IoT Edge Device Firmware Update Protocol

Secure Delivery, Installation, Activation, and Recovery

EFUP 1.0

Protocol Engineering Specification

Document ID:	EFUP-SPEC-1.0
Status:	Stable
Published:	2026-08-02
Compatibility:	Protocol major version 1
Intended audience:	Device, bootloader, cloud, and security engineers

Abstract. This specification defines a transport-neutral protocol for securely updating constrained and intermittently connected IoT edge devices. It covers update discovery, signed manifests, artifact transfer, integrity verification, anti-rollback enforcement, atomic activation, health confirmation, automatic recovery, staged rollout, status reporting, and conformance requirements.

Document control

Version	Date	Change	Status
1.0	2026-08-02	Initial stable protocol definition.	Stable

The key words “**MUST**”, “**MUST NOT**”, “**SHOULD**”, “**SHOULD NOT**”, and “**MAY**” in this document indicate requirement levels. Requirements are normative only when these words appear in uppercase bold type. Examples and explanatory notes are informative unless explicitly identified as normative.

Change policy

The major protocol version changes when wire compatibility or required security semantics change. A minor document revision may clarify behavior or add optional capabilities without invalidating a conforming version-1 implementation. New fields are additive: receivers **MUST** ignore unknown non-critical fields and **MUST** reject an unknown field listed in the manifest’s `critical` array.

Contents

1 Purpose and scope

The IoT Edge Device Firmware Update Protocol (EFUP) coordinates a trusted update service, one or more artifact repositories, and an edge device. It is designed for devices that may have limited memory, metered connectivity, sudden power loss, long offline periods, and a hardware or software root of trust.

EFUP defines:

- how a device identifies itself and asks whether an update applies;
- how an authority describes and signs a release;
- how a device downloads a full or delta artifact resumably;
- how authenticity, integrity, freshness, and rollback resistance are enforced;
- how installation and activation remain safe across power loss;
- how post-boot health is confirmed or automatically rolled back;
- how rollout policy, cancellation, telemetry, and failures are represented; and
- what behavior is required for interoperability and conformance.

The protocol does not prescribe a radio, IP stack, operating system, flash layout, cloud vendor, or fleet-management user interface. It also does not make an insecure boot chain secure: devices are expected to verify executable code from an immutable or independently protected trust anchor.

1.1 Design goals

1. **Fail-safe operation.** An interrupted or invalid update cannot permanently prevent the device from booting its last confirmed image.
2. **End-to-end authorization.** Artifact acceptance depends on the signed manifest, not solely on transport security or repository trust.
3. **Constrained-device viability.** Transfer can resume by chunk and verification can be streamed with bounded RAM.
4. **Operational control.** Rollouts can be staged, paused, expired, superseded, and observed without weakening device-side checks.
5. **Deterministic behavior.** A device presented with the same trusted inputs makes the same applicability and security decisions.

1.2 Out of scope

Factory provisioning, application data migration semantics, physical debug-port security, and organization-specific release approval are out of scope. A device may update applications, configuration, radio firmware, or other components using this protocol, but component-specific installation behavior remains a platform responsibility.

2 Conventions and terminology

Term	Meaning
Artifact	Immutable byte sequence installed on a device, such as a full image, delta patch, bootloader, or component bundle.
Campaign	Server-side policy that assigns a release to an eligible population over time.
Device agent	Trusted device software that evaluates manifests, transfers artifacts, invokes the installer, and reports status.
Digest	SHA-256 hash unless the manifest explicitly selects another mandatory algorithm from a later protocol revision.
Image security version	Monotonically increasing integer used by the secure boot chain to prevent rollback.
Manifest	Canonically encoded, signed release metadata that binds artifact bytes to target constraints and installation policy.
Release	A manifest and the immutable artifacts it authorizes.
Repository	Service that returns artifact bytes. It need not be trusted for authenticity because bytes are verified end to end.
Update service	Authority that evaluates campaign policy and returns a signed manifest or a no-update response.
Confirmed image	Image that passed post-boot health checks and is accepted as the recovery baseline.
Trial image	Newly activated image that has not yet been confirmed.
Trust store	Protected set of accepted signing keys, key identifiers, roles, validity metadata, and revocation state.

All sizes are in octets. Unsigned integers use network byte order when a transport binding exposes them as raw binary. Timestamps are UTC strings in RFC 3339 form with a trailing `Z`; devices compare them only when they have a trustworthy time source. A semantic version is informational unless a rule explicitly uses it; security ordering always uses integer counters.

3 System model and trust boundaries

3.1 Roles

Role	Primary responsibility	Security assumption
Release signer	Approves canonical manifests using an offline or tightly controlled signing key.	Private signing material is never present on the device or public repository.
Update service	Selects campaigns and delivers signed manifests.	May affect availability and targeting, but cannot authorize unapproved bytes.
Repository/CDN	Serves immutable artifact ranges.	Is untrusted for integrity and may return stale, truncated, or malicious content.
Device agent	Enforces applicability and security policy; controls installation.	Executes within the device's trusted computing base.
Bootloader	Verifies the bootable image and manages trial/confirmed selection.	Protected from ordinary firmware modification.
Fleet operator	Starts, pauses, cancels, and monitors campaigns.	Authenticated and authorized by the fleet-management control plane.

3.2 Minimum device capabilities

A conforming device **MUST** provide:

- a stable, non-secret `device_id` and an immutable or protected `model` identifier;
- a cryptographically secure SHA-256 implementation;
- verification for the required signature suite in Section 9;
- protected storage for trust anchors and the highest accepted security version;
- an installation method that preserves a bootable confirmed image after interruption;
- a source of cryptographically strong randomness when TLS or nonce generation requires it; and
- durable storage sufficient for protocol state or a way to reconstruct it safely.

