

Independent Submission
Internet-Draft
Intended status: Informational
Expires: 1 December 2026

B. El Khatabi
30 May 2026

Verifiable Telemetry Ledgers for Resource-Constrained Environments
draft-elkhatabi-verifiable-telemetry-ledgers-06

Abstract

This document specifies a verifiable telemetry ledger profile for accepted telemetry in resource-constrained sensing environments. The profile begins after local policy has admitted a device and the gateway accepts its telemetry under the active transport and anti-replay contract. From that point onward, it defines a reference framed transport profile, deterministic accepted-telemetry-to-canonical-record projection, authoritative canonical-record and day artifacts, verifier-facing manifests, disclosure classes, and the binding of day artifacts to external timestamp evidence. OpenTimestamps (OTS) is the default anchoring channel; RFC 3161 timestamp responses and peer signatures are optional parallel channels.

The goal is interoperability and independent auditability, not a full device-lifecycle system or new cryptographic primitives. Verification claims are limited by the disclosed artifacts, the claimed disclosure class, and the checks the verifier actually executes. A successful result establishes internal consistency and correct proof binding for the disclosed bundle; it does not by itself establish dataset completeness, physical truth of measurements, or safety for autonomous actuation.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 December 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	3
1.1. Scope Boundary	5
1.2. Gateway Trust Boundary	5
1.3. Relationship to Publication and Adjacent Systems	5
2. Conventions and Terminology	7
3. System Roles	8
4. Data and Commitment Model	9
4.1. Reference Framed Transport Profile	9
4.2. Anti-Replay Admission Criterion	11
4.3. Accepted-Telemetry-to-Canonical-Record Projection	13
4.4. Deterministic Concise Binary Object Representation (CBOR) Commitment Encoding	16
4.5. Deterministic Commitment Tree Calculation	17
4.6. Day Artifact Schema	19
4.7. Day Chaining	20
5. Artifacts and Verification Bundles	20
6. Anchoring and Verification	22
6.1. Anchoring Contract	22
6.2. OTS Anchoring Profile	23
6.2.1. OTS Anchoring Lifecycle	23
6.2.2. Handling Delayed or Failed Anchoring	24
6.2.3. Proof Status Vocabulary	24
6.3. Optional Parallel Attestation	24
6.4. Verification	25
7. Disclosure Classes	29
8. Versioning	32
9. Conformance Vectors	33
10. Operational Considerations	34
10.1. Gateway Time and Rollover	34
10.2. Health and Fault Management	34
10.3. Configuration Management	35
10.4. Fault-to-Verifier-Result Mapping	35
10.5. Performance and Scaling	36

10.6. Verifying Correct Operation	36
11. Security Considerations	37
12. Privacy Considerations	40
13. IANA Considerations	40
13.1. Media Types Registry	40
13.2. No CBOR Tag Allocation	43
13.3. commitment_profile_id Registry	43
14. Future Extension Points	43
15. References	43
15.1. Normative References	44
15.2. Informative References	44
Appendix A. Example Verification Manifest	46
Appendix B. Illustrative Conformance Vector Bundle	48
Appendix C. Minimal Device Requirements	50
Appendix D. Verification Manifest CDDL	51
Appendix E. Rejection Audit Record Shape	53
Acknowledgments	54
Author's Address	55

1. Introduction

Long-lived telemetry deployments need evidence that disclosed telemetry is the same telemetry a gateway committed when it was accepted, even when uplink is intermittent and verification happens later.

This profile standardizes the gateway-side commitment contract for accepted telemetry. In this document, the commitment contract is the set of deterministic rules for accepted-telemetry-to-canonical-record projection, commitment encoding, day-artifact structure, verifier-facing manifests, disclosure classes, and proof binding.

The profile begins at an explicit handoff. Before this profile applies, lifecycle or control-plane systems determine whether a device is known, onboarded, admitted, credentialed, and allowed to send. This profile begins once local policy has accepted telemetry under the active transport and anti-replay contract and then turns that telemetry into authoritative, verifier-facing evidence.

Device reporting cadence and gateway artifact cadence are distinct. A device can report at fixed or irregular intervals, but this profile uses one Coordinated Universal Time (UTC) day as the batching unit for authoritative artifacts. Other batching intervals are possible design choices for future profiles; they are not implicit variations of this one. One UTC day is used in the current profile because it gives a predictable rollover boundary, bounded artifact size, and a stable unit for later disclosure, export, and audit. The gateway, not the device, is responsible for assigning accepted records to UTC day boundaries.

The one-day artifact cadence is a gateway batching rule, not a device reporting requirement. A device can emit every few hours, every few weeks, or irregularly; if the gateway accepts those records, it assigns each record to the applicable UTC day artifact. A day artifact therefore represents the set of accepted canonical records assigned to one UTC day. This profile also defines the deterministic empty-day root for deployments that emit explicit empty day artifacts, but it does not require every deployment to publish an artifact for every UTC day.

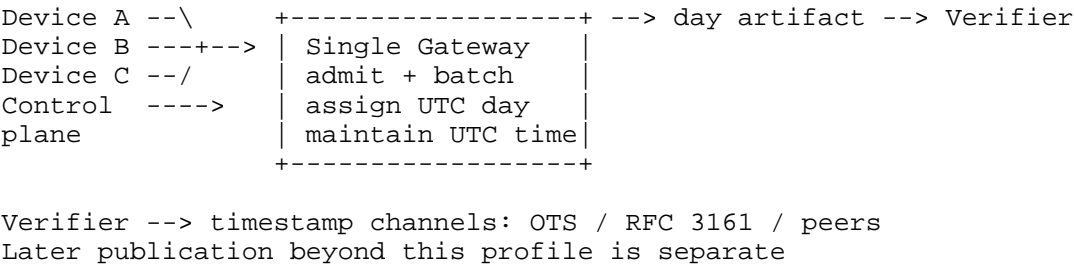


Figure 1: Reference Architecture and Trust Boundaries

Figure 1 shows the acceptance handoff and the verifier-facing trust boundary. Lifecycle systems remain responsible for whether a device is admitted; this profile begins at accepted telemetry. The lower line in the figure is not an additional ingest hop; it shows optional verifier-side timestamp-validation channels over the same day-artifact digest. The gateway is expected to maintain UTC time for ingest_time assignment and day-artifact rollover. Deployments SHOULD use an authenticated time-synchronization mechanism, such as authenticated Network Time Protocol (NTP) or an equivalent source, and SHOULD monitor clock health, since gateway clock errors propagate directly into committed ingest_time values and UTC day-boundary assignment. Behavior on detected clock step, clock regression, or loss of synchronization is a deployment policy matter and SHOULD be documented by the operator. The device is not required to keep UTC wall-clock time for this profile.

In the deployment model assumed here, intermittency primarily applies on the device-to-gateway path. External timestamping or later publication can also be delayed, but those delays do not change the accepted-telemetry commitment contract once the gateway has admitted the telemetry.

A deployment can separately publish verifier-facing manifests or exported evidence bundles through a supply chain integrity, transparency, and trust (SCITT) service only under a publication profile defined elsewhere. Such a publication profile would need to specify the submitted statement content, the submission procedure, and any verifier use of SCITT state. This profile defines none of those behaviors. Verification defined here does not consume SCITT state, and SCITT is not required to create the underlying commitment artifacts.

1.1. Scope Boundary

This profile covers processing after accepted telemetry exists:

- * transport validation, anti-replay, and canonical record admission;
- * deterministic batching into authoritative artifacts;
- * verifier-facing manifests, disclosure, and export behavior; and
- * artifact hashing, anchoring, and independent verification.

It does not specify manufacturer identity, onboarding, fleet inventory, public key infrastructure (PKI) policy, network admission enforcement, or firmware and update orchestration.

