VTX Devices 1: wake-ups and enrolment

Nov 15, 2026 · VTX 1 · extension · draft d01 · MIT licence · vtx.foundation/spec/v1/devices-1/

§1 Status and scope

Devices 1 is an extension of VTX 1, draft d01, in conformance profile 1, built on the core at draft d03. It covers what a person's devices need beyond a single sign-in:

- **Push wake-ups.** When an agent asks for an approval, or an application raises a notice, and no client is attached to the session, the server wakes the user's phone or browser. It uses Web Push directly, or the client publisher's gateway to APNs or FCM. The push carries no content, only a random tag the device chose; the device then connects and asks what is waiting.
- **Device keys.** A device signs in with its own key pair, kept in secure hardware where it has some. The key goes in the TLS handshake as a raw public key (RFC 7250, method `rpk`), or signs WELCOME's challenge (method `device`). Clients that pin a server's raw key can also connect without a certificate authority.
- **Enrolment.** A new device joins the user's devices in one of three ways:
  - approved from a device that is already signed in, by scanning a QR code or typing an eight-character code;
  - by enrolling its own key after a full sign-in;
  - with an enrolment token, for managed fleets. Fleet devices can also get a certificate from the server's enterprise CA.

Out of scope.

- Account recovery when every device is lost. This stays with the v1 sign-in methods: passkeys, OIDC and smart cards.
- Notifications that carry content. They are left out on purpose: the push service never sees what the user is asked.
- Several clients on one session. That is the multi-viewer extension, next on the roadmap.
- Mobile device management itself.

What changes in v1. Nothing changes for a peer that does not offer the extension. Devices 1 adds:

- three sign-in methods: `rpk`, `device` and `enrol`;
- an AUTH_FAILED reason, `denied`;
- a close code, 0x0F `revoked`;
- a dynamic channel type, `device`;
- two capability keys, `notify` and `device`.

The v1 registries already leave new methods and reasons to specification. To express them, the v1 CDDL gains four sockets that v1 itself leaves empty: `$auth-method-ext`, `$$auth-offer-ext`, `$$auth-ext` and `$auth-failed-ext`.

Headless use. A HELLO that offers `notify` or `device` but no display capability asks for a headless session, as in Exec 1. A woken phone uses this to sign in, ask what is waiting and leave, without starting an application.

§2 Capabilities and limits

The client offers what it can do in HELLO. The server answers with what its policy grants the signed-in user, as with every v1 capability.

Key	Client offers	Server answers
`notify`	`push`: the kinds of registration it can make, `webpush` and `gateway`. An empty map means it shows notices but cannot be woken	The kinds it accepts. With `webpush`, also `vapid`: its VAPID public key (RFC 8292), 65 bytes, uncompressed P-256
`device`	`enrol` (it can enrol its own key), `approve` (it can approve other devices), `certs` (it can use a certificate)	The same flags as granted, plus `pins`: the SHA-256 of the SubjectPublicKeyInfo of the server's current and next raw keys
`auth` (v1)	May now list `rpk`, `device` and `enrol`	WELCOME offers only methods the client listed

With `notify` or `device` accepted, the client may open one `device` channel per connection; Section 10 lists its frames. Device listing and removal need only the `device` key; `{}` grants just those.

Limits. Devices 1 adds no negotiated limits. Three fixed numbers apply:

- A session holds at most 16 live notices; a 17th drops the oldest.
- An enrolment code lives at most 300 seconds.
- After 5 lookups of codes that do not exist, the server answers `limit` to that user for a while.

§3 Push wake-ups

A phone in a pocket has no connection open. Devices 1 lets the server wake it when something needs the user, without telling the push service what that is.

Registering. After READY the client opens a `device` channel and sends REGISTER:

- `tag`: 16 random bytes the client chooses for this registration. It is the only thing a wake-up carries, and tells the client which server woke it. A client never reuses a tag with another server.
- `push`, one of two kinds:
  - `webpush`: the `endpoint`, `p256dh` and `auth` of a Web Push subscription (RFC 8030, RFC 8291), made with the server's `vapid` key. Browsers and many desktop platforms work this way, with no gateway.
  - `gateway`: the `url` of the client publisher's gateway, an opaque `token` and the `app` ID. APNs and FCM accept pushes only from the app's publisher, so the publisher's gateway forwards the wake-up. The token is best sealed for the gateway, so the VTX server never sees the APNs or FCM token itself.
- `scope`: `user` (the default) wakes for any of the user's sessions; `session` only for this one.