3.3 Threat model

The adversary may observe, delay, replay, reorder, truncate, or replace network traffic; compromise a repository; know device identifiers and installed versions; cause repeated network or power interruption; and attempt to install older valid firmware. EFUP also limits the impact of a compromised online update service by requiring offline-authorized manifests.

The base protocol does not protect a device after compromise of its manifest signing key, immutable boot code, hardware trust anchor, or active firmware with arbitrary access to protected storage. Key rotation, threshold signing, measured boot, and remote attestation can reduce those risks but are deployment profiles.

4 Identifiers, versions, and persistent state

4.1 Identifiers

Name	Syntax	Requirement
`device_id`	1–128 printable ASCII characters	Unique within a fleet; opaque to protocol logic.
`model`	1–64 characters matching `[A-Za-z0-9._-]+`	Bound to hardware or immutable provisioning data.
`release_id`	UUID string	Globally unique per signed release.
`campaign_id`	UUID string	Identifies rollout policy; not trusted for artifact authorization.
`artifact_id`	UUID string	Uniquely identifies immutable bytes and transfer metadata.
`session_id`	128-bit random value encoded as 32 lowercase hex digits	Unique per discovery-to-terminal update attempt.
`key_id`	1–64 ASCII characters	Selects a trust-store verification key.

Identifiers are case-sensitive. Services **MUST** treat identifiers as opaque and **MUST NOT** embed credentials or personally identifying data in URLs, logs, or MQTT topics. A device **MUST NOT** derive authorization solely from a mutable model string reported by application firmware.

4.2 Persistent state

The following values **MUST** survive power loss using atomic records, journaling, or an equivalent mechanism:

- the highest confirmed `image_security_version` per protected component;
- active, confirmed, and trial slot identifiers;
- current `release_id`, `artifact_id`, manifest digest, and session ID;
- verified download offset or verified chunk bitmap;
- remaining trial boot attempts and confirmation deadline basis; and
- the most recent status sequence number for the active session.

A persistent update record **MUST** include an integrity check and a monotonically advancing generation number. If no valid record can be recovered, the device **MUST** boot the independently verified confirmed slot and restart discovery; it **MUST NOT** activate partially written content.

5 Protocol overview

EFUP is a device-initiated, manifest-driven protocol. A normal exchange is:

```
Device                  Update Service        Repository        Bootloader
  |-- DISCOVER ------------>|                     |                 |
  |<-- OFFER(manifest) ------|                      |                 |
  |-- ACCEPT --------------->|                      |                 |
  |-- GET Range ---------------------------------->|                 |
```

```
  |<-- artifact bytes ----------------------------|                |
  |-- STATUS(verified) ----->|                    |                |
  |-- install to inactive slot                    |                |
  |-- STATUS(staged) ------->|                    |                |
  |-- set trial boot ---------------------------------------------->|
  |                        reboot                                   |
  |<---------------------------------------------- trial image ---|
  |-- health checks                                              |
  |-- confirm image ------------------------------------------->|
  |-- STATUS(confirmed) ---->|                    |                |
```

Every offer is independently verifiable. Transfer and installation may span many connections. Status delivery is at least once, so the service **MUST** process it idempotently using `device_id`, `session_id`, and `sequence`.

5.1 Device state machine

State	Entry action	Normal next state
Idle	Wait for schedule, operator request, or notification.	Discovering
Discovering	Send inventory and capabilities; validate response envelope.	Offered or Idle
Offered	Verify signature, freshness, applicability, policy, and storage.	Downloading, Deferred, or Failed
Downloading	Fetch ranges, verify chunks, and persist progress.	Verifying or Paused
Verifying	Verify full artifact digest and authenticated metadata.	Staging or Failed
Staging	Write/transform inactive component and verify installed image.	Ready or Failed
Ready	Wait for activation policy and create atomic trial-boot record.	Rebooting
Rebooting	Reboot into the trial image.	Trial
Trial	Run health checks before the attempt/deadline limit.	Confirmed or Rollback
Confirmed	Commit security counter and report terminal success.	Idle
Rollback	Select confirmed image, reboot, and report failure.	Idle
Paused	Preserve verified progress while policy or connectivity blocks work.	Downloading or Idle
Deferred	Record a policy reason and retry at/after the advised time.	Discovering
Failed	Clean unsafe staging state and report a terminal error.	Idle

Security failures such as an invalid signature, digest mismatch, target mismatch, or rollback attempt are terminal for that manifest. Transient transport and power conditions are resumable

and **MUST NOT** be reported as terminal until configured retry limits are exhausted.

6 Common message envelope

Protocol control messages use canonical CBOR in production. Diagnostic examples in this document use the deterministic JSON mapping: byte strings are lowercase hex, integers remain integers, and map keys remain strings. A receiver **MUST** enforce maximum nesting depth 12, maximum map entries 128, and maximum decoded control-message size 64 KiB unless a stricter device profile applies.

Each request or event contains:

Listing 1: Common logical envelope

```
{
  "protocol": "EFUP",
  "protocol_version": 1,
  "message_type": "DISCOVER",
  "message_id": "9e6d17e5-8fb0-4e22-b18c-38b61e381b52",
  "device_id": "edge-dhaka-004271",
  "sent_at": "2026-08-02T06:20:00Z",
  "body": { }
}
```

Field	Required	Semantics
`protocol`	Yes	Literal `EFUP`.
`protocol_version`	Yes	Integer major version; this specification uses 1.
`message_type`	Yes	Uppercase message name defined by this document.
`message_id`	Yes	UUID used for deduplication and trace correlation.
`device_id`	Device messages	Stable device identifier.
`sent_at`	When trusted time exists	Sender's UTC creation time.
`body`	Yes	Message-specific map.

The recipient **MUST** reject an unsupported major version with `UNSUPPORTED\_PROTOCOL`. Duplicate requests with the same `message_id` and identical authenticated content **SHOULD** receive an equivalent response. Reuse of a message ID with different content **MUST** be rejected as `MESSAGE\_ID\_CONFLICT`.