1.2. Gateway Trust Boundary

The gateway is inside the trust boundary for admission and commitment. This profile provides tamper evidence for telemetry after the gateway commits it; it does not prove ingestion completeness, physical truth of measurements, or the absence of selective pre-commitment drops by a compromised or misconfigured gateway.

1.3. Relationship to Publication and Adjacent Systems

- * Certificate Transparency ([RFC9162]): Certificate Transparency logs publish append-only records about certificate issuance. They are a useful analogy for later transparency publication and audit, but they do not define the local commitment artifacts, disclosure classes, or baseline verification behavior specified here.

- * `_Supply Chain Integrity, Transparency, and Trust (SCITT)_`
([SCITT]): SCITT publishes claims about verifier manifests or exported evidence bundles only when a deployment separately defines such publication across a transparency-service trust boundary. Verifiers defined here do not consume SCITT state.
- * `_COSE Merkle Tree Proofs_` ([COSE-MERKLE]): COSE-MERKLE defines proof encodings that future disclosure bundles could adopt.
- * `_Remote ATtestation procedureS (RATS)_` ([RFC9334]): RATS can supply attestation inputs that influence admission decisions before this profile applies.

These systems are adjacent to this profile; they do not define the accepted-telemetry commitment contract specified here.

Certificate Transparency and SCITT are both publication systems: they make externally visible statements available for later audit. This profile instead defines the local evidence contract that exists before any such publication.

More concretely, this profile differs from SCITT in that it does not require a transparency service to create or verify its baseline artifacts. It fixes the local accepted-telemetry contract itself: deterministic accepted-telemetry-to-canonical-record projection, authoritative day artifacts, verifier-facing manifests, disclosure classes, and proof binding. A deployment can later publish those outputs through SCITT only if a separate specification defines the publication semantics and any verifier processing of SCITT state. Such publication remains outside baseline verification defined here.

This profile also differs from COSE-MERKLE in scope. COSE-MERKLE can supply proof encodings for future disclosure bundles, but this profile fixes the telemetry-specific behavior around those proofs: how admitted telemetry becomes canonical commitment inputs, how daily batching and day chaining are computed, and what verifiers report about exercised scope, executed checks, and skipped checks. RATS and similar attestation systems remain pre-admission inputs; this profile begins only once accepted telemetry exists.

For architectural background on transparency-service publication and prospective Merkle proof encodings related to later disclosure workflows, see [SCITT] and [COSE-MERKLE].

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Terms:

- * **Frame:** In the reference framed transport profile, one UTF-8 JavaScript Object Notation (JSON) text serialized as a single newline-delimited JSON (NDJSON) record, containing header and authenticated encryption with associated data (AEAD; [RFC5116]) fields.
- * **Canonical record:** Canonical telemetry record admitted after gateway validation and projection. The term refers to the committed record shape, not to physical truth of the underlying measurement.
- * **Commitment profile:** The serialization, hash, and Merkle rules that produce deterministic commitment outputs.
- * **Commitment contract:** The full interoperable commitment behavior defined by this profile, including accepted telemetry admission, accepted-telemetry-to-canonical-record projection, commitment encoding, day-artifact structure, verifier-facing manifests, disclosure classes, and verification rules. The commitment profile is one component of this broader contract.
- * **Connectivity window:** The deployment-defined maximum outage or buffering interval across which a device or an adjacent trusted transport component must preserve enough durable state to retransmit or disambiguate counters without replay ambiguity. It is selected by deployment policy and sizing assumptions; it is not itself a conformance timer and does not require the device to keep UTC wall-clock time.
- * **Day artifact:** The authoritative canonical day record written as day/YYYY-MM-DD.cbor.
- * **Authoritative:** Used for the canonical artifact that verifiers MUST treat as the cryptographic source of truth.
- * **Projection:** A non-authoritative representation (for example JSON) derived from an authoritative artifact.

- * Block metadata: A standalone projection of a batch object (for example blocks/YYYY-MM-DD-00.block.json) exported for convenience. When disclosed, it MUST match the corresponding batch object in the authoritative day artifact.
- * OTS metadata sidecar: day/YYYY-MM-DD.ots.meta.json, a separate non-authoritative metadata file associated with an OTS proof and authoritative day artifact, linking the artifact digest to the proof path. It is not a commitment input.
- * Anchor evidence: The proof artifacts and binding metadata disclosed for an external timestamping or attestation channel.
- * Peer signature quorum: A deployment-defined set or threshold of peer signatures over the same day-artifact digest or day root, treated as one optional parallel attestation channel.
- * Replay unit: The deployment-defined unit consumed exactly once for replay prevention. In the reference framed transport profile, it is the pair (dev_id, fc), where fc is a frame counter for device dev_id.
- * Verification scope: The set of disclosed artifacts, checks executed, and claim boundaries that a verifier actually asserts for a verification result.
- * Day boundary: A UTC calendar day boundary. Day labels in this profile use YYYY-MM-DD in UTC.
- * Disclosure class: The level of artifact disclosure associated with a verification claim.

3. System Roles

In this document, optional parallel attestation channels are not required for baseline conformance. A conforming deployment MUST implement at least one anchoring channel, with OpenTimestamps (OTS; [OTS]) as the default channel described here. Timestamp responses under [RFC3161] and peer signatures MAY be absent entirely; when present, they MUST bind to the same authoritative day-artifact digest and verifiers MUST report their results separately.

- * Device: Produces telemetry for gateway admission. Under the reference framed transport profile, that telemetry is framed as described in Section 4.1.

- * Gateway: Applies the active transport validation and anti-replay contract, projects accepted telemetry into canonical records, batches, and anchors day artifacts.
- * Verifier: Recomputes commitments and validates proofs from disclosed artifacts.
- * OpenTimestamps calendar(s): provide OTS attestations for day artifact hashes.
- * Timestamp authority (TSA): a timestamp authority under [RFC3161] over the same digest, when that optional channel is used.
- * Peers: Co-sign daily roots for short-term provenance, when that optional channel is used.

4. Data and Commitment Model

This section separates one reference framed transport profile from the reusable commitment contract. The most deployment-specific surface is the transport profile in Section 4.1. The interoperable core is the deterministic accepted-telemetry-to-canonical-record projection, commitment encoding, Merkle calculation, day artifact, disclosure model, and verification behavior.

4.1. Reference Framed Transport Profile

This section defines one reference framed transport profile. Use of this profile is not required for every deployment that produces the canonical records and day artifacts defined by this document. A deployment that uses a different transport MUST define equivalent accepted-input, anti-replay, field-source, and projection rules before claiming interoperability with a commitment profile.

Under this reference profile, a frame is transported as newline-delimited JSON (NDJSON) with fields:

```
{
  "hdr": { "dev_id": 101, "msg_type": 1, "fc": 42, "flags": 0 },
  "nonce": "base64-24B",
  "ct": "base64-ciphertext",
  "tag": "base64-16B"
}
```

Figure 2

This reference profile uses XChaCha20-Poly1305 authenticated encryption with associated data (AEAD) in the sense of [RFC5116]. XChaCha20-Poly1305 is an extended-nonce variant of the ChaCha20-Poly1305 construction described by [RFC8439]; it is named here as a profile algorithm, and this document does not register a new IETF AEAD algorithm name. The wire shape shown here carries the encrypted payload in `ct`, carries the authentication tag separately in `tag`, and leaves `hdr` in cleartext. Gateways claiming this reference framed transport profile MUST validate header field presence, header ranges, nonce binding, and AEAD authentication before canonical-record emission.

- * `hdr.dev_id` MUST be an unsigned integer in the range 0..65535.
- * `hdr.msg_type` MUST be an unsigned integer in the range 0..255.
- * `hdr.fc` MUST be an unsigned integer in the range 0..(2³²-1).
- * `hdr.flags` MUST be an unsigned integer in the range 0..255.
- * `nonce` MUST be base64 text that decodes to exactly 24 bytes.
- * `tag` MUST be base64 text that decodes to exactly 16 bytes.

