the important choice. A webhook listens to exactly one resource, and you only get events from that one:

Resource	Events
A main number	<code>call.created</code> , <code>call.ended</code> , <code>incoming_message.created</code>
A chat	<code>chat.new</code> , <code>chat.answered</code> , <code>chat.ended</code>
Contacts (company-wide)	<code>contact.created</code> , <code>contact.updated</code> , <code>contact.destroyed</code>

If you want calls from two main numbers, create two webhooks. There is no “all main numbers” option. That is deliberate, so a webhook can never start sending more than you asked for.

The event list changes when you pick a resource, so you only see events that exist for it.

Scope & events

Resource

Hovednummer ▾

The webhook only receives events for the resource you pick.

Events

Main number

☐ `call.created`

☐ `call.ended`

☐ `incoming_message.created`

ADVANCED

Advanced

Timeout (seconds)

Max retry attempts

Custom headers

```
{"Authorization": "Bearer abc123"}
```

Timeout is how long we wait for your response. Default and maximum are 30 seconds. Not answering in time counts as a failure.

Max attempts is how many times we try before giving up. Default and maximum are 8.

Extra headers is for receivers that require something like an `Authorization` header. Give them as JSON:

```
{"Authorization": "Bearer abc123"}
```

They are encrypted at rest and kept out of our request logs.

The signing key

Below, **Patientkartotek** has just been created, so its key is shown for the first and only time.

Patientkartotek

`https://hooks.tandklinik.example/relatel/5` Active

Success rate (24h)
—

Deliveries (24h)
0

Typical response
—

Slowest 5%
—

Failures in a row
0

Send test delivery Pause Edit Delete

Recent deliveries

Refresh

No deliveries yet. When events fire on this scope, they'll appear here.

Signing secret

Copy your signing key

This is the only time you will see it. Store it with your webhook receiver before leaving this page.

59vu4JCdJb6NUPNUKytkKD7oP6ouqZeHivRrWdFXEJFx2maJtRp8wsyWHbeN3tTH

Copy

If you lose it, rotate the key to get a new one.

Signing key

22c4f217

Rotate key

Creating a webhook gives you a signing key. **It is shown this one time only.** Copy it into your receiving system before you leave the page.

Afterwards we only show a *fingerprint* of the key, meaning the first eight characters of a SHA-256 of it, for example `22c4f217`. That is enough to see which key is in use, but the fingerprint cannot be used to derive the key. So it is safe in an email or a support ticket.

If you lose the key you cannot retrieve it. Rotate it instead and get a new one.

Verifying what you receive

Every delivery carries these headers:

```
Content-Type: application/json
Relatel-Delivery-Id: 8f2e1c4a-6b03-4a7d-9e11-5c2f0a3b8d47
Relatel-Event-Type: call.created
Relatel-Timestamp: 1785062400
Relatel-Signature: t=1785062400,v1=3b8f2c91d4...
```

`Relatel-Timestamp` is Unix time in whole seconds, so the integer `1785062400`, not an ISO date string.

`Relatel-Delivery-Id` is the same across every attempt of one event. Use it to recognise something you have already handled.

The body looks like this:

```
{
  "delivery_id": "8f2e1c4a-6b03-4a7d-9e11-5c2f0a3b8d47",
  "event_type": "call.created",
  "occurred_at": "2026-08-07T09:20:00Z",
  "payload": { }
}
```

VERIFY THE SIGNATURE

`v1` is an HMAC-SHA256 over a string you assemble yourself: the timestamp, a full stop, and the raw body exactly as you received it. With the values above, the string being signed is:

```
1785062400>{"delivery_id":"8f2e1c4a-6b03-4a7d-9e11-
5c2f0a3b8d47","event_type":"call.created", ...}
```

The approach is the same in any language: read the raw body, build that string, compute an HMAC-SHA256 with your signing key, and compare against `v1`. Below it is shown in Ruby and in JavaScript.