7 Discovery and update selection

7.1 DISCOVER request

The device sends `DISCOVER` at boot after network readiness, on its periodic schedule, after an authenticated notification, or on a local operator request. The body contains:

```
{
  "session_id": "4df0b956e28d41cd8e5cb5849b84906a",
  "model": "gw-x2-revB",
  "hardware_revision": "B3",
  "bootloader_version": "2.4.1",
```

```
  "agent_version": "1.3.0",
  "components": [
    {
      "name": "system",
      "version": "3.7.2",
      "image_security_version": 42,
      "digest": "4b1c...9a20"
    }
  ],
  "capabilities": {
    "artifact_formats": ["raw", "squashfs"],
    "compression": ["none", "zstd"],
    "delta": ["bsdiff-v1"],
    "max_chunk_size": 262144,
    "installation": "ab-slots",
    "signature_suites": ["COSE_EdDSA_Ed25519"]
  },
  "conditions": {
    "power_source": "external",
    "battery_percent": 100,
    "network": "ethernet",
    "metered": false,
    "free_staging_bytes": 536870912
  }
}
```

The device **MUST** accurately report security-relevant capabilities and **MUST NOT** claim a format or algorithm that it cannot safely install. The service may use operational conditions for selection, but the signed manifest remains the sole authority for release content and mandatory device-side constraints.

7.2 Discovery responses

The update service returns exactly one of:

- `NO\_UPDATE`: no applicable campaign, with `check_after`;
- `OFFER`: a signed manifest and campaign metadata;
- `DEFER`: a transient service condition, with bounded `retry_after`; or
- `ERROR`: a permanent request problem.

A `NO\_UPDATE` or `DEFER` response **MUST NOT** cause the device to remove a confirmed image. A server-provided delay is advisory but **SHOULD** be honored within local safety policy. The device **SHOULD** add 0–10% random jitter to fleet polling and **MUST** cap any server delay to seven days unless locally configured.

7.3 Offer campaign metadata

Campaign metadata includes `campaign_id`, `offered_at`, optional `not_before`, and optional `activation_window`. It is authenticated by TLS but does not modify signed manifest semantics. If campaign metadata and the manifest conflict, the stricter safe behavior wins. An offer never overrides device-side power, thermal, occupancy, maintenance, or regulatory safeguards.

8 Signed manifest

8.1 Canonical form and signature

The signed payload is a canonical CBOR map. It is wrapped in a COSE_Sign1 structure whose protected headers contain the algorithm and `kid`. The signature covers the exact canonical payload bytes. Gateways and repositories **MUST NOT** rewrite, normalize, or reserialize a signed payload.

The device validates the manifest in this order:

1. parse with strict size, type, duplicate-key, and canonical-encoding checks;
2. select a trusted key by protected `kid` and authorized signer role;
3. verify the COSE signature over the original payload bytes;
4. verify protocol and manifest schema versions;
5. enforce validity, revocation, target, dependency, and anti-rollback rules;
6. select exactly one compatible artifact for each required component; and
7. persist the manifest digest before beginning transfer.

No artifact URL, filename, content type, or server response header is security authoritative unless its value is bound into the signed manifest.

8.2 Manifest fields

Field	Req.	Description
`manifest_version`	Yes	Integer schema version; value 1.
`protocol_min/max`	Yes	Inclusive supported protocol major-version range.
`release_id`	Yes	Unique UUID for this immutable release definition.
`product`	Yes	Product namespace controlled by the signing authority.
`version`	Yes	Human-readable release version.
`issued_at`	Yes	Audit timestamp; not by itself a freshness guarantee.
`expires_at`	Profile	Last acceptance time when trusted time is required.
`sequence`	Yes	Monotonically increasing unsigned release sequence.
`targets`	Yes	Allowed models, hardware revisions, and optional device groups.
`components`	Yes	Ordered component requirements and artifact choices.
`install_policy`	Yes	Power, space, reboot, trial, and confirmation constraints.
`dependencies`	No	Required component, agent, bootloader, or schema versions.
`revokes`	No	Release or artifact IDs that must no longer be installed.
`critical`	No	Extension-field names that receivers must understand.
`release_notes`	No	Non-executable display text, URI, and locale metadata.

8.3 Artifact descriptor

Each component contains one or more artifact alternatives. An artifact descriptor **MUST** contain `artifact_id`, `format`, `size`, `sha256`, `compression`, `chunk_size`, `chunk_hashes`, and at least one HTTPS `uri`. It also contains the resulting installed-image digest and security version when downloaded bytes are compressed, packaged, or a delta.

Listing 2: Abbreviated diagnostic manifest representation

```
{
  "manifest_version": 1,
  "protocol_min": 1,
  "protocol_max": 1,
  "release_id": "80c43260-26d2-4e1a-a34b-0914c176f85e",
  "product": "acme.edge-gateway",
  "version": "3.8.0",
  "issued_at": "2026-08-01T00:00:00Z",
  "expires_at": "2026-09-01T00:00:00Z",
  "sequence": 108,
  "targets": {
    "models": ["gw-x2-revB"],
    "hardware_revisions": ["B2", "B3"]
  },
  "components": [{
    "name": "system",
    "required": true,
    "version": "3.8.0",
    "image_security_version": 43,
    "installed_sha256": "e93a...6b11",
    "artifacts": [{
      "artifact_id": "ca9f67bd-ecc4-4ec0-ac14-a56a9003a616",
      "kind": "full",
      "format": "squashfs",
      "compression": "none",
      "size": 188743680,
      "sha256": "e93a...6b11",
      "chunk_size": 262144,
      "chunk_hashes": ["2a11...af03", "..."],
      "uris": ["https://updates.example.net/artifacts/ca9f67bd"]
    }]
  }],
  "install_policy": {
    "min_battery_percent": 40,
    "require_external_power": false,
    "required_free_bytes": 377487360,
    "activation": "maintenance_window",
    "max_trial_boots": 3,
    "confirmation_timeout_seconds": 300
  },
  "critical": []
}
```

The ellipses above abbreviate hashes and arrays for readability; an encoded manifest **MUST** contain complete 32-octet SHA-256 values and every chunk hash.