A deployment claiming conformance to this reference framed transport profile MUST use XChaCha20-Poly1305 and MUST identify the key-distribution method, key epoch, and device-to-gateway key binding used for frame validation. For the reference wire shape shown here, the associated data (AAD) supplied to the AEAD algorithm is exactly four octets: `uint16_be(hdr.dev_id) || uint8(hdr.msg_type) || uint8(hdr.flags)`. Receivers MUST derive those bytes from the received cleartext header before AEAD verification. A frame whose cleartext header values do not produce the same AAD bytes authenticated by the sender MUST fail validation.

The reference framed transport profile fixes the 24-octet nonce construction to `nonce = salt8 || uint64_be(hdr.fc) || tail8`. The `salt8` field is the provisioned per-device gateway salt. The `uint64_be(hdr.fc)` field MUST equal the cleartext frame counter encoded as an unsigned 64-bit big-endian integer. The `tail8` field is deployment-generated tail material. The full 24-octet nonce value MUST NOT repeat under the same AEAD key. If a sender can emit overlapping counter-ranges after reset, replay-state loss, device replacement, or tail regeneration, the deployment MUST rotate the affected AEAD key or perform an explicit resynchronization before further frames are accepted. The nonce and tag lengths fixed here are profile-defined transport parameters; they are not generic AEAD defaults.

- * Receivers MUST reject a frame whose decoded nonce length is not 24 octets.
- * Receivers MUST reject a frame whose nonce salt does not match the device salt8.
- * Receivers MUST reject a frame whose nonce counter does not match `hdr.fc`.
- * Receivers MUST reject a frame whose `hdr.flags` value is not 0.
- * Receivers MUST reject AEAD failures before canonical-record commitment and MUST NOT produce committed canonical records from them.

Frames that fail parse, range, nonce, or AEAD validation MUST be rejected before canonical-record commitment and MUST NOT produce committed canonical records. The `hdr.fc` field is bound through the nonce-counter check above and the anti-replay admission rules in Section 4.2. The `hdr.flags` field has no vl commitment or payload-interpretation semantics and MUST be 0 in this reference profile. Future semantic flags require a new transport profile and AAD rule.

Other deployments MAY use different transport framing or different cryptographic suites if they preserve the acceptance semantics needed to produce the same canonical-record objects under Section 4.3. Such deployments MUST NOT claim conformance to this reference framed transport profile unless they use the envelope, AAD, nonce, and rejection rules defined in this section.

The minimum device-side requirements needed to emit frames under this reference profile are summarized in Appendix C.

4.2. Anti-Replay Admission Criterion

For the reference framed transport profile, the replay unit is `(dev_id, fc)`. A gateway MUST commit a canonical record from a frame only if all of the following hold:

1. No canonical record has already been committed for the same replay unit `(dev_id, fc)`.
2. The frame counter `fc` is within the configured acceptance window for that device relative to durable replay state.
3. No continuity-break, reset, or replay-state-loss condition requires explicit resynchronization before further acceptance.

Frames that do not satisfy these conditions MUST NOT produce committed canonical records.

The RECOMMENDED default acceptance window is 64 frame counter values per device. Frames with fc more than window_size behind the highest accepted counter for a device MUST be rejected. In the reference gateway profile defined by this document, frames with fc more than window_size ahead of the highest accepted counter MUST also be rejected, rather than silently accepted across a large discontinuity.

The acceptance window in this section is counter-based, not a required wall-clock interval. Deployments can size buffering, retry, and resynchronization policy according to the configured connectivity window, but conformance here is defined by durable replay state and accepted counter progression rather than by device timekeeping.

Structured Rejection Evidence

Gateways SHOULD produce structured rejection evidence for frames rejected during parsing, header validation, AEAD authentication, anti-replay admission, or continuity-break handling. A rejection record SHOULD include at minimum:

- * device_id: the verifier-facing device label if the frame header could be parsed, else the empty string;
- * fc: the received hdr.fc if parseable, else null;
- * source: one of parse, decrypt, or replay;
- * reason: a short machine-usable reason string from the rejection-audit reason taxonomy;
- * observed_at_utc: the gateway-assigned observation time as an [RFC3339] UTC timestamp ending in Z; and
- * frame_sha256: lowercase hexadecimal SHA-256 over the exact received NDJSON octet string, excluding only trailing line terminators.

The rejection-audit reason taxonomy is closed for interoperable operator audit records. It includes parse and shape failures such as `invalid_json`, `missing_frame_fields`, and `unsupported_flags`; material and nonce failures such as `invalid_base64`, `nonce_length`, `nonce_salt_mismatch`, `nonce_fc_mismatch`, and `decrypt_failed`; and replay outcomes such as `duplicate` and `out_of_window`. Deployments that need additional operator-only rejection reasons MUST either use extension fields outside the interoperable taxonomy or define a new audit schema; they MUST NOT treat extension reasons as accepted telemetry semantics.

Rejection evidence is an audit artifact and MUST NOT be hashed into ledger commitments. It MUST NOT be represented as accepted telemetry and MUST NOT be used as a substitute for canonical-record artifacts. Rejection evidence is not part of the baseline commitment profile. If a future disclosure profile requires auditors to rely on rejection evidence as a published proof, that profile needs a distinct artifact family with its own schema and reason taxonomy.

Replay State Persistence

Gateways SHOULD persist replay state across restart. Loss of replay state can lead to duplicate acceptance of previously committed counter/nonce pairs under the same AEAD key epoch; see Section 10.4. If replay state is lost, gateways MUST record a continuity-break event, MUST NOT silently re-accept counters that could already have been committed, and MUST require explicit resynchronization or AEAD-key rotation before further frames from affected devices are accepted.

Scope Limitation

This profile provides tamper-evidence for committed canonical records. It does not prove the absence of selective pre-commitment drops by a malicious gateway.

4.3. Accepted-Telemetry-to-Canonical-Record Projection

The accepted-telemetry-to-canonical-record projection transforms an input that has passed the active transport validation and anti-replay contract into a canonical record suitable for commitment. The committed canonical record is a structured record derived from the accepted logical payload, not a copy of transport bytes.

When the reference framed transport profile is not used, the deployment profile MUST define the equivalent accepted input, anti-replay unit, payload interpretation, and field-source rules before applying the canonical-record and commitment rules below.

For the reference framed transport profile, the projection steps are:

1. The gateway MUST verify AEAD authentication.
2. The gateway MUST decrypt the ciphertext.
3. The gateway MUST parse the decrypted plaintext according to the frame's msg_type.
4. The gateway MUST construct a canonical-record object.
5. The gateway MUST serialize that canonical-record object under the gateway commitment profile.
6. The canonical-record bytes are then hashed for Merkle inclusion.

A committed canonical record under this profile MUST contain at minimum:

- * pod_id,
- * fc,
- * ingest_time,
- * pod_time,
- * kind, and
- * payload.

Interoperability depends on two distinct layers: the projection contract that maps accepted telemetry into a canonical-record object, and the commitment profile that serializes that canonical-record object and reduces the resulting digests. Implementations that claim the same verification result MUST agree on both layers.

A commitment profile or deployment profile claiming conformance MUST fix the type and source of each committed field so independent implementations can construct identical canonical-record bytes.

For each committed field, the applicable projection contract MUST state whether the field is transport-derived, payload-derived, gateway-assigned, or deployment metadata-derived. Independent implementations MUST NOT substitute a different field source while claiming the same projection contract.

This document defines one concrete commitment profile, verifiable-telemetry-canonical-cbor-v1. The projection requirements above are not limited to that profile; the field set and encoding below are. Any other commitment profile needs its own `commitment_profile_id` and conformance vectors.

