eGuard Sign API

v1 · alpha

Your website sends a PDF. The person it belongs to approves it on their own phone, with their own USB token and PIN. **The signed PDF comes back to you in the same call.** No token on your server, no software to install, and the document is never stored with us.

```
POST https://alpha.eguard.info/v1/sign-requests
```

AUTH	BODY	DOCUMENT
`X-API-Key: eg_live_…`	`JSON or XML`	`base64, ≤ 20 MB`

REQUEST LIFETIME
15 minutes

1 Your server calls
One request: the PDF, who must sign it, where the signature goes.

2 Their phone asks them
It arrives in eGuard as an ordinary signature request.

3 Their token signs
PIN on their device. The key never leaves the token.

4 You get the PDF
Back in the open call, or from `signedUrl` later.

ON THIS PAGE

Getting a key

In eGuard's admin console: **Security** → **Authorised Websites** → **Add a website**. You get a key and a list of your own domains.

That list is the only place the key may point. A `documentUrl`, `uploadUrl`, `returnUrl` or `webhookUrl` on any other host is refused. Send the PDF in the request and collect the signed one from us and the list can stay empty — the key then accepts no URL at all, which is the tightest it can be.

> **The key belongs on your server.** A key in a web page is a key in everybody's hands. Every call on this page is server-to-server.

Create a request

JSON or XML — and the answer comes back in whichever you sent. The XML shape is eGuard's existing sign format, tag for tag, so code written against the host signing API needs almost nothing changed.

JSON

```
POST /v1/sign-requests
X-API-Key: eg_live_…
Content-Type: application/json

{
  "pdf":         "JVBERi0xLjcKJcfs…",       // base64, ≤ 20 MB
  "fileName":    "Form 16.pdf",
  "signer":      { "email": "owner@example.com" },
  "reference":    "482",                     // your id, returned untouched

  "serial":       "5C7C3E92DD",              // sign with this certificate only
  "page":         "1,3",
  "coordinates":  [100, 90, 300, 170],       // PDF points, from the bottom-left

  "enableLTV":    "yes",
  "enableTimestamp": "yes",
  "lock":         "yes",
  "reason":       "Annual filing",
  "location":     "Jaipur",
  "customText":   "For ACME Ltd",

  "webhookUrl":   "https://acme.example.com/hooks/eguard",
  "returnUrl":    "https://acme.example.com/signed?o=482"
}
```

XML

```
POST /v1/sign-requests
X-API-Key: eg_live_…
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<request>
  <command>eguardapisign</command>
  <certificate><attribute>SN=5C7C3E92DD</attribute></certificate>
  <signer><email>owner@example.com</email></signer>
  <data>JVBERi0xLjcKJcfs…</data>            <!— base64 PDF —>
  <filename>Form 16.pdf</filename>
  <reference>482</reference>
  <webhookurl>https://acme.example.com/hooks/eguard</webhookurl>
  <returnurl>https://acme.example.com/signed?o=482</returnurl>
  <pdf>
    <page>1,3</page>
    <cood>100,90</cood>                     <!— x,y … —>
    <size>220,100</size>                    <!— … plus w,h —>
    <enableltv>yes</enableltv>
    <enabletimestamp>yes</enabletimestamp>
    <lock>yes</lock>
    <reason>Annual filing</reason>
    <location>Jaipur</location>
    <customtext>For ACME Ltd</customtext>
```

```
  </pdf>
</request>
```

Only `pdf` / `<data>` and the signer's email are required. Everything else has a sensible default.

> **There is no `<pin>`.** The token is on the signer's own device and they enter the PIN there. A request carrying a PIN is refused — a website has no business holding one.

Every field

The document, and who signs it

JSON	XML	WHAT IT IS
`pdf`	`<data>`	The PDF, base64, up to 20 MB. `pdfBase64` and `document` are accepted too
`documentUrl`	`<documenturl>`	*Instead* of the PDF: an https URL we fetch it from. Must be on your key's list
`fileName`	`<filename>`	What the signer sees
`signer.email` REQUIRED	`<signer><email>`	The eGuard account that must sign. Resolved here — never taken as an id
`reference`	`<reference>`, `<txn>`	Your own id. Returned untouched in every answer and webhook

Which certificate

JSON	XML	WHAT IT IS
`serial`	`<attribute>SN=…`	The serial printed on the certificate
`thumbprint`	—	Its SHA-1 thumbprint, if that is what you hold

Checked while you are still on the call: a serial that account does not hold is refused there and then, and the answer tells you which certificate it matched — `certificate: { serial, thumbprint, subject, matchedBy }`. If the named certificate is not on the token when the signer opens the request, their app refuses it. It is never quietly swapped for another.