8.4 Applicability rules

An offered manifest is applicable only when all of the following are true:

1. the signature and signer role are valid;
2. the protocol version lies within `protocol_min` and `protocol_max`;
3. product, model, and hardware revision match the signed target expression;
4. all critical fields and required components are understood;
5. dependencies are satisfied before the dependent component is activated;
6. each selected artifact matches a reported capability;
7. its release sequence and image security version satisfy Section 10;
8. the release and artifact are not present in signed revocation metadata; and
9. installation constraints can be satisfied without risking the confirmed image.

Target expressions are positive allow-lists. Empty target lists match nothing. Wildcards, regular expressions, and negative matches are forbidden in version 1 to keep evaluation deterministic. Group targeting is an operational filter and **MUST** be backed by authenticated provisioning state if used as a signed target.

9 Cryptographic profile and key management

9.1 Required algorithms

All version-1 implementations **MUST** implement SHA-256 and COSE_Sign1 with Ed25519 (COSE algorithm EdDSA). An implementation **MAY** also support ECDSA P-256 with SHA-256 (ES256), but a deployment using ES256 **MUST** define deterministic, strict signature-encoding validation. TLS 1.3 is preferred; TLS 1.2 is permitted only with authenticated encryption, forward-secret key exchange, certificate validation, and obsolete algorithms disabled.

Digest comparison **MUST** be constant-time where practical. Devices **MUST** reject non-canonical CBOR, duplicate map keys, unsupported critical COSE headers, unprotected algorithm identifiers, malformed public keys, and signatures of the wrong length before installation.

9.2 Trust-store requirements

Each trust-store record contains `key_id`, public key, algorithm, signer role, product scope, activation constraint, and revocation state. The trust store **MUST** be integrity protected and updated only by an already trusted key with the `root` or `key-management` role. A release key **MUST NOT** authorize new root keys unless the deployment explicitly grants that capability.

Key rotation uses an overlap period:

1. an existing key signs a key-set update containing the new key and scope;
2. devices durably install the new record and acknowledge its digest;
3. releases are signed by the new key after sufficient fleet adoption; and
4. a later signed key-set update revokes the old key.

A device with unreliable wall-clock time **MUST** use signed monotonic key-set versions rather than certificate dates for trust-store rollback resistance. Emergency revocation data **SHOULD** be small, independently signed, monotonically versioned, cached, and checked before accepting a new manifest.

9.3 Transport authentication

HTTPS endpoints **MUST** authenticate the server name using a platform trust store or deployment pin set. Managed deployments **SHOULD** use mutual TLS or an application-layer device credential. Bearer credentials **MUST** be short-lived, audience-restricted, sent only in an authorization header, and never included in artifact URIs. Transport authentication protects confidentiality, fleet metadata, and service availability; it does not replace manifest verification.

10 Anti-rollback and freshness

Each protected component has an unsigned `image_security_version`. Before staging, the device **MUST** require:

$$\textit{candidate security version} \geq \textit{highest confirmed security version}.$$

If equal security versions are permitted for servicing releases, the signed `sequence` must be strictly greater than the highest accepted sequence for that product and update channel. A production profile **SHOULD** require a strictly greater security version whenever a release fixes a security vulnerability.

The device commits a higher hardware monotonic counter only after the candidate passes cryptographic verification and immediately before or during confirmation, according to platform recovery semantics. It **MUST NOT** advance the only available counter before a recoverable boot path exists. A rollback to the previous image during trial is allowed even when that image has a lower candidate version, because it is the previously confirmed recovery image. After the new image is confirmed, that exception ends.

Where trusted time exists, the device **MUST** reject a manifest after `expires_at`. Without trusted time, it **MUST** enforce signed release sequence, security version, and revocation state, and **SHOULD** obtain authenticated time before accepting an initial release after factory reset. Resetting user data **MUST NOT** erase rollback counters or trust-store versions.

11 Artifact transfer

11.1 Transport-independent requirements

Artifacts are immutable and content-addressed by the signed digest. A repository **MUST** support byte-range retrieval or whole-object retrieval. The device **MUST**:

- verify that the returned length and range agree with the request;
- reject data beyond the signed artifact size;
- hash every received chunk before marking it durable and verified;
- calculate the full artifact digest over bytes in artifact order;
- never execute or parse unverified data with privileged, unsafe code; and

- erase or quarantine staging content after a terminal integrity failure.

11.2 Chunking and resume

`chunk_size` **MUST** be a power of two from 16 KiB through 1 MiB. All chunks except the last have exactly that size. Chunk index i covers:

$$[i \times \mathit{chunk\_size}, \min((i+1) \times \mathit{chunk\_size}, \mathit{artifact\_size})).$$

The ordered `chunk_hashes` array contains SHA-256 for every chunk. A device persists progress only after data reaches durable storage and its chunk hash matches. After reboot it **MUST** revalidate the manifest digest and staging-record integrity before trusting the bitmap. It **SHOULD** re-hash the last verified chunk to detect torn writes.

When a repository changes ETag, total size, or range semantics during a session, the device **MUST** stop and validate already stored content against signed chunk hashes; it may resume only if all retained bytes remain valid. HTTP cache headers never authorize content changes.

11.3 Full and delta artifacts

The device should prefer the artifact with the lowest safe total cost, accounting for transfer size, scratch space, energy, and failure recovery. A delta artifact is eligible only if its signed `base\_installed\_sha256` exactly matches the installed base image. After applying a delta, the device **MUST** verify both the patch artifact digest and the signed `installed\_sha256` of the produced image. On delta failure it **SHOULD** fall back to a compatible full artifact from the same manifest; it **MUST NOT** accept an unsigned alternative.