For verifiable-telemetry-canonical-cbor-v1, the committed canonical-record object has six logical fields: `pod_id`, `fc`, `ingest_time`, `pod_time`, `kind`, and `payload`. Its authoritative commitment encoding is a fixed Concise Binary Object Representation (CBOR) array whose positional elements are:

1. `schema/version` discriminator, currently 1;
2. `pod_id` encoded as an 8-byte byte string;
3. `fc`;
4. `ingest_time`;
5. `pod_time` or null;
6. `kind` as the profile-defined family discriminator; and
7. `payload` encoded under the applicable payload family.

The profile-specific field definitions are:

- * `pod_id` MUST be a deterministic device identifier. When framed telemetry is used, the profile MUST define a deterministic mapping from `hdr.dev_id` or equivalent deployment alias to `pod_id`. In textual projections, this profile renders the identifier as lowercase hexadecimal text such as 0000000000000065. In authoritative CBOR commitment bytes, the same identifier is encoded as the corresponding 8-byte byte string.
- * `fc` MUST be the accepted monotonic counter as a non-negative integer. Under the reference framed transport profile, this is `hdr.fc`.
- * `ingest_time` MUST be a UTC integer timestamp fixed by the projection contract.
- * `pod_time` MUST be either a device-supplied integer timestamp or null when the device does not supply one.

- * kind MUST identify the record family under the applicable projection contract. JSON projections use lowercase public labels such as env.sample and custom.raw. In authoritative CBOR commitment bytes, this field is the numeric family discriminator defined by the commitment profile. For the current profile, the discriminator mapping is Env=1, Pipeline=2, Health=3, and Custom=250.
- * payload MUST be the parsed logical payload associated with the accepted telemetry input. Under the reference framed transport profile, the plaintext is decoded according to the applicable payload profile before this field is constructed.

The field names pod_id and pod_time are stable commitment-field names for verifiable-telemetry-canonical-cbor-v1. They do not require a deployment to use pod-based lifecycle semantics.

A different deployment profile MAY define additional logical fields, but such an extension is not compatible with verifiable-telemetry-canonical-cbor-v1 unless it is given a distinct commitment_profile_id and corresponding conformance vectors.

An implementation claiming parity with verifiable-telemetry-canonical-cbor-v1 MUST reproduce that exact logical record and its fixed array encoding before applying the deterministic CBOR rules in Section 4.4.

Ciphertext, raw transport bytes, and the authentication tag MUST NOT be part of the committed canonical-record object. The exact payload schema is deployment-specific; the deterministic projection contract is the normative requirement and MUST be published for any commitment profile that claims interoperability.

Published conformance vectors for a commitment profile MUST include the post-projection canonical-record objects used as commitment inputs, not only transport inputs.

4.4. Deterministic Concise Binary Object Representation (CBOR) Commitment Encoding

This section does not define a new general-purpose CBOR variant. It records the narrow deterministic CBOR encoding used for commitment bytes by this profile. The identifier verifiable-telemetry-canonical-cbor-v1 names this commitment recipe so verifiers can tell which byte-level rules were used; see Section 13.3 for allocation policy.

The authoritative commitment artifacts, namely CBOR canonical-record artifacts and the canonical day artifact, use a constrained subset of deterministic encoding under Section 4.2.1 of [RFC8949]. For commitment bytes in this profile, the following concrete choices apply:

- * All commitment-path items MUST use definite-length encoding.
- * Integers MUST use the shortest encoding width permitted by [RFC8949].
- * Map keys MUST be CBOR text strings.
- * Map keys MUST be sorted by encoded key length ascending, then by lexicographic order of the encoded key bytes.
- * Finite floating-point values MUST be encoded using the shortest of float16, float32, or float64 that exactly preserves the value.
- * NaN, positive infinity, and negative infinity MUST be rejected in commitment paths.
- * CBOR tags MUST NOT appear in commitment bytes.
- * Supported values are unsigned integers, negative integers, byte strings, text strings, arrays, maps, booleans, null, and deterministic finite floats.

Implementations MUST NOT accept generic CBOR serializers as authoritative commitment encoders. An encoder is acceptable only if it yields the same bytes as these rules.

JSON projections of canonical-record artifacts and day artifacts are optional and non-authoritative. They MUST NOT be used as commitment inputs. When produced, such projections SHOULD follow [RFC8785].

Device-side or embedded components MAY use other internal encodings, including different deterministic CBOR layouts optimized for local constraints. Those encodings are not the authoritative commitment encoding described here unless they are explicitly identified by a distinct `commitment_profile_id` and verified under their own rules.

4.5. Deterministic Commitment Tree Calculation

For a given day D, the current commitment profile computes a daily root from the canonical-record commitment bytes produced under Section 4.4. The following steps describe that calculation.

Leaf Digests

- * Each canonical-record byte string is hashed with SHA-256, yielding a 32-byte leaf digest.

Digest Ordering

- * To make the daily root independent of file order or ingest order, leaf digests MUST be sorted in ascending byte order before reduction.
- * Lowercase hexadecimal is a representation format for artifacts and examples only; internal Merkle computation operates on raw hash bytes.
- * Sorting by lowercase hexadecimal is equivalent to bitwise ascending order over the raw digests.

Pairwise Reduction

- * The sorted digests are reduced pairwise by computing `SHA-256(left_child_bytes || right_child_bytes)`, where both operands are raw 32-byte digests.
- * If a layer has an odd number of digests, the final digest is duplicated to form the last pair.
- * The current commitment profile does not prepend domain-separation bytes to leaf or parent hashes.

This profile relies on fixed-size 32-octet child operands for parent hashes and on leaf digests computed over canonical-record bytes. The resulting tree is a deterministic commitment profile for this document; it is not a general-purpose Merkle proof format. Implementations MUST NOT mix this hash construction with a domain-separated construction under the same `commitment_profile_id`.

Empty Day

- * If no canonical records are committed for the day, the daily root is the SHA-256 digest of zero bytes:
`e3b0c44298fclcl149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`.

The resulting daily root is deterministic for the set of committed canonical records. Because the leaf digests are sorted before reduction, the result depends on the committed record set rather than on ingestion order.

Any future change to this calculation that alters commitment bytes (for example, adding domain separation) MUST use a new `commitment_profile_id`.

4.6. Day Artifact Schema

The authoritative day artifact is a CBOR-encoded day record produced under Section 4.4. The day record contains the following fields:

The day artifact is the stable day-scoped object of commitment verification in this profile. It commits to accepted canonical records for one UTC day through embedded batch metadata and `day_root`. It is not a transport envelope, not a publication statement, and not by itself a complete verification claim.

A day artifact is interpreted only under an explicit `commitment_profile_id`. For `verifiable-telemetry-canonical-cbor-v1`, that identifier is verifier-visible claim context supplied by the verifier-facing day manifest or an explicitly profiled equivalent; it is not an in-band field of the day record. Verifiers MUST NOT infer the commitment profile from the file name, media type, day-record version, or proof sidecar alone.

- * `version (uint)`: day-record schema version, currently 1.
- * `site_id (tstr)`: site identifier.
- * `date (tstr)`: UTC day label in YYYY-MM-DD form.
- * `prev_day_root (tstr)`: previous day root as 64 lowercase hexadecimal characters.
- * `batches (array)`: array of batch objects.
- * `day_root (tstr)`: deterministic day root as 64 lowercase hexadecimal characters.

Each batch object contains:

- * `version (uint)`: batch-record schema version, currently 1.
- * `site_id (tstr)`: site identifier.
- * `day (tstr)`: UTC day label in YYYY-MM-DD form.
- * `batch_id (tstr)`: batch identifier.

- * merkle_root (tstr): batch Merkle root as 64 lowercase hexadecimal characters.
- * count (uint): number of committed canonical records in the batch.
- * leaf_hashes (array of tstr): sorted leaf hashes as lowercase hexadecimal strings.

The batch objects embedded in the day artifact are the authoritative batch metadata for verification.