REGISTER without `push` removes the registration with that tag. The client registers again whenever the platform rotates its subscription or token. The server drops a registration when the push service or gateway answers 404 or 410, when the device is removed, and, for scope `session`, when the session ends.

What wakes a device. A wake-up goes out when a session the registration covers has no connection attached, and one of these is waiting in it:

- an APPROVAL in the trusted zone;
- a notice with urgency `normal` or `high`.

Permission and clipboard requests never wake a device: they answer something the user just did. Notices with urgency `low` wait quietly for the next connection.

The wake-up. It names the tag and nothing else, so the push service learns only that this device has something waiting.

- Web Push: an RFC 8030 message, encrypted as RFC 8291 describes and signed with VAPID. The plaintext is exactly `{"vtx":"wake","tag":"<tag>"}`, the tag in unpadded base64url. The headers:
  - `Topic` is the same tag, so a wake-up the device has not yet received is replaced, not stacked;
  - `Urgency` is `high` when an approval or a high notice waits, `normal` otherwise;
  - `TTL` is the seconds until the earliest of those items expires, at most a day.
- Gateway: an HTTPS POST of `{"vtx":"wake","tag":…,"token":…,"urgency":…,"ttl":…}`. A 2xx answer means accepted; 404 or 410 means the token is gone; 429 means back off as `Retry-After` says. The gateway turns it into an APNs or FCM message for the app.

Pacing. The server sends at most one wake-up per registration until that device connects again. If items still wait 10 minutes later and the device has not connected, it may send one reminder. Pushes are best-effort, and the reminder covers a lost one without letting a busy agent drain the phone's battery.

After waking. The device reads what is waiting, shows it, and leaves:

1. It connects with a headless HELLO, signs in with `device` or `rpk`, opens a `device` channel and sends WAITING.
2. ITEMS lists every approval and waiting notice in the user's sessions: the session, the title, the requester and risk of an approval, the body of a notice.
3. The client shows them as system notifications and closes the connection. If ITEMS is empty, because the user already answered on another device, it shows nothing.
4. When the user taps one, the client resumes that session (v1 Section 6). The server sends the pending APPROVAL again, and the user answers it as in v1, with a passkey when the approval asks for proof.

Notices. An application raises a notice with the `notice` line (Section 7):

- The server sends each live notice as NOTICE on the attached client's `device` channel, again whenever such a channel opens.
- The client shows a notice as a system notification when its window is not in front. It labels the notice with the server's host and the application's claimed name, and never styles it like the trusted zone.
- Opening or dismissing it sends ACTION. NOTICE without a title withdraws one.
- A session holds at most 16 live notices; a 17th drops the oldest, and the application hears `dropped`.

§4 Device keys and raw public keys

An enrolled device signs in with a key pair of its own instead of a user's passkey or password. The private key stays on the device: in the Secure Enclave, a TPM, the Android Keystore, or a non-extractable WebCrypto key in a browser.

Keys.

- A device key is P-256, used with ES256 (COSE −7), or Ed25519, used with EdDSA (COSE −8). Servers accept both; a client picks what its hardware holds, which is usually P-256.
- ES256 signatures are encoded as r ‖ s, as v1 does for `x509`.
- A key's ID, `kid`, is the SHA-256 of its DER SubjectPublicKeyInfo. That is the same hash as a DANE 3 1 1 record and v1's `cas` list, so one fingerprint serves everywhere.

Two sign-in methods. They mirror v1's pair for certificates, `tls-cert` and `x509`: one proves the key in the transport, the other in AUTH.