Decompression and patching **MUST** enforce declared output-size limits, reject trailing unauthorized content, avoid path traversal and symbolic-link escape, and operate within an unprivileged or otherwise isolated installer when the platform supports it.

11.4 Retry behavior

For timeouts, connection loss, HTTP 429, and HTTP 5xx, the device uses exponential backoff with full jitter:

$$d_n \sim U(0, \min(d_{\max}, d_0 2^n)),$$

where the recommended d_0 is 5 seconds and $d_{\max}$ is 15 minutes. An authenticated `Retry-After` value may raise the delay but cannot exceed the local cap. HTTP 401/403 triggers credential refresh at most once before terminal authorization failure. HTTP 404/410 is terminal for that URI; the device may try another signed URI. Integrity failures **MUST NOT** be retried indefinitely and **SHOULD** trigger a security alert after two independent retrieval attempts.

12 Transport bindings

12.1 HTTPS binding

The default binding uses HTTPS with `Content-Type: application/cbor`.

Operation	Method	Path
Discover	POST	`/efup/v1/devices/\{device_id\}/discover`
Accept offer	POST	`/efup/v1/devices/\{device_id\}/sessions/\{session_id\}/accept`
Status event	POST	`/efup/v1/devices/\{device_id\}/sessions/\{session_id\}/status`
Artifact bytes	GET	Signed absolute URI from the manifest, with optional `Range: bytes=start-end`.

Successful control responses use HTTP 200; an accepted asynchronous status event may use 202. A no-update result is a protocol body with HTTP 200, not HTTP 404. Malformed input uses 400, authentication failure 401, authorization failure 403, conflict 409, rate limiting 429, and transient service failure 503. Control responses **MUST** include `Cache-Control: no-store`. Artifact responses may be cached if immutable and keyed by artifact identity.

12.2 MQTT binding

MQTT 5 deployments use QoS 1 for control messages and status. Recommended topics are:

```
efup/v1/devices/{device_id}/discover       device -> service
efup/v1/devices/{device_id}/offer          service -> device
efup/v1/devices/{device_id}/status         device -> service
efup/v1/devices/{device_id}/command        service -> device
```

Topic authorization **MUST** restrict a device to its own identifier. Retained offer messages are forbidden unless they carry an unambiguous release and expiration policy and the device still performs every manifest check. Commands are hints to poll, pause, resume, or cancel; they **MUST NOT** contain unsigned replacement artifact metadata. Large artifacts should be transferred over HTTPS even when control messages use MQTT.

13 Offer acceptance and policy evaluation

After validation, the device sends `ACCEPT` with session, release, selected artifact IDs, current conditions, and an estimated start time. Acceptance means the device intends to proceed; it does not guarantee activation. The device sends `DECLINE` instead when the release is valid but cannot be applied, using a reason such as `INSUFFICIENT\_SPACE`, `POLICY\_BLOCKED`, `UNSUPPORTED\_ARTIFACT`, or `DEPENDENCY\_UNSATISFIED`.

Before download and again before staging and activation, the device evaluates:

- battery percentage, external power, and estimated energy reserve;
- staging and recovery storage, including worst-case temporary expansion;
- device temperature and platform thermal limits;
- network type, metering, roaming, and local data budget;
- maintenance window, user activity, and process safety interlocks; and
- campaign cancellation or signed release revocation learned since acceptance.

A temporary policy failure moves the session to Deferred or Paused without discarding verified bytes. A permanent incompatibility is terminal. Safety-critical interlocks always take precedence

over campaign timing.

14 Installation and atomic activation

14.1 A/B slot profile

The recommended profile has an immutable first-stage bootloader and two bootable slots. At all times, at least one slot is marked confirmed. The installer writes only the inactive slot, reads it back, verifies the signed installed-image digest, flushes all writes, then atomically creates trial metadata containing:

- target slot and release ID;
- expected installed-image digest and security version;
- maximum trial boots and remaining boot count;
- confirmation timeout; and
- previous confirmed slot.

The bootloader verifies the target image signature or digest against protected metadata before every trial boot. It decrements the durable trial counter before transferring control. If verification fails, the counter reaches zero, the image resets before confirmation, or the watchdog expires, the bootloader selects the previous confirmed slot.

14.2 Single-slot profile

A single-slot device is conforming only if it has an independently bootable recovery environment that can restore a complete signed image after arbitrary power loss. In-place overwrite without a recovery path is non-conforming. The recovery environment **MUST** use the same signature, targeting, digest, and anti-rollback checks as the normal agent.

14.3 Multi-component transactions

The manifest defines component installation order and compatibility dependencies. When components must change atomically, the platform **MUST** stage all of them before activation and commit a single transaction record. On failure it rolls back the complete compatibility group. If hardware prevents atomic rollback, the manifest **MUST** encode a safe forward-compatible order, and each intermediate combination must be explicitly supported.

Bootloader updates require a dedicated platform profile. At minimum they use redundant boot blocks or immutable recovery code, write protection, independent read-back verification, and a power-loss-safe selector. An ordinary system image **MUST NOT** directly rewrite the only boot block.

15 Trial boot, health, and confirmation

The trial image starts a confirmation controller as early as practical. It reports `TRIAL\_STARTED` only after the network stack is available, but network availability alone is not a sufficient health signal.

The minimum health gate includes:

- verified boot and required filesystem mounts;
- successful schema/data migration or verified backward-compatible state;
- watchdog stability for the configured observation interval;
- startup of all mandatory services and device drivers;
- access to critical sensors, actuators, and secure storage; and
- no platform-defined fatal diagnostic or safety condition.

Cloud reachability **SHOULD** be a confirmation requirement only when the device can reliably distinguish a firmware defect from an external outage. Otherwise the device confirms locally and reports when connectivity returns.