- * For each batch object, count MUST equal the length of leaf_hashes.
- * For each batch object, merkle_root MUST equal the Merkle reduction of that batch's leaf_hashes under Section 4.5.
- * The multiset union of all batch leaf_hashes for the day MUST equal the day's leaf digest multiset from which day_root is computed.

day/YYYY-MM-DD.cbor is authoritative. The corresponding day/YYYY-MM-DD.json file is a projection only.

The normative field tables in this section are consistent with the published commitment-family Concise Data Definition Language (CDDL) and vector corpus for this profile. An implementation claiming parity with verifiable-telemetry-canonical-cbor-v1 MUST satisfy this text and reproduce the published machine-readable artifacts for that profile.

4.7. Day Chaining

Day records include prev_day_root.

- * The epoch day (the first committed day) for a site MUST set prev_day_root to 64 zero characters, representing 32 zero bytes.
- * Non-epoch days MUST set prev_day_root to the previous committed day's day_root.

Because day labels are UTC-based, chaining semantics are also defined on UTC day boundaries.

5. Artifacts and Verification Bundles

Illustrative artifact layout:

- * records/<record-id>.cbor - authoritative canonical records

- * records/<record-id>.json - optional projections
- * day/YYYY-MM-DD.cbor - authoritative canonical day artifact
- * day/YYYY-MM-DD.json - optional projection
- * day/YYYY-MM-DD.verify.json - verifier-facing day manifest
- * blocks/YYYY-MM-DD-00.block.json - optional standalone block-metadata projection of one batch object
- * day/YYYY-MM-DD.cbor.sha256 - convenience digest
- * day/YYYY-MM-DD.cbor.ots - OTS proof
- * day/YYYY-MM-DD.ots.meta.json - OTS binding metadata

Deployments MAY store artifacts differently and MAY export them as bundles. The path shapes above are illustrative.

Standalone block metadata files are convenience projections of batch objects already carried in the authoritative day artifact. They are not additional commitment inputs. Verifiers that process disclosed standalone block-metadata projections MUST compare them with the corresponding batch object in the authoritative day artifact and reject mismatches.

Every verification bundle MUST disclose the `commitment_profile_id`. In this profile, the normal machine-readable disclosure surface is the verification manifest `day/YYYY-MM-DD.verify.json`. That manifest carries artifact digests, disclosure class, channel state, and the executed or skipped verification checks together with the day-scoped verification result. It is evidence metadata, not merely a local report.

Verifier-facing manifests MUST be portable. Artifact paths in the manifest are interpreted relative to the disclosed evidence bundle root. Absolute paths and parent-directory traversal are not verifier claims and MUST be rejected by strict verifiers. Every artifact listed in the manifest MUST carry a lowercase hexadecimal SHA-256 digest entry.

The verifier-facing metadata binds claim semantics to the stable day artifact. A bundle that discloses `day/YYYY-MM-DD.cbor` and a timestamp proof but does not disclose the applicable `commitment_profile_id` can at most support artifact-digest and timestamp checks; it MUST NOT be represented as semantic verification of the day artifact.

The verification manifest is the primary disclosure surface for verifier-visible metadata. A representative manifest shape is defined in Appendix D. Deployment-specific schemas can carry additional operational fields, but they **MUST** preserve the semantics defined here.

At minimum, an OTS sidecar **MUST** bind:

- * artifact,
- * artifact_sha256, and
- * ots_proof.

Verifiers **MUST** recompute the day artifact digest and compare it with the sidecar before accepting any proof validation result.

6. Anchoring and Verification

6.1. Anchoring Contract

The generic anchoring contract is simple: a gateway computes the SHA-256 digest of the authoritative day artifact and submits that digest to one or more external timestamping channels. Verifiers **MUST** first recompute the day artifact digest locally; proof validation occurs only after digest binding validation succeeds.

A timestamp or attestation over the day-artifact digest proves only a binding to the artifact bytes. It does not identify the `commitment_profile_id`, disclosure class, or verifier checks exercised for a verification claim unless those semantics are also disclosed by verifier-facing metadata or an explicitly profiled publication statement.

A deployment conforming to this profile **MUST** use at least one anchoring channel. OpenTimestamps (OTS; [OTS]) is the default channel described by this document; timestamp responses under [RFC3161] and peer signatures are optional parallel channels.

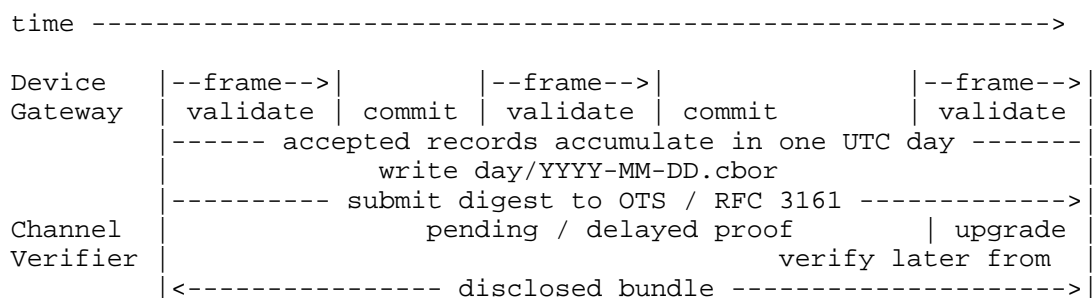


Figure 3: Illustrative Commitment and Proof Lifecycle

Figure 3 illustrates the current lifecycle. The device-to-gateway transport can be intermittent, while proof completion or later publication can lag behind the authoritative day-artifact write.

6.2. OTS Anchoring Profile

When [OTS] is used, the gateway stamps `SHA-256(day/YYYY-MM-DD.cbor)` and stores an OTS proof plus an OTS sidecar.

OpenTimestamps is referenced here as a deployed public timestamping ecosystem rather than an IETF-standardized proof format. Implementations claiming OTS support depend on the interoperable behavior of the public OTS project, its calendar servers, and compatible client tooling. OTS proof-format interoperability is therefore defined operationally by that ecosystem and its reference implementations.

6.2.1. OTS Anchoring Lifecycle

1. `_Submission_`: the gateway submits the day artifact digest to one or more OTS calendars.
2. `_Pending_`: a calendar can return an incomplete proof while awaiting Bitcoin commitment.
3. `_Upgrade_`: the gateway or a background process can later upgrade the proof to include completed attestations.
4. `_Verification_`: a verifier recomputes the artifact digest and validates the proof.

Gateways SHOULD submit to multiple calendars operated by administratively distinct entities to reduce single-calendar unavailability risk.

6.2.2. Handling Delayed or Failed Anchoring

If OTS submission fails, times out, or yields only an incomplete proof, the gateway **MUST** still write the authoritative day artifact and **MUST** treat OTS as a separate channel whose state is not yet complete. The gateway **MAY** retain a placeholder or incomplete proof artifact and **MAY** later replace or upgrade it as additional OTS evidence becomes available. Until a valid proof is disclosed and verified, verifiers **MUST** report the OTS channel as pending, missing, or failed according to the disclosed artifacts and local policy, rather than treating the day artifact itself as invalid. Any later replacement or upgrade of the OTS proof **MUST** continue to bind to the same authoritative day-artifact digest.

Gateways **MUST** retain disclosed OTS proof artifacts for at least as long as the corresponding day artifacts remain available for verification under local retention policy.

6.2.3. Proof Status Vocabulary

Verifiers and bundle manifests **SHOULD** use a consistent status vocabulary for OTS and optional parallel attestation channels:

- * **verified**: proof validation succeeded for the disclosed artifact binding.
- * **pending**: a proof exists but is incomplete or awaiting upgrade; this is not equivalent to invalid.
- * **missing**: the expected proof or channel artifact is absent.
- * **failed**: validation was attempted and did not succeed.
- * **skipped**: validation was not attempted because of disclosure class, verifier configuration, or local policy.