Method	How the key is proved	Where it works
`rpk`	The client presents its raw public key in the TLS 1.3 handshake (RFC 7250, client certificate type RawPublicKey). AUTH carries no credential; the server looks up the kid	TLS and QUIC, when TLS ends at the VTX server. Like `tls-cert`, it does not pass load balancers that end TLS
`device`	WELCOME's offer carries a 32-byte `challenge`. The client signs `"vtx1-device"` ‖ host ‖ challenge, with host as for the v1 OIDC nonce, and sends `device`: `{ kid, alg, sig }`	Every binding, including WebSocket behind a proxy and browsers

A kid the server does not know fails with `credential`; a removed device fails with `revoked`. Over Noise, where no host name was provisioned, host is the lower-case hex SHA-256 of the server's static key, so a signature still names the server it was made for.

Server raw keys. RFC 7250 works in both directions. A server may present a raw public key instead of a certificate, to clients that have its key pinned:

- from a DNSSEC-secure TLSA record with usage DANE-EE, selector SPKI (v1 Section 5);
- from provisioning, such as a fleet's configuration;
- on first use, where the client is configured to allow it. The trusted zone then shows the key fingerprint as the server's identity, as with Noise XX.

The server's `device` capability lists in `pins` the hashes of its current and next keys. A client that pinned the current key adds the next one, so a planned key rotation does not lock pinned clients out. The trusted zone shows the trust mode as “pinned key”.

§5 Enrolling a device

A device gets its key accepted in one of three ways:

Way	Who vouches for the device	Typical use
Approved	A device the user is already signed in on, after the user scans or types the new device's code	A person's new phone, tablet or laptop
Self	The device's own full sign-in, with a passkey, OIDC or a smart card	Remembering a device after a strong sign-in
Token	An enrolment token from an administrator or an MDM profile	Kiosks, fleets, and devices without a user present

Approved by another device. The user holds both devices; the new one has nothing to sign in with yet.

> **Diagram: approving a new device · 3 parties, 9 messages.** Drawn in the PDF version, `07-devices-1.pdf`.

The tablet joins only after the user confirms on the phone, and the server checks that the phone saw this tablet's key.

1. The new device makes a key pair, lists `enrol` in HELLO, and gets an `enrol` offer with a 32-byte challenge. AUTH carries `enrol`: the public key, `alg`, a signature over `"vtx1-enrol"` ‖ host ‖ challenge, the `name` the user gave it, and its `platform`. The signature proves the device holds the key it asks to enrol.

2. The server answers with AUTH_CHALLENGE, carrying `code` and `expires` (at most 300 seconds). While the code is valid, it replaces v1's 120-second sign-in limit. The new device shows the code in two groups of four, `K7QF-M2XD`, and a QR code of the enrolment link: `vtx://ops.example.net/#enrol=K7QFM2XD&k=…`. Here `k` is the first 16 bytes of its key ID, in base64url. Both sit in the fragment, which never leaves a client, so a client without Devices 1 that scans the link just opens the server.

3. On a signed-in device, the user scans the QR code or types the code. The client opens a `device` channel and sends LOOKUP. PENDING returns:

   - the new device's key ID, name and platform;
   - where the request came from, as the server sees it;
   - how long ago it was made;
   - whether the approval needs proof.

   After a scan, the client compares the key ID with `k` and refuses if they differ.

4. The client's trusted zone asks whether to add this device, showing its name, platform, place and age. There is no default button, and the prompt says to allow only a device the user is holding.

5. DECIDE carries the decision and the SHA-256 of PENDING as received, like a v1 approval. When PENDING asks for proof, DECIDE also carries a passkey assertion or a card signature over `"vtx1-approval"` ‖ session ‖ digest. The server checks both and binds the key to the approving user. It then sends the new device READY, or AUTH_FAILED with reason `denied`.

Codes.

- Eight symbols of Crockford's base32, which leaves out I, L, O and U: 40 bits.

- Typed codes ignore case, spaces and hyphens, and read I and L as 1 and O as 0.
- A code works once and lives at most 300 seconds.
- After five lookups that find nothing, the user gets `limit` for a while, so codes cannot be guessed online.

Enrolling its own key. A client signed in by other means can send ENROL on its `device` channel: the key, a signature over `"vtx1-enrol"` ‖ host ‖ the challenge from OPEN_OK, a name and a platform. From then on it signs in with `device` or `rpk`. A server may require a session signed in with `webauthn`, `oidc` or `x509`, and answer `policy` after `token` or `device`.