Confirmation is an atomic boot-control operation. The device **MUST** first ensure the trial slot is fully verified, then mark it confirmed, commit applicable security counters, and finally allow the old slot to become reusable. A reset at any point must yield either the old confirmed image or the new confirmed image, never an unverified slot.

If health fails, the agent should record a compact durable diagnostic before requesting rollback. The bootloader performs rollback even if the trial agent is nonfunctional. After booting the recovery image, the agent reports `ROLLED\_BACK` with the failed release ID and reason.

16 Status reporting and observability

16.1 STATUS message

Status events are monotonic within a session. Each body includes:

```
{
  "session_id": "4df0b956e28d41cd8e5cb5849b84906a",
  "sequence": 17,
  "release_id": "80c43260-26d2-4e1a-a34b-0914c176f85e",
  "state": "DOWNLOADING",
  "progress": {
    "bytes_verified": 104857600,
    "bytes_total": 188743680,
    "artifact_id": "ca9f67bd-ecc4-4ec0-ac14-a56a9003a616"
  },
  "device_time": "2026-08-02T06:44:10Z",
  "boot_id": "541790e7-3585-44f3-b88c-bec9528bc242"
}
```

Required state events are `OFFER\_ACCEPTED`, `DOWNLOAD\_STARTED`, `DOWNLOAD\_VERIFIED`, `STAGED`, `ACTIVATION\_SCHEDULED`, `TRIAL\_STARTED`, `CONFIRMED`, `ROLLED\_BACK`, and `FAILED`. Devices **SHOULD** report progress no more frequently than every 30 seconds or each 5% increase, whichever is less frequent, to avoid fleet storms.

The service acknowledges the highest contiguous status sequence received. Devices retain terminal status until acknowledged or until a locally defined minimum retention of seven days. Duplicate events are normal. A service **MUST NOT** infer success from silence, download completion, or reboot; only `CONFIRMED` is terminal success.

16.2 Privacy and log hygiene

Logs may contain device, release, campaign, artifact, error, duration, and byte counts. They **MUST NOT** contain bearer tokens, private keys, raw device credentials, full signed URLs, or unredacted user data. Operators **SHOULD** apply retention and access controls appropriate to device identifiers and location-sensitive network metadata. Error detail sent to the service must be bounded and structured, not an unrestricted memory dump.

17 Commands, cancellation, and supersession

The service may send authenticated `POLL`, `PAUSE`, `RESUME`, or `CANCEL` commands. Commands include a unique ID, campaign ID, issue time, optional expiry, and reason. The device acknowledges a command idempotently.

- `POLL` triggers discovery; it does not directly authorize a release.
- `PAUSE` stops new transfer work at the next chunk boundary and preserves progress.
- `RESUME` allows policy evaluation and transfer to continue.
- `CANCEL` prevents future staging or activation for the campaign.

Cancellation cannot safely interrupt flash erase/program operations, trust-store updates, or an activation commit. The device completes the current atomic step, then reaches a safe state. After a trial boot, ordinary cancellation does not force rollback; only local health policy or a separately signed emergency revocation can do so.

A newer offer supersedes an active one only when it is valid, targets the device, has a greater signed sequence, and local policy permits switching. Verified bytes may be reused only when their artifact ID, size, digest, and chunk parameters are identical in both signed manifests.

18 Error model

Errors contain `code`, `category`, `retryable`, optional `retry_after`, and a bounded diagnostic `detail`. Devices **MUST** base retry behavior on the code and authenticated context, not free-form detail text.

Code	Retryable	Meaning / required action
`UNSUPPORTED\_PROTOCOL`	No	No compatible major protocol version.
`AUTHENTICATION\_FAILED`	After refresh	Refresh credentials once; do not expose credential data.
`MANIFEST\_MALFORMED`	No	Reject before processing nested content.
`SIGNATURE\_INVALID`	No	Quarantine offer and emit a security event.
`SIGNER\_UNTRUSTED`	No	Key absent, revoked, inactive, or outside authorized scope.
`MANIFEST\_EXPIRED`	No	Obtain a newly signed manifest.
`TARGET\_MISMATCH`	No	Product, model, hardware, or group does not match.
`ROLLBACK\_BLOCKED`	No	Candidate violates security-version or sequence policy.

Code	Retryable	Meaning / required action
UNSUPPORTED_ARTIFACT	No	No safe compatible alternative is present.
DEPENDENCY_UNSATISFIED	Conditional	Rediscover after dependency installation.
INSUFFICIENT_SPACE	Conditional	Retry after safe cleanup; never erase confirmed slot.
POLICY_BLOCKED	Yes	Wait until power, network, time, or safety condition changes.
TRANSFER_FAILED	Yes	Resume with bounded exponential backoff.
CHUNK_HASH_MISMATCH	Limited	Refetch from another URI; alert after repeated mismatch.
ARTIFACT_HASH_MISMATCH	Limited	Discard artifact; do not stage.
INSTALL_FAILED	Platform-specific	Keep confirmed image and collect bounded diagnostics.
HEALTH_CHECK_FAILED	No	Roll back trial image.
TRIAL_TIMEOUT	No	Bootloader rolls back after deadline/attempt limit.
COMMAND_EXPIRED	No	Ignore command and preserve safe state.
INTERNAL_ERROR	Yes	Back off; avoid reboot loops and fleet storms.

Error codes are stable API values. New codes may be added; an unknown code is handled according to its authenticated `retryable` flag, with a conservative local retry cap. Security categories always override `retryable=true`.

19 Power-loss and recovery invariants

A conforming implementation demonstrates the following invariants under power loss at every storage operation:

1. at least one cryptographically verified, confirmed image remains bootable;
2. a partially received artifact is never treated as fully verified;
3. a partially written slot is never selected as confirmed;
4. boot-selection records resolve deterministically using validity and generation;
5. security counters never decrease and do not strand the only recovery image;
6. trial attempts are bounded even when resets occur before application startup; and
7. resumption never combines bytes authorized by different manifest digests.