Whether missing, pending, or skipped is verifier-fatal depends on the disclosure class, local verifier policy, and whether the relevant channel is required.

6.3. Optional Parallel Attestation

Deployments **MAY** also produce:

- * An [RFC3161] timestamp response over the same day-artifact digest.
- * A deployment-specific peer signature quorum over the same day-artifact digest or day root.

This document does not define an interoperable peer-signature validation profile. A deployment that reports `peer_quorum_verification` as an interoperable check MUST also disclose, or reference, a profile that defines the signed payload encoding, signature algorithm, public key or certificate identity model, quorum threshold, and verifier processing rules.

When multiple channels are present, verifiers SHOULD validate all available channels independently and report per-channel results.

If a verifier is configured in strict mode for optional channels, failure of those channels MUST cause overall verification failure.

6.4. Verification

Verifiers MUST first determine the applicable `commitment_profile_id` from disclosed verifier metadata. When the authoritative day artifact does not carry that identifier in-band, the verifier MUST obtain it from `day/YYYY-MM-DD.verify.json` or an explicitly profiled manifest with equivalent verifier-visible fields and MUST reject absent or unsupported values.

The verifier MUST treat `commitment_profile_id` as the semantic key for interpreting the authoritative day artifact. A structurally well-formed day artifact and a valid timestamp proof are insufficient for semantic verification if the applicable `commitment_profile_id` is absent, unsupported, or not bound to the same disclosed day-artifact digest.

Verifiers MUST determine the applicable verification scope from the disclosed artifacts, the claimed disclosure class, and local verifier policy. Reported outcomes MUST NOT claim checks or assurances outside that scope.

Verifiers MUST NOT trust the manifest's own `checks_executed` or `checks_skipped` values as proof that those checks were performed. A verifier produces its own check lists after validating the disclosed artifacts.

Verifiers SHOULD apply checks in the following fail-fast order, subject to the claimed disclosure class:

1. Validate that disclosed artifacts are sufficient for the claimed disclosure class and disclose a `commitment_profile_id`, and report `bundle_disclosure_validation`.

2. Validate the authoritative day artifact and, when disclosed or consumed, the verifier-facing manifest, and report `day_artifact_validation` and `verification_manifest_validation` as applicable.
3. For Class A bundles, recompute canonical-record leaf digests from disclosed canonical-record CBOR artifacts, validate the batch metadata contract, and recompute `day_root`. Compare the recomputed result to the authoritative `day_root`, and report `record_level_recompute` and `batch_metadata_validation`.
4. For Class B bundles, validate the authoritative day artifact and any disclosed batch metadata. If withheld material prevents public recomputation, report `record_level_recompute` in `checks_skipped`. Deployment-specific withheld-material checks MAY be reported only through extension names beginning with `x-`.
5. For Class C bundles, report `record_level_recompute` and `batch_metadata_validation` in `checks_skipped` as out of scope, and treat the result as anchor-only evidence.
6. Recompute `SHA-256(day/YYYY-MM-DD.cbor)` and compare it to the sidecar artifact `sha256`, and report `day_digest_binding`.
7. Validate the OTS proof when OTS is required or present, and report `ots_verification`.
8. Validate optional [RFC3161] timestamp responses and peer attestations as configured, and report `tss_verification` or `peer_quorum_verification` as applicable.

When batch metadata is within the exercised verification scope, verifiers MUST apply the following checks before accepting a result:

- * each batch count equals the length of its `leaf_hashes`;
- * each batch merkle_root equals the Merkle reduction of its `leaf_hashes`;
- * the union multiset of batch `leaf_hashes` equals the leaf digest multiset derived from disclosed canonical records when canonical-record artifacts are available; and
- * any standalone block-metadata projection matches the corresponding batch object in the authoritative day artifact.

Verifier implementations SHOULD expose machine-usable failure categories:

- * malformed or missing artifacts,
- * missing or unsupported commitment_profile_id,
- * Merkle mismatch,
- * batch metadata mismatch,
- * missing or invalid OTS proof,
- * sidecar mismatch or digest mismatch,
- * insufficient disclosure for the claimed verification level, and
- * optional-channel failure.

Verifier output SHOULD state the claimed disclosure class, the verification scope actually exercised, the per-channel proof status, which checks were executed, which checks were skipped, and whether the resulting claim is public recompute, partial verification, or anchor-only evidence.

Standardized Check Identifiers

When verifier-facing manifests or verifier results report checks_executed or checks_skipped, this profile defines the following interoperable check identifiers:

- * bundle_disclosure_validation: disclosure-class sufficiency and commitment_profile_id presence were validated.
- * verification_manifest_validation: the verifier-facing manifest was validated.
- * day_artifact_validation: the authoritative day artifact was validated structurally and semantically under the claimed profile.
- * record_level_recompute: record-level recomputation from disclosed canonical-record artifacts was performed.
- * batch_metadata_validation: authoritative or disclosed batch metadata was validated against the day artifact and disclosed record material within scope.
- * day_digest_binding: the recomputed SHA-256(day/YYYY-MM-DD.cbor) digest was compared with disclosed binding metadata.
- * ots_verification: the OTS proof channel was validated.

- * `tsa_verification`: the [RFC3161] timestamp channel was validated.
- * `peer_quorum_verification`: the peer-signature quorum channel was validated.

A manifest or verifier result that reports one of these standardized checks MUST use the exact identifier listed here. For any standardized check whose applicability conditions are met within the exercised verification scope, the manifest or verifier result MUST report that check exactly once in either `checks_executed` or `checks_skipped`. The same standardized identifier MUST NOT appear in both lists and MUST NOT appear more than once. Silent omission of an applicable standardized check is non-conformant.

Applicability is determined as follows:

- * `bundle_disclosure_validation` and `day_artifact_validation` apply whenever verification proceeds on a disclosed bundle.
- * `verification_manifest_validation` applies when a verifier-facing manifest is disclosed or consumed to obtain verifier-visible metadata.
- * For Class A bundles, `record_level_recompute`, `batch_metadata_validation`, and `day_digest_binding` apply. If required artifacts for one of these checks are absent, that check MUST appear in `checks_skipped` with a reason indicating insufficient disclosure or missing artifacts.
- * For Class B bundles, `batch_metadata_validation` and `day_digest_binding` apply when the corresponding artifacts are disclosed. If withheld material prevents public recomputation, `record_level_recompute` MUST appear in `checks_skipped`.
- * For Class C bundles, `record_level_recompute` and `batch_metadata_validation` MUST appear in `checks_skipped` as out of scope, while `day_digest_binding` applies when the day artifact and binding metadata are disclosed.
- * `ots_verification`, `tsa_verification`, and `peer_quorum_verification` apply only when the corresponding channel is disclosed or required by verifier policy.

Additional deployment-specific checks MAY be reported, but they MUST NOT redefine these identifiers and MUST use a distinct extension name beginning with `x-`.

Verifier output MUST NOT be represented as proving more than the exercised verification scope. In particular, a successful result does not by itself establish dataset completeness, physical truth of measurements, or suitability for autonomous actuation or sanctions.

7. Disclosure Classes

Verification claims depend on what artifacts are disclosed. This profile defines three disclosure classes.

- * *Class A (Public Recompute)*: sufficient material for independent record-level recomputation.
- * *Class B (Partner Audit)*: controlled disclosure with redacted or partitioned record material.
- * *Class C (Anchor-Only)*: existence and timestamp evidence only.

A verifier claim for any disclosure class MUST be limited to the verification scope supported by the disclosed artifacts.

Disclosure classes are verifier behavior, not prose intent. A verifier MUST decide each class from the artifacts it can recompute, the artifacts it cannot recompute, and the checks it reports as executed or skipped.