Leave it out and the signer picks.

How it is signed

JSON	XML	DEFAULT	WHAT IT IS
`enableLTV` · `ltv`	`<enableltv>`	on	Long-term validation — revocation data embedded so it still verifies years later

JSON	XML	DEFAULT	WHAT IT IS
`enableTimestamp` · `timestamp`	`<enabletimestamp>`	`on`	A TSA timestamp on the signature
`lock`	`<lock>`	`on`	No changes allowed after signing
`reason`	`<reason>`	—	Printed on the signature
`location`	`<location>`	—	Printed on the signature
`customText` · `text`	`<customtext>`	—	An extra line in the appearance

`yes` / `no` and `true` / `false` both work, in both formats.

Waiting, or not

JSON	XML	DEFAULT	WHAT IT IS
`wait`	`<wait>`	`true`	Hold the connection until it is signed
`waitSeconds`	`<waitseconds>`	`180`	Up to 300. Set your HTTP client's timeout above this

Where the signature goes

Coordinates are **PDF points, measured from the bottom-left** of the page — the same convention Acrobat and every PDF library use. A4 is 595 × 842 points.

JSON	XML	WHAT IT IS
`coordinates`	`<cood>`	`[x1, y1, x2, y2]`. In XML, `<cood>x,y</cood>` plus `<size>w,h</size>` says the same thing
`page`	`<page>`	`3` · `"1-5"` · `"1,3,6"` · `all` · `odd` · `even` · `first` · `last`
`placements`	`<page>custom</page>` + `<custom>`	A different box on each page
`formField`	`<formfield>`	Sign in a named signature field the document already has
`invisibleSign`	`<invisiblesign>`	No appearance drawn at all — the signature is in the document

A different box on each page

JSON

```
"placements": [
  [1, 100,  90, 300, 170],
  [3,  60, 700, 260, 780]
]
```

XML

```
<page>custom</page>
<custom>
  <page><pageno>1</pageno><cood>100,90,300,170</cood></page>
  <page><pageno>3</pageno><cood>60,700,260,780</cood></page>
</custom>
```

One entry per box; the same signature is shown at each. Inside `<custom>`, `<cood>` takes all four numbers.

> **Say nothing and it still arrives ready to sign** — the usual box, bottom right, first page. A box off the edge of a page is pulled back onto it rather than refused, and the signer can move it before signing.
>
> The one thing the signer cannot change is the rest of it: the reason, the location, the timestamp, the LTV and the lock are yours. Their app shows them and offers no way to edit them.

The answer

Three of them, and the status code tells them apart.

`200` Signed, in the same call

JSON

```
{
  "success":     true,
  "requestId":   "sr_9f2c…",
  "signedPdf":   "JVBERi0xLjcKJTRl…",   // the signed document, base64
  "certificate": "CN=…",
  "signedAt":    "2026-09-15T05:12:44.101Z",
  "reference":   "482"
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <success>true</success>
  <requestId>sr_9f2c…</requestId>
  <signedPdf>JVBERi0xLjcKJTRl…</signedPdf>
  <certificate>CN=…</certificate>
  <signedAt>2026-09-15T05:12:44.101Z</signedAt>
  <reference>482</reference>
</response>
```

`202` Nobody has picked up yet

```
{
  "success":     false,
  "state":       "pending",
  "requestId":   "sr_9f2c…",
  "waitUrl":     "…/v1/sign-requests/sr_9f2c…?wait=180",
  "signedUrl":   "…/v1/sign-requests/sr_9f2c…/signed",
```

```
  "openUrl":     "https://alpha.eguard.info/s/sr_9f2c…",
  "liveDevices": 1
}
```

Nothing failed. The request stays alive for fifteen minutes from when you made it: wait again on `waitUrl`, collect from `signedUrl` when it is done, or let the webhook tell you. `202` rather than `200` so a client that reads only the status code cannot mistake *pending* for *signed*.

`liveDevices` is how many of the signer's devices were holding a connection at that instant. Zero is not a failure — the push still reaches a phone in a pocket.