In **Ruby**, where `openssl` is in the standard library. In a Rails controller the raw body is `request.raw_post`, and the header is `request.headers['Relatel-Signature']`:

```
require 'openssl'

def verify(raw_body, header, signing_key)
  timestamp = header[/t=(\d+)/, 1]
  return false if timestamp.nil?

  # Reject stale deliveries. We recommend 5 minutes at most.
  return false if (Time.now.to_i - timestamp.to_i).abs > 300

  expected = OpenSSL::HMAC.hexdigest(
    'sha256', signing_key, "#{timestamp}.#{raw_body}"
  )

  signatures = header.split(',')
    .select { |part| part.start_with?('v1=') }
    .map { |part| part.delete_prefix('v1=') }

  # There may be several v1= segments. Accept if any one of them matches.
  signatures.any? do |candidate|
    candidate.bytesize == expected.bytesize &&
      OpenSSL.fixed_length_secure_compare(candidate, expected)
  end
end
```

The same in **JavaScript for Node.js**, where `crypto` is built in:

```
// Node.js. rawBody is the request body as text, not as a parsed object.
// header is the value of Relatel-Signature, for example
// "t=1785062400,v1=3b8f2c91d4...,v1=91acdd0e77..."
const crypto = require('crypto')

function verify(rawBody, header, signingKey) {
  const timestamp = header.match(/t=(\d+)/)[1]

  // Reject stale deliveries. We recommend 5 minutes at most.
  if (Math.abs(Date.now() / 1000 - Number(timestamp)) > 300) {
    return false
  }

  const expected = crypto
    .createHmac('sha256', signingKey)
    .update(timestamp + '.' + rawBody)
    .digest('hex')

  const expectedBuffer = Buffer.from(expected, 'hex')

  // There may be several v1= segments. Accept if any one of them
  matches.
  return header
    .split(',')
    .filter((part) => part.startsWith('v1='))
    .some((part) => {
      const received = Buffer.from(part.slice(3), 'hex')
      // timingSafeEqual throws if the two buffers are not the same
      length.
      return (
        received.length === expectedBuffer.length &&
        crypto.timingSafeEqual(received, expectedBuffer)
      )
    })
}
```

Three things worth getting right:

- **Use the raw body.** If you parse the JSON and re-serialise it, the bytes change and the signature no longer matches.
- **Compare in constant time.** Use `openssl.fixed_length_secure_compare` or `timingSafeEqual`, not `==` or `===`.

- **Accept multiple `v1=` segments.** During a key rotation we send one signature per valid key. If any one of them matches, the delivery is genuine.

SEND A TEST DELIVERY

Send test delivery sends a `webhook.test` event through exactly the same queue, signing and retry path as real events. If your receiver can verify a test delivery, it can verify a real one. The test delivery carries no customer data.

Following your deliveries

The list below is from **Patient-SMS**, which has been running for a day. At the top is an attempt that just failed, and an event we have given up on. The rest were delivered.

TIME	EVENT	RESPONSE	STATUS	RESPONSE TIME	ATTEMPT	
31 Aug 14:09	call.created	502	Retry in 13 minutes	—	3	Send again
31 Aug 13:12	call.ended	500	Given up	—	8	Send again
31 Aug 12:32	call.created	200	Delivered	164ms	1	
31 Aug 11:52	call.ended	200	Delivered	201ms	1	
31 Aug 11:12	call.created	200	Delivered	238ms	1	
31 Aug 10:32	call.ended	200	Delivered	275ms	1	
31 Aug 09:52	call.created	200	Delivered	312ms	1	
31 Aug 09:12	call.ended	200	Delivered	349ms	1	

The top of the page shows the numbers that matter for the last 24 hours: success rate, delivery count, typical response time, the slowest 5%, and how many failures are currently in a row.

In the list, status means:

Status	Meaning
Delivered	You answered <code>2xx</code> .
Retry in ...	The attempt failed, and we will try again at the time shown.
Given up	We have used every attempt. We won't try again on our own.

Note that **Retry in** and **Given up** can show the same response code. It is the status, not the code, that tells you whether more attempts are coming.

A failed or given-up delivery can be re-sent with **Send again**.

We keep deliveries for **30 days**, and the list shows the most recent 50.

WHEN DO WE RETRY?