Class A is the public-recompute case. A Class A bundle MUST include:

- * all canonical-record artifacts required to recompute the claimed day root,
- * the canonical day artifact, including its authoritative batch objects,
- * any disclosed standalone block-metadata projections, and
- * the OTS proof plus its sidecar metadata.

A Class A bundle SHOULD also include the verifier-facing day manifest and MUST record the `commitment_profile_id`.

Class A permits record-level recomputation, batch-metadata validation, day-root recomputation, and anchor validation when the corresponding artifacts are disclosed and checked.

A Class A verifier can recompute canonical-record digests from disclosed record artifacts, the batch Merkle root as defined in Section 4.5, batch/day consistency, the day artifact digest, manifest

artifact digests, and enabled anchoring or publication proofs when those artifacts are present. It cannot prove physical truth of measurements, omitted device lifecycle state, or omitted external channels. It MUST report recomputation and manifest/profile status. A claim of public recomputability is false if required record artifacts are missing, root recomputation was skipped, manifest/profile binding is absent under a strict verifier policy, or digest checks fail.

Class B is the controlled-disclosure case. Class B outputs MUST NOT be represented as publicly recomputable. A Class B bundle SHOULD include the canonical day artifact, any disclosed standalone block-metadata projections, at least one timestamp proof plus its binding metadata, any disclosed commitments covering withheld material, and a policy artifact describing the withheld or partitioned material.

This document defines the reporting boundary for Class B but does not require one universal withheld-material artifact format. Class B validation is therefore limited to the authoritative day artifact, disclosed batch metadata, any disclosed withheld-material commitments, and any anchor channels present in the bundle. Verifier output SHOULD explicitly state when record-level recomputation was partial or was not attempted for withheld material. If deployment-specific checks are reported for withheld-material commitments or policy artifacts, they MUST use extension names beginning with x-.

A Class B verifier can validate day/batch consistency, day artifact digest binding, manifest artifact digests, enabled anchors, and any disclosed partition commitments or auditor-supplied material. It cannot publicly recompute the complete canonical-record set when material is withheld. It MUST report `record_level_recompute` as skipped when public record-level recomputation is impossible and MUST report that the result is not publicly recomputable. A claim of complete public record-level recomputation or complete public auditability is false for Class B.

A Class C disclosure MUST be labeled as existence and timestamp evidence only and MUST NOT claim record-level reproducibility.

A Class C bundle MUST include:

- * the canonical day artifact, and
- * at least one timestamp proof artifact plus the metadata needed to bind it to the day artifact digest and disclose `commitment_profile_id`.

Class C verification validates artifact-digest binding and external timestamp evidence only. It does not establish record-level reproducibility.

Class C verifier output SHOULD be reported as anchor-only evidence.

A Class C verifier can validate day artifact digest binding, manifest digests, chain or anchor existence, and timestamp or publication status when the relevant artifacts are present. It cannot recompute canonical- record contents, record-set completeness, Merkle leaves, or semantic telemetry values. It MUST report record-level and batch recomputation as skipped. A claim of record-level authenticity, record completeness, or recomputed telemetry contents is false for Class C.

Bundle Selection Guidance

- * Class A is appropriate when the goal is public recomputation from disclosed canonical records and authoritative artifacts, for example when a regulator or external researcher needs independent replay of the disclosed day.
- * Class B is appropriate when the goal is partner or regulator review over a controlled disclosure set that still carries commitment and anchor evidence, for example when some record content must remain withheld while the committed day artifact is still audited.
- * Class C is appropriate when the goal is existence and timestamp evidence without public record-level reproducibility, for example when a party only needs to show that a committed artifact existed by a given time.

The verifier-facing manifest MUST include:

- * disclosure_class,
- * commitment_profile_id,
- * artifact path and digest entries,
- * per-channel anchor status,
- * a list of checks executed using the standardized identifiers defined in Section 6.4, and
- * a list of checks skipped, each with the standardized check identifier and the reason for the skip.

8. Versioning

This profile has several independent version surfaces:

- * Document revision (for example -00, -01) is editorial and is not part of commitment output.
- * Artifact schema versions are carried by the version fields in day and batch records.
- * `commitment_profile_id` identifies the canonical CBOR, hash, and Merkle rules that define commitment outputs.

The commitment profile defined in this document is verifiable-telemetry-canonical-cbor-v1. If a verifier encounters an unsupported `commitment_profile_id`, it MUST reject the verification claim rather than silently using a fallback interpretation.

This revision defines exactly one non-private `commitment_profile_id`. Additional non-private profile identifiers are allocated as described in Section 13.3 and require a stable public specification before they are used for interoperability claims.

Bundles disclose the applicable `commitment_profile_id` via the verifier-facing day manifest. The required OTS sidecar metadata does not currently carry that identifier.

A day-record version identifies the schema of the day artifact under a known commitment profile. It is not a substitute for `commitment_profile_id`. Likewise, a media type or file extension can identify the artifact family, but it does not authorize a verifier to select canonical CBOR, hash, or Merkle rules without an explicit commitment profile.

The following migration rules apply to the public contract surfaces:

New `commitment_profile_id`

Required for changes to authoritative fact, batch, or day commitment bytes; deterministic CBOR encoding rules; accepted integer, float, null, timestamp, or unknown-field behavior for commitment inputs; hash algorithm; Merkle leaf ordering; parent construction; odd-leaf policy; day-root chaining semantics; or disclosure-class recomputation semantics.

New schema version or schema identifier

Required for changes to a public schema's required fields, closed-field policy, field type, enum vocabulary, or field meaning while preserving the same artifact family and commitment profile.

New artifact family

Required for a new primary evidence object, trust domain, publication statement, disclosure package, rejection-audit public contract, sealed-state object, or non-day commitment unit.

Verifier tolerance only

Appropriate for accepting legacy aliases, optional historical fields, alternate discovery locations, or stricter and looser validation lanes when produced commitments and verifier truth claims do not change. Material tolerance behavior MUST be reported by verifier output.

Projection change only

Appropriate for SensorThings output, JSON presentation, CLI rendering, report formatting, non-authoritative labels, or read-only consumer exports that are not hashed into commitments and do not change verifier claims.

A change can trigger more than one action. For example, a new publication statement that also changes commitment bytes requires both a new artifact family and a new `commitment_profile_id`.

9. Conformance Vectors

Determinism claims in this profile are testable. Implementations that claim conformance to a published commitment profile MUST be able to reproduce its published machine-readable corpus. For verifiable-telemetry-canonical-cbor-v1, the authoritative commitment-family CDDL and vector corpus define the machine-readable conformance artifacts for this profile. The appendix in this document is explanatory only.

The current repository publishes the verifiable-telemetry-canonical-cbor-v1 corpus under `toolset/vectors/verifiable-telemetry-canonical-cbor-v1/`. Future publication can move or mirror the corpus to a versioned public Git repository, but published bundles MUST continue to identify the applicable `commitment_profile_id`.

Published vector sets should include coverage for:

- * post-projection canonical-record fixtures with fixed field types and values,
- * empty day,
- * single canonical record,
- * odd leaf count,

- * power-of-two leaf count,
- * duplicate leaf hashes,
- * epoch chaining,
- * non-epoch chaining, and
- * a full Class A disclosure example.

Cross-implementation checks MUST verify byte-for-byte parity across independent implementations. Any mismatch in canonical bytes or roots is a conformance failure.

Published vector bundles MUST include the `commitment_profile_id`.

10. Operational Considerations

This section consolidates deployment guidance for operators and follows the operational topics described in [I-D.ietf-opsawg-rfc5706bis]. It does not make operator policy part of the cryptographic commitment output, but it identifies the state and configuration that determine whether the committed evidence remains useful and verifiable.