Enrolment tokens. AUTH `enrol` with a `token` needs no approval: the server checks the one-time token and sends READY. Tokens come from an administrator or an MDM profile. Like v1 `token` credentials, each is bound to one host and sent only to that host.

Listing and removing devices. LIST returns DEVICES: for each device its key ID, name and platform, when it was added and last seen, how it was enrolled, and whether it is this one. REVOKE removes a device. The server then:

- forgets its key and its push registrations;
- revokes the certificates issued to it;
- closes its open connections with 0x0F `revoked`.

A client that receives `revoked` deletes its key and its resume tokens. A user removes a lost phone from any other device. A device can also remove itself, which is how a client signs out for good.

§6 Fleet enrolment: certificates

Managed fleets often want a certificate from their own CA rather than a bare key, so the same identity works for Wi-Fi, VPN and VTX alike. This is the enterprise CA row of the trust table in the outline: a customer-installed root, opt-in.

Getting a certificate. A device whose key is enrolled sends CSR on its `device` channel:

- The request is a DER PKCS #10 for that same key: the key this connection signed in with, or just enrolled. The request's own signature proves possession.
- The server answers CERT with the chain, end-entity first, or RESULT with a reason.
- What the certificate says is the CA's choice: subject, lifetime, and the clientAuth extended key usage.
- The server may front an existing CA through EST (RFC 7030), ACME or SCEP. Devices 1 only carries the request and the chain.

With the certificate, the device can sign in with v1 `tls-cert` or `x509` to any server that trusts the fleet's root. That includes servers that know nothing of Devices 1.

Renewal.

- A device renews when two thirds of the certificate's lifetime have passed. It sends a new CSR for the same key over any signed-in connection.
- To change keys, it enrols the new key with ENROL, requests a certificate for it, then removes the old key with REVOKE.

- DEVICES shows each device's newest certificate expiry, so an administrator's client can spot devices that stopped renewing.

Without a user. A kiosk or a sensor enrols with an enrolment token, gets its certificate in the same session, and from then on signs in with `tls-cert` or `rpk`. Its token is spent on first use, so a copied provisioning profile cannot enrol a second device.

§7 Application interface

Applications see one thing new: notices. Push registrations, device keys and enrolment stay between the client and the server. An application never sees a key, a push endpoint or an enrolment code, and cannot add a device.

Line from the application (`vtx-app-devices-1.schema.json`):

Kind	Fields	Effect
`notice`	`id`; `title` (up to 120 characters); `body` (up to 1,000); `urgency`: `low`, `normal` (default) or `high`; `target`: a node to show when the user opens it	Raises a notice, or replaces the live one with the same `id`. Without `title`, withdraws it

Line to the application (`vtx-app-events-devices-1.schema.json`):

Kind	Fields	Meaning
`notice`	`id`, `action`	`open` or `dismiss`: the user acted on it, and it is gone. `dropped`: a 17th notice pushed out this oldest one

Rules.

- Approvals are unchanged: the v1 `approval` line is all an agent needs. A pending approval now also wakes the user's device when no client is attached.
- A notice's `target` must name a node in the scene; opening the notice brings that node into view.
- `urgency` chooses how the user is reached. `low` never wakes a device and waits for the next connection. `normal` wakes a device. `high` also asks the platform to deliver at once, and may break through quiet hours where the user allows it.
- Notices are not a second trusted zone. The client labels them with the server and the application's claimed name, and they carry no buttons beyond open and dismiss. Anything the user must approve goes through `approval`.

A build server, for example:

```
{"kind":"notice","id":"build-7","title":"Build 7 finished","body":"All 1,204 tests passed in 4 min 12 s","target":"ci"}
```

It hears `{"kind":"notice","id":"build-7","action":"open"}` when the user taps it, whether on the desktop where the session runs, or on a phone that was woken and then resumed the session.

§8 Security considerations

What the push path learns.

- Apple, Google and the Web Push services see a random tag, the time and the urgency, never what waits.
- Web Push payloads are also encrypted end to end (RFC 8291).
- A gateway sees the tag, its own token, and the address of the VTX server that called it. Publishers should run gateways that keep no logs beyond rate limiting.
- What the user is asked travels only over the authenticated VTX connection, in ITEMS and in the zone.