Boot-control records **SHOULD** use two copies with generation, state, payload digest, and CRC or authenticated tag. The writer commits a new valid record before invalidating the old one. Flash wear limits must be considered when selecting progress checkpoint frequency; a chunk bitmap may be journaled in batches only if uncommitted chunks are safely reverified after reboot.

Recovery mode **MUST** accept only properly targeted signed images and must expose no unauthenticated command that disables secure boot, rollback prevention, or artifact verification. Physical recovery may use a deployment-specific presence test, but its trust implications must be documented.

20 Fleet rollout requirements

Campaign assignment **SHOULD** use a stable pseudorandom bucket so that reconnects do not move a device between rollout cohorts:

$$bucket = \text{uint64}(\text{HMAC}_K(campaign\_id \parallel device\_id))[0..9999].$$

The campaign includes devices whose bucket is below the current basis-point threshold and that satisfy inventory filters. The HMAC key is a service secret; the bucket controls availability, not device-side authorization.

A recommended rollout proceeds through internal devices, 1%, 5%, 25%, 50%, and 100% cohorts with an observation period at each gate. Automated halt criteria should consider confirmation rate, rollback rate, health-check failures, boot loops, installation failures, and device-specific safety signals. Success denominators must exclude devices that never accepted an offer and separately show deferred/offline populations.

The service must support pause and cancellation, but devices remain responsible for safe atomic boundaries. Rollout controls **MUST NOT** bypass signed targeting, anti-rollback, or health confirmation. Emergency deployment may shorten timing; it may not weaken cryptographic acceptance.

21 Security considerations

21.1 Required controls

Implementations **MUST** address:

- **Repository compromise:** signed manifests, artifact and chunk digests, immutable identifiers, and strict size enforcement;
- **Replay and freeze:** security versions, signed release sequence, expiration where trusted time exists, and revocation metadata;
- **Target confusion:** signed product/model/hardware allow-lists and protected device identity;
- **Parser attacks:** canonical encoding, bounded allocation, duplicate-key rejection, strict types, and fuzz testing;
- **Decompression/patch attacks:** output bounds, isolated processing, base/result digest checks, and path confinement;
- **Power interruption:** atomic records, inactive-slot installation, read-back verification, and independent rollback;
- **Credential leakage:** TLS, scoped credentials, sanitized telemetry, and no credentials in URLs; and
- **Fleet-wide failure:** stable staged cohorts, rate limits, jitter, halt thresholds, and bounded device retries.

21.2 Denial of service

The device should authenticate small control objects before reserving large storage or performing expensive transformations. It must cap message sizes, redirects, URI count, retry count, concurrent

transfers, decompressed size, patch operations, and log growth. It should reject cyclic dependencies and manifests with excessive component or artifact alternatives.

An attacker able to block all connectivity can prevent updates; no network protocol can guarantee availability in that case. Devices should retain safe autonomous behavior and surface prolonged update unavailability when connectivity returns.

21.3 Signing operations

Release pipelines should produce reproducible artifacts, generate a software bill of materials, scan and test the exact bytes to be signed, require independent approval for production, and keep online services separated from signing keys. Signing logs should be append-only and include manifest digest, release ID, key ID, approvers, and build provenance. Deterministic builds and transparency logs are recommended for high-assurance deployments.

22 Reference device algorithm

The following pseudocode is informative but reflects normative ordering:

```
function handle_offer(cose_bytes, campaign):
    require length(cose_bytes) <= MAX_MANIFEST
    protected, payload, signature = parse_strict_cose(cose_bytes)
    key = trust_store.lookup(protected.kid, protected.alg)
    require key.role permits RELEASE for local_product
    require verify_cose(key, protected, payload, signature)

    manifest = decode_canonical_cbor(payload)
    require supported_schema(manifest)
    require target_matches(manifest.targets, protected_identity())
    require not revoked(manifest, current_revocations)
    require freshness_and_rollback_ok(manifest, protected_counters())

    selection = choose_compatible_artifacts(manifest, inventory())
    require selection covers every required component
    require install_constraints_can_eventually_be_met(manifest)

    session = persist_new_session(
        release_id = manifest.release_id,
        manifest_digest = sha256(payload),
        selection = selection)
    send_accept(session)

    for artifact in selection:
        while not all_chunks_verified(artifact):
            require current_manifest_digest(session) == sha256(payload)
            wait_until_transfer_policy_allows()
            chunk = fetch_next_range(artifact)
            require sha256(chunk.bytes) == artifact.chunk_hashes[chunk.index]
            durably_store_then_mark_verified(chunk)
        require sha256(stored_artifact_bytes) == artifact.sha256

    wait_until_install_policy_allows()
    stage_into_inactive_target(selection)
    require verify_installed_images(manifest.components)
    atomically_schedule_trial_boot(manifest.install_policy)
    reboot()
```

```
function on_trial_boot():
    if required_health_checks_pass_before_limit():
        atomically_confirm_and_commit_security_counters()
        report(CONFIRMED)
    else:
        persist_bounded_failure_reason()
        request_rollback_and_reboot()
```

23 Conformance requirements

23.1 Implementation classes

Class	Required behavior
Device Core	Discovery, strict signed-manifest validation, SHA-256, Ed25519, full-artifact HTTPS transfer, resume, safe installation, anti-rollback, trial confirmation/rollback, and status events.
Device Delta	Device Core plus at least one declared delta format with base and result verification and safe fallback.
Update Service	Idempotent discovery/status processing, compatible offer selection, campaign controls, bounded retry advice, and protocol-version handling.
Repository	Immutable object and correct whole/range retrieval with stable size.
Boot Platform	Verified boot, protected counter storage, atomic trial/confirmed selection, bounded attempts, watchdog integration, and independent rollback.