`200` Declined, or failed

```
{
  "success": false,
  "state":   "declined",
  "error": { "code": "DECLINED", "message": "The owner declined.", "category": "user" }
}
```

A refusal comes back as a refusal, not as a timeout.

Errors

CODE	MEANS
`DECLINED`	The signer said no
`SIGN_FAILED`	The signature could not be made — `message` says why
`EXPIRED`	Fifteen minutes passed unanswered
`BAD_REQUEST`	The request itself was wrong. XML parse errors arrive as this

HTTP	MEANS
`400`	Your request — a missing field, a PDF that is not a PDF, both `pdf` and `documentUrl`
`401`	Unknown or inactive API key
`403`	A URL that is not on your key's website list
`404`	No eGuard account for that address, no such certificate, or no such request
`409`	The signed copy has already been collected
`413`	Over 20 MB
`429`	Rate limit — 600 an hour on a live key, 60 on a sandbox one
`503`	Too many documents in flight. Try again shortly

Ask again, and collect

```
GET /v1/sign-requests/{id}
X-API-Key: eg_live_…
Accept: application/xml          // optional — or add ?format=xml

{ "state": "signed", "signedAt": "…", "certificate": "CN=…",
  "signedUrl": "…", "collectedAt": null, "reference": "482", "error": null }
```

`state` is one of `pending`, `opened`, `signed`, `declined`, `failed`. A key only ever sees its own requests.

Add `?wait=120` and it holds the line instead of answering *pending* — that is how you get back on a request whose first wait ran out, without polling.

```
GET /v1/sign-requests/{id}/signed
X-API-Key: eg_live_…

→ 200 application/pdf
```

Collected **once**: after that the bytes are gone and this answers `409`. A waiting call that received `signedPdf` has already collected it.

Webhook

Fires on every final state, signed with your API key so you can prove it came from us. Three attempts, then it gives up — the state is still there to be asked for.

```
POST <your webhookUrl>
X-eGuard-Signature: t=1788…,v1=<HMAC-SHA256 of "t.body">

{ "requestId": "sr_9f2c…", "state": "signed", "signedUrl": "…",
  "certificate": "CN=…", "reference": "482" }
```

Verifying it

```
$raw = file_get_contents('php://input');
[$t, $v1] = sscanf($_SERVER['HTTP_X_EGUARD_SIGNATURE'], 't=%[^,],v1=%s');

$ok = hash_equals(hash_hmac('sha256', "$t.$raw", EGUARD_API_KEY), $v1)
      && abs(time() - (int)$t) < 300;   // and not a replay
```

The timestamp is inside the signed string, so a captured callback cannot be replayed later.

Opening the app from your page

Every answer carries `openUrl`. On an Android phone that link opens the eGuard app straight on the request — the way a UPI link opens a UPI app — because the domain publishes the app's fingerprint and Android verifies it.

```
<!— on a phone: the app opens on the request —>
<a class="btn" href="{{ openUrl }}">Sign on your phone</a>
```

```
<!— at a desk: show the same URL as a QR code and watch the status —>
<img src="/qr?d={{ openUrl | urlencode }}" alt="Scan to sign">
```

A laptop cannot open an app on a phone — nothing can. The QR is how every desktop flow of this kind works, UPI included.

If the app is not installed, the same URL is a real web page that offers the download. The signer's own desktop eGuard app, where they have one, receives the request directly and never needs the link at all.

Snippets

curl

```
curl -s https://alpha.eguard.info/v1/sign-requests \
  -H "X-API-Key: $EGUARD_KEY" -H 'Content-Type: application/json' \
  -d "$(jq -n --arg pdf "$(base64 -w0 form16.pdf)" \
        '{pdf:$pdf, fileName:"Form 16.pdf", signer:{email:"owner@example.com"},
          page:"1", coordinates:[100,90,300,170], reference:"482"}')" \
  | jq -r 'if .success then .signedPdf else .error end' | base64 -d > signed.pdf
```