Server-side request forgery. Registrations make the server send HTTPS requests to URLs a client chose. So a server:

- sends only to `https` URLs;
- refuses addresses in loopback, private and link-local ranges unless configured otherwise;
- may limit Web Push to known push services and gateways to an allow-list;
- caps registrations per device and answers `limit` beyond that.

Device-code phishing. An attacker can start an enrolment on their own device and persuade the victim to type the code “to verify the account”. Approval would add the attacker's device. Devices 1 limits this in five ways:

- The approval is shown in the trusted zone only, with the device's name, platform, place and age, and the instruction to allow only a device the user is holding.
- High-value accounts can require proof: PENDING asks for a passkey or card signature, bound to the PENDING the user saw.
- Codes expire within five minutes and work once.
- Scanning beats typing, because a QR code needs the two devices side by side. A server may turn typed codes off.
- A server may flag, or refuse, a request whose place differs from the approving device's.

The reverse attack, a victim's new device enrolled into the attacker's account, shows in the victim's trusted zone as a different signed-in user.

Key theft and removal.

- Device keys should be non-exportable. An exportable key is only as safe as the device's disk encryption.
- A lost device is removed from any other device, which closes its connections at once.
- Removal also revokes its certificates and drops its push registrations, so a stolen phone stops receiving even the content-free wake-ups.

Relay. Device and enrolment signatures name the host, like v1 `x509` signatures. A signature made for one server is useless at another. They do not bind the TLS session, so that they pass load balancers; TLS server authentication is what stops a man in the middle.

Notices are not approvals. An application could word a notice to look urgent or official. Clients label notices as the application's, keep them out of the trusted zone, and give them no action besides open and dismiss.

What ITEMS reveals. Approval titles and notice bodies reach every device the user enrolled, including a phone's lock screen. Clients follow the platform's lock-screen privacy setting. A server may also replace the title of a `high` risk approval in ITEMS with a generic one, such as “An approval is waiting”; the full request is shown once the user resumes the session.

§9 Definitions and checks

The machine-readable definitions are in the bundle under `ext/devices-1/`:

File	What it holds
`vtx-devices-1.cddl`	The sign-in additions (methods, offer and AUTH fields, the `denied` reason), the capabilities, and every frame of the `device` channel
`vtx-app-devices-1.schema.json`, `vtx-app-events-devices-1.schema.json`	Application lines and events, generated by `tools/gen_ext_devices.py` on top of the earlier extensions' schemas
`examples/vectors.json`	Test keys (P-256 and Ed25519) with their key IDs; sign-in and enrolment signatures; the enrolment link; typed-code normalisation; both wake-up bodies; the PENDING digest and proof challenge
`examples/valid/trace-*.jsonl`	Three traces: a tablet approved by QR code; a phone woken for an approval; self-enrolment, removal, a certificate, a notice and raw-key sign-in over TLS
`examples/invalid/*.jsonl`	52 broken traces and 16 broken application lines, each naming the check that must catch it

`tools/make_devices.py` writes the vectors and traces. Every key derives from a fixed label, and every signature is deterministic (Ed25519, and ECDSA as RFC 6979 describes), so a rerun produces the same bytes. The v1 CDDL gains the four empty sockets of Section 1, and v1 on its own still rejects every Devices 1 form.

191 checks. Section 14 of `tools/validate.py` adds 191 checks, all passing, for 2,543 in the bundle:

- 5: the schemas and the CDDL load, and v1 alone rejects the new methods, offer fields and reason.
- 60: 30 well-formed and 30 malformed samples against the CDDL.
- 27: the vectors, recomputed independently. That covers key IDs from private keys, both kinds of signature verified and reproduced, the link, six typed codes, the wake-up bodies and Topic, and the PENDING digest.
- 55: three valid traces replayed, and 52 invalid ones, each failing for its stated reason.
- 44: application lines, their schemas and their rules.