23.2 Mandatory test suite

A Device Core implementation **MUST** pass all applicable tests below. Power-failure tests inject reset before and after every persistent write and flash erase/program boundary.

ID	Test	Expected result
C-001	Valid full update	Device downloads, verifies, stages, boots, confirms, and reports success.
C-002	Unknown optional field	Field is ignored without changing signed semantics.
C-003	Unknown critical field	Manifest is rejected before transfer.
C-004	Modified manifest byte	Signature verification fails.
C-005	Unknown/revoked signer	Manifest is rejected as untrusted.
C-006	Wrong model/hardware	Manifest is rejected as target mismatch.
C-007	Older security version	Candidate is rejected; confirmed image remains.
C-008	Duplicate CBOR key	Manifest is rejected as malformed.
C-009	Oversized control object	Object is rejected without memory exhaustion.
C-010	Interrupted chunk	Partial chunk is not marked verified; transfer resumes.
C-011	Corrupt chunk	Chunk hash fails; bytes are discarded and bounded retry occurs.

ID	Test	Expected result
C-012	Corrupt complete artifact	Full digest fails and staging is forbidden.
C-013	Repository changes object	Signed size/digest controls; invalid bytes are rejected.
C-014	Power loss during download	Device boots confirmed image and safely resumes.
C-015	Power loss during slot write	Partial target is never selected for boot.
C-016	Power loss during activation record	Boot selection resolves to a valid slot.
C-017	Trial image boot loop	Bootloader rolls back within configured attempt limit.
C-018	Health-check failure	Trial is not confirmed; recovery image boots.
C-019	Power loss during confirmation	Exactly one valid confirmed outcome is recovered.
C-020	Status redelivery	Service deduplicates; state does not regress.
C-021	Pause/cancel at flash boundary	Atomic operation completes, then device reaches safe state.
C-022	Delta wrong base	Delta is rejected and signed full alternative is selected.
C-023	Decompression expansion attack	Declared output bound stops processing safely.
C-024	TLS/authentication failure	No unauthenticated offer or artifact is trusted.
C-025	Factory reset after upgrade	Protected anti-rollback state is retained.

23.3 Acceptance evidence

Conformance evidence includes protocol traces with secrets removed, manifest test vectors, signature verification results, flash fault-injection coverage, boot-slot transition logs, counter behavior, and results for each mandatory test. A platform claim must name the hardware revision, bootloader version, agent version, supported artifact formats, maximum tested artifact size, and cryptographic library versions.

24 Operational checklist

24.1 Before release signing

☐ Exact production artifacts are reproducible, scanned, and integration-tested.

☐ Product, model, hardware revisions, dependencies, and security version are correct.

☐ Full and installed-image digests, sizes, and every chunk hash are regenerated.

☐ Recovery and downgrade implications have been reviewed.

☐ Signing key role, scope, validity, and revocation status are correct.

☐ A compatible full artifact exists when a delta is offered.

☐ Rollout gates, observation periods, halt criteria, and support runbook are approved.

24.2 Before fleet expansion

☐ Confirmation, rollback, install-failure, and offline rates are within thresholds.

☐ Health and safety metrics are compared with the pre-release baseline.

☐ Status deduplication and denominators are verified.

☐ Repository capacity, range behavior, cache integrity, and regional availability are healthy.

☐ Revocation, pause, cancellation, and recovery procedures have named owners.

24.3 Incident response

When a release is suspected unsafe, operators should pause campaign expansion, preserve evidence, distinguish distribution failure from firmware failure, and decide whether cancellation, signed revocation, or a forward-fix release is safest. They must not delete repository bytes still needed for diagnosis or recovery. Devices already in trial rely on local health and rollback rules; confirmed devices require a newer signed remediation whose security version respects anti-rollback policy.

A Protocol constants

Constant	Value	Notes
Protocol name	`EFUP`	Exact case.
Protocol major version	1	Integer.
Manifest schema version	1	Integer.
Control media type	`application/cbor`	Canonical CBOR.
Required digest	SHA-256	32-octet output.
Required signature	COSE_Sign1 / Ed25519	COSE algorithm EdDSA.
Maximum control message	65,536 octets	A profile may lower it.
Maximum nesting depth	12	Includes arrays and maps.
Maximum map entries	128	Per map.
Chunk-size range	16 KiB–1 MiB	Power of two.
Recommended initial retry base	5 seconds	Full jitter.
Recommended retry cap	15 minutes	Local policy may lower it.
Recommended status interval	30 seconds / 5%	Use less frequent trigger.

B Terminal outcome mapping

Outcome	Device condition	Fleet interpretation
`CONFIRMED`	Candidate is boot-confirmed and protected counters are committed.	Success.
`ROLLED\_BACK`	Previous confirmed image is active after trial failure.	Failed release attempt; device recovered.
`FAILED`	Session ended before confirmation; safe confirmed image remains.	Failure categorized by error code.
`DECLINED`	Valid offer cannot be applied under permanent capability/policy.	Not attempted; investigate eligibility.
`CANCELLED`	Work stopped at a safe boundary before trial activation.	Operator-controlled terminal outcome.
`DEFERRED`	Temporary condition prevents progress.	Non-terminal; exclude from success denominator until attempted.

C Implementation notes

These notes are informative:

- Keep protocol parsing and signature verification in a small, fuzzed module.
- Stream hashing directly from network to staging storage to minimize RAM.
- Separate “downloaded”, “verified”, “staged”, and “confirmed” in code and telemetry; collapsing them creates unsafe recovery behavior.
- Test with real flash fault modes, including torn writes and bit errors, not only process termination in a simulator.
- Treat a boot attempt counter as security-critical state and decrement it before launching an unconfirmed image.
- Rate-limit discovery after reboot so a faulty fleet does not overwhelm the service while still preserving prompt recovery reporting.

End of EFUP 1.0 Specification