PHP

JSON

```
$body = json_encode([
  'pdf'         => base64_encode(file_get_contents('form16.pdf')),
  'fileName'    => 'Form 16.pdf',
  'signer'      => ['email' => 'owner@example.com'],
  'serial'      => '5C7C3E92DD',
  'page'        => '1,3',
  'coordinates' => [100, 90, 300, 170],
  'reference'   => '482',
  'waitSeconds' => 240,
]);

$ch = curl_init('https://alpha.eguard.info/v1/sign-requests');
curl_setopt_array($ch, [
  CURLOPT_POST           => true,
  CURLOPT_POSTFIELDS     => $body,
  CURLOPT_HTTPHEADER     => ['Content-Type: application/json',
                             'X-API-Key: ' . EGUARD_API_KEY],
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_TIMEOUT        => 300,        // it waits for a person
]);
$res  = json_decode(curl_exec($ch), true);
$code = curl_getinfo($ch, CURLINFO_HTTP_CODE);

if ($code === 200 && ($res['success'] ?? false)) {
    file_put_contents('signed.pdf', base64_decode($res['signedPdf']));
} elseif ($code === 202) {
    // Nobody at the phone yet. Keep $res['requestId'] and come back to
    // $res['signedUrl'], or wait for the webhook.
} else {
    error_log('eGuard: ' . ($res['error']['message'] ?? 'unknown'));
}
```

XML

```
$xml = '<?xml version="1.0" encoding="UTF-8"?>
<request>
  <command>eguardapisign</command>
  <certificate><attribute>SN=5C7C3E92DD</attribute></certificate>
  <signer><email>owner@example.com</email></signer>
  <data>' . base64_encode(file_get_contents('form16.pdf')) . '</data>
  <filename>Form 16.pdf</filename>
  <reference>482</reference>
  <pdf><page>1,3</page><cood>100,90</cood><size>220,100</size>
       <enableltv>yes</enableltv><lock>yes</lock>
       <reason>Annual filing</reason></pdf>
</request>';

$ch = curl_init('https://alpha.eguard.info/v1/sign-requests');
curl_setopt_array($ch, [
  CURLOPT_POST           => true,
  CURLOPT_POSTFIELDS     => $xml,
  CURLOPT_HTTPHEADER     => ['Content-Type: application/xml',
                             'X-API-Key: ' . EGUARD_API_KEY],
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_TIMEOUT        => 300,
]);
$r = simplexml_load_string(curl_exec($ch));

if ((string)$r->success === 'true') {
    file_put_contents('signed.pdf', base64_decode((string)$r->signedPdf));
}
```

Node

```
const r = await fetch('https://alpha.eguard.info/v1/sign-requests', {
  method: 'POST',
  headers: { 'X-API-Key': process.env.EGUARD_KEY,
             'Content-Type': 'application/json' },
  body: JSON.stringify({
    pdf: fs.readFileSync('form16.pdf').toString('base64'),
    fileName: 'Form 16.pdf',
    signer: { email: 'owner@example.com' },
    page: '1,3',
    coordinates: [100, 90, 300, 170],
    reference: '482',
  }),
  signal: AbortSignal.timeout(300_000),   // it waits for a person
});
const j = await r.json();

if (r.status === 200 && j.success) {
  fs.writeFileSync('signed.pdf', Buffer.from(j.signedPdf, 'base64'));
} else if (r.status === 202) {
  // come back to j.signedUrl, or wait for the webhook
}
```

Testing

A **sandbox** key (`eg_test_…`) accepts `localhost` , a private network address and `.local` / `.test` , so you can work from your own machine or Postman. A PDF sent in the request needs no address at all.

Sandbox keys are capped at 60 requests an hour; live keys get 600. One request at a time per signer.

> **See it work before you write anything.** alpha.eguard.info/try-signing runs this API with no code: a PDF, an account email, and every option on this page. Useful for watching what the signer sees.

What we keep

THING	WHERE IT LIVES
The document	In memory while the request is alive, sealed to the signer's device, dropped the moment it is answered. Never written to a disk or a database
The signed copy	Until you collect it, or fifteen minutes after signing — whichever comes first
The PIN	Never leaves the signer's device. Not sent to you, not sent to us
What is kept	That it was signed: the certificate, the time and your reference, in the signer's own activity record

Send a `documentUrl` and an `uploadUrl` instead and the document does not pass through eGuard in either direction: the signer's device fetches it from you and puts the signed copy back.

---

eGuard Sign API · v1 · Base URL `https://alpha.eguard.info` — production hosts differ; use the one your key was issued for.

Keys, websites and usage: eGuard admin console → Security → Authorised Websites. Questions: support@eGuard.info