We wait longer after each attempt: 1 minute, 5 minutes, 25 minutes, about 2 hours, about 10 hours, and 24 hours at most between attempts after that. With the default of 8 attempts you have roughly **3½ days** to get your endpoint healthy again before we give up on an event.

When a webhook stops by itself

Here it is **Vagttelefon** that we stopped, and the notice at the top says why.

Vagttelefon

https://hooks.tandklinik.example/relatel/3 Stopped

Stopped automatically because 10 deliveries were given up on within one hour. Fix your endpoint, then choose Resume.

Success rate (24h)
0.0%

Deliveries (24h)
10

Typical response
155ms

Slowest 5%
—

Failures in a row
5

Send test delivery
Resume
Edit
Delete

Recent deliveries Refresh

TIME	EVENT	RESPONSE	STATUS	RESPONSE TIME	ATTEMPT	
31 Aug 11:56	webhook.test	500	Given up	120ms	1	Send again
31 Aug 11:53	webhook.test	500	Given up	149ms	1	Send again
31 Aug 11:50	webhook.test	500	Given up	178ms	1	Send again
31 Aug 11:47	webhook.test	500	Given up	207ms	1	Send again
31 Aug 11:44	webhook.test	500	Given up	126ms	1	Send again
31 Aug 11:41	webhook.test	500	Given up	155ms	1	Send again
31 Aug 11:38	webhook.test	500	Given up	184ms	1	Send again
31 Aug 11:35	webhook.test	500	Given up	213ms	1	Send again

If your endpoint keeps failing, we stop the webhook. That avoids us continuing to post to something that is down, and it makes the problem visible to you.

To start again:

1. Fix the cause in your own system.
2. Press **Send test delivery** and confirm it comes back **Delivered**.
3. Press **Resume**.

If your endpoint is still broken when you resume, we will stop it again. We deliberately do not reset the failure count when a webhook stopped itself. That way you cannot click your way out of the problem without fixing it.

If you pause a webhook yourself, that is different. There we do reset the count when you start it again.

Rotating the key

Booking is mid-rotation below: both keys work, and the page shows when the old one expires.

Booking

https://hooks.tandklinik.example/relatel/1 Active

Success rate (24h)
100.0%

Deliveries (24h)
1

Typical response
197ms

Slowest 5%
—

Failures in a row
0

Send test delivery
Pause
Edit
Delete

Recent deliveries

Refresh

TIME	EVENT	RESPONSE	STATUS	RESPONSE TIME	ATTEMPT
31 Aug 13:32	webhook.test	200	Delivered	197ms	1

Signing secret

Both keys are accepted until 01 Sep, 2026 13:57, about 24 hours from now. Update your receiver to the new key before then.

New key

Previous key

a2893d83

8a9f8988

Rotate again
Cancel rotation

To change signing key, because it ended up somewhere it shouldn't or because you rotate on a schedule, press **Rotate key**.

Then:

- You get a new key, shown once. Save it.
- **For 24 hours we accept both keys.** We send one `v1=` signature per key, so you can switch over at your own pace without losing deliveries.
- The page shows fingerprints for both keys, and when the old one expires.

If you pressed it by mistake, **Cancel rotation** within those 24 hours puts the old key back and discards the new one.

After those 24 hours only the new key works.

Pausing

Pause stops all deliveries immediately, including attempts that were already queued. Use it when you need to take your receiver down for maintenance.

We do **not** store events during a pause. Events that happen while paused are not delivered afterwards. If you need to recover what you missed, your system has to fetch it through our API.

Current limits

- Only the events listed in the table above. More will follow.
 - One resource per webhook. There is no “all main numbers” option.
 - No filtering on whether a call is inbound or outbound.
 - Timeout 30 seconds maximum, 8 attempts maximum.
 - Deliveries kept for 30 days.
 - We do not guarantee ordering. Use `occurred_at` in the body if ordering matters to you.
 - In rare cases the same event may be delivered more than once. Use `Relatel-Delivery-Id` to recognise a delivery you have already handled.
-

Questions

Talk to your contact at Relatel. If you have a problem with a specific delivery, it helps a lot to include the `Relatel-Delivery-Id` and the timestamp. That lets us look up exactly that delivery.