What the trace checks prove. The simulator follows the server, every connection and every device, and checks real signatures:

- **Sign-in.** WELCOME offers only methods the client listed. READY follows only a sign-in whose signature verifies against an enrolled key that has not been removed. A passkey assertion is checked for its challenge, its rpId hash, its flags and its signature.
- **Approval.** A new device gets READY only after an approval whose digest matches the PENDING the approver saw. When PENDING asked for proof, the passkey assertion must cover that session and digest. After a scan, the key in the link must match PENDING.

- **The device channel.** Only the client opens it, once per connection, and every request gets one answer of the right kind. REGISTER, ENROL, LOOKUP, WAITING and CSR succeed only within what the server granted.
- **Wake-ups.** ITEMS lists exactly what waits in the user's sessions. A wake-up goes only to a registered device, only when no connection is attached, and carries nothing but the tag, with the right Topic and urgency. It goes once, plus at most one reminder, until the device connects.
- **Removal.** A removed device's connections are closed with `revoked` before anything else reaches them, and it cannot sign in again.
- **Certificates.** A certificate is issued only for this device's enrolled key.

§10 Wire summary

The `device` channel.

- Only the client opens it, with an odd channel ID and no OPEN params. OPEN_OK's params carry a 32-byte `challenge` for ENROL signatures.
- At most one per connection; the server refuses a second with `in-use`.
- Requests carry `req`, unique among the client's outstanding requests on the channel. Each is answered once: WAITING by ITEMS, LOOKUP by PENDING or RESULT, LIST by DEVICES, CSR by CERT or RESULT, and every other request by RESULT.

Type	Name	Direction	Payload
0x10	REGISTER	C→S	`{ req, tag, ? push, ? scope }`; without `push`, removes the registration
0x11	WAITING	C→S	`{ req }`
0x12	ITEMS	S→C	`{ req, items }`: approvals and notices waiting in the user's sessions
0x13	NOTICE	S→C	`{ id, ? title, ? body, ? urgency, ? target }`; without `title`, withdraws the notice
0x14	ACTION	C→S	`{ id, action }`: `open` or `dismiss`
0x15	ENROL	C→S	`{ req, key, alg, sig, name, ? platform }`
0x16	LOOKUP	C→S	`{ req, code }`
0x17	PENDING	S→C	`{ req, code, kid, name, ? platform, ? from, age, expires, ? uv, ? proof }`
0x18	DECIDE	C→S	`{ req, code, decision, digest, ? assertion, ? x509 }`
0x19	LIST	C→S	`{ req }`
0x1A	DEVICES	S→C	`{ req, devices }`
0x1B	REVOKE	C→S	`{ req, kid }`
0x1C	CSR	C→S	`{ req, csr }`
0x1D	CERT	S→C	`{ req, certs }`: end-entity first
0x1E	RESULT	S→C	`{ req, ok, ? reason, ? message }`; reasons `invalid`, `not-found`, `expired`, `policy`, `proof`, `digest`, `limit`, `unsupported`

Additions to v1.

Where	Addition
Auth methods	`rpk`, `device`, `enrol`
Auth offers (WELCOME, AUTH_CHALLENGE)	`challenge` (32 bytes) for `device` and `enrol`; `code` and `expires` for the second step of `enrol`
AUTH	`device`: `{ kid, alg, sig }`; `enrol`: `{ key, alg, sig, name, ? platform, ? token }`
AUTH_FAILED reasons	`denied`
Capability keys	`notify`, `device`
Close codes	0x0F `revoked`: the device was removed; the client does not resume
Dynamic channel types	`device`
OPEN_REFUSED reasons, `device`	`policy`, `in-use`
Signed bytes	`"vtx1-device"` ‖ host ‖ challenge; `"vtx1-enrol"` ‖ host ‖ challenge. Approvals of a new device reuse v1's `"vtx1-approval"`
Application interface kinds	`notice`, in both directions
CDDL sockets	`$auth-method-ext`, `$$auth-offer-ext`, `$$auth-ext`, `$auth-failed-ext`

Outside the VTX connection.

- The Web Push plaintext: `{"vtx":"wake","tag":"<tag>"}`.
- The gateway POST body: `{"vtx":"wake","tag","token","urgency","ttl"}`.
- The enrolment link: `vtx://host/#enrol=<code>&k=<key ID prefix>`.

The link uses v1's `vtx` URI scheme, so Devices 1 needs no registrations beyond v1's.