10.1. Gateway Time and Rollover

The gateway assigns UTC-based `ingest_time` values, day labels, and day-artifact rollover boundaries. The correctness of those assignments depends on its authenticated time source. Deployments SHOULD use authenticated time synchronization, SHOULD monitor gateway clock health, and SHOULD alarm on loss of synchronization, clock regression, unexpected clock step, or rollover outside the configured UTC day boundary. Operator documentation SHOULD state whether ingestion pauses, continues with a continuity-break event, or enters a deployment-specific degraded mode when clock health is unknown.

10.2. Health and Fault Management

Operators SHOULD monitor at least the following signals:

- * replay-state persistence success and continuity-break events,
- * AEAD authentication failure rate and malformed-frame rejection rate,
- * day-artifact write success, digest production, and day-root calculation,

- * anchoring backlog, calendar reachability, and proof-upgrade lag for OTS,
- * status of optional [RFC3161] and peer-signature channels when configured, and
- * local storage pressure for retained day artifacts, proof artifacts, manifests, and rejection-audit records.

An operator-visible fault state SHOULD distinguish an invalid commitment artifact from a delayed or failed external anchoring channel. A delayed OTS proof, for example, is not by itself evidence that the authoritative day artifact is malformed.

10.3. Configuration Management

Deployments SHOULD document the configured acceptance window, connectivity window, replay-state durability mechanism, artifact and proof retention period, OTS calendars, optional TSA identities, anchoring policy, verifier strictness for optional channels, and any peer-signature quorum threshold and identity set. The RECOMMENDED default acceptance window for the reference framed transport profile is 64 frame counter values per device. That default is intended to cover short outages such as approximately one message per minute for about one hour; deployments with different cadence or outage models SHOULD document the selected value and replay-risk tradeoff. Replay-state persistence SHOULD be enabled by default. When OTS is used, gateways SHOULD submit to multiple calendars operated by administratively distinct entities and retain disclosed OTS proof artifacts for at least as long as the corresponding day artifacts remain available for verification.

10.4. Fault-to-Verifier-Result Mapping

Operational faults do not all have the same verifier meaning. Verifier output MUST distinguish at least the following cases when the condition is visible from disclosed artifacts or local verifier policy:

OTS proof exists but is incomplete or awaiting upgrade
Report the OTS channel as pending. Place `ots_verification` in `checks_skipped` with reason `pending_proof` unless validation was attempted and failed. Overall success is permitted only when local policy does not require completed OTS evidence.

[RFC3161] timestamp response is absent

Report the TSA channel as missing when required by policy, or skipped with reason disabled when the channel is not configured.

Report `tsa_verification` exactly once in either `executed` or `skipped` checks when applicable.

Replay-state loss or continuity break is disclosed for affected devices

Report the continuity-break condition. Do not treat affected later frames as normal accepted telemetry unless explicit resynchronization or AEAD-key rotation is also disclosed.

10.5. Performance and Scaling

Artifact size grows primarily with the number of accepted canonical records and the number of batches in a UTC day. Class A verification requires access to the disclosed canonical-record artifacts and scales with the number of records that must be decoded, hashed, and reduced into the day root. Class C verification can be substantially smaller because it validates the day-artifact digest and disclosed proof channels without public record-level recomputation. OTS submission traffic is one digest submission per emitted day artifact per selected calendar; using multiple calendars increases availability but also increases submission attempts and proof-status tracking.

10.6. Verifying Correct Operation

Operators SHOULD periodically run an independent verifier against a recently produced bundle using the same disclosure class that consumers are expected to rely on. A healthy run should confirm that disclosed artifacts are present, the `commitment_profile_id` is supported, the authoritative day artifact is well formed, the day-artifact digest matches binding metadata, expected checks appear exactly once in either `checks_executed` or `checks_skipped`, and configured proof channels report a status permitted by local policy and reported according to Section 10.4. Operators SHOULD alert when required checks are silently absent, when OTS proofs remain pending beyond local policy, or when replay-state, clock-health, or day-artifact-write faults occur.

11. Security Considerations

This profile does not introduce new cryptographic primitives. Its security depends on correct AEAD use on the transport path, deterministic commitment encoding, trustworthy gateway admission and timekeeping, accurate verifier reporting, and disciplined artifact and proof handling. The threats below are stated in the threat-and-remediation style described by [RFC3552]. Unless explicitly stated otherwise, a successful verification result establishes only that the disclosed artifacts are internally consistent with this profile and with any validated proof channels.

Gateway Compromise and Pre-Commit Omission

An attacker can compromise the gateway or the local admission path and then fabricate accepted telemetry, suppress received telemetry before commitment, or assign accepted telemetry to the wrong device or UTC day. This profile does not by itself detect a malicious gateway; it makes committed outputs tamper-evident only after commitment, as described in Section 1.2. Deployments that need stronger assurances should harden and audit the gateway, protect admission and replay state, protect gateway time sources, and add independent observation, device attestation inputs, or admission-path audit evidence.

Replay and Duplicate Submission

An attacker can replay a previously observed frame or resubmit a frame with reused (dev_id, fc) values in an attempt to create duplicate committed records or counter ambiguity. Gateways mitigate this attack by enforcing single-consumption of the replay unit (dev_id, fc), by rejecting counters outside the configured acceptance window, and by recording continuity-break conditions instead of silently re-accepting ambiguous counters after replay-state loss. Operational handling is described in Section 10.3 and Section 10.4.

Nonce Misuse and Transport-Key Compromise

An attacker can recover plaintext or forge framed transport messages if AEAD keys are disclosed, if nonce uniqueness fails under a key, or if weak generation causes nonce-tail repetition. Implementations mitigate this attack by rejecting frames that fail AEAD authentication before commitment, by enforcing the nonce construction and checks in Section 4.1, by assigning a clear key epoch to each device-to-gateway key, and by using durable uniqueness discipline for the deployment-generated tail material and frame counter. Reuse of nonce under the same AEAD key can invalidate confidentiality and can also invalidate integrity, depending on the AEAD algorithm. If a

durable counter or tail state is lost, or if a device can resume with an overlapping counter-range, the deployment MUST treat further frames under that key as unsafe until resynchronization or rekeying has occurred; see Section 10.4. This profile therefore depends on deployment key management to prevent unsafe key and nonce reuse and to retire compromised keys.

Canonicalization, Profile, and Metadata Confusion

An attacker can exploit differences in parsing, field typing, record ordering, hash composition, or non-authoritative metadata handling while implementations still claim the same `commitment_profile_id`. An attacker can also present a verifier-facing manifest, block-metadata projection, or OTS sidecar that does not match the authoritative day artifact in the hope that a verifier will treat convenience metadata as authoritative. The mitigation is that this profile fixes deterministic encoding and hash rules, treats the authoritative day artifact as the cryptographic source of truth, requires verifiers to recompute commitment material from authoritative artifacts, requires the applicable `commitment_profile_id` to be disclosed and bound to the same day-artifact digest, and requires standalone metadata projections and sidecars to match the authoritative artifact. Any future semantic or hash-composition change MUST use a new `commitment_profile_id`.

Artifact Mutation and Proof Substitution

An attacker can modify a committed day artifact after disclosure, or can present a valid proof over the wrong artifact digest. The mitigation is that verifiers recompute the authoritative day-artifact digest independently and compare it with the disclosed binding metadata before accepting any proof result. Mutation or substitution therefore changes the digest or its binding and causes verification to fail.

Calendar Withholding and Optional-Channel Downgrade

An attacker can operate or compromise a timestamping or attestation service so that proof issuance is delayed, withheld, or selectively unavailable, can present pending or placeholder proofs as if they were final attestations, or can exploit verifier policy that silently ignores a missing required channel. Implementations mitigate this attack by treating timestamping and other attestations as separate channels, by disclosing per-channel status explicitly, by distinguishing pending, missing, and failed from verified, and by binding every exercised channel to the same authoritative day-artifact digest. Using calendars operated by administratively distinct entities reduces but does not eliminate coordinated

withholding risk. Verifier policy must make clear which channels are required and must not treat an unverified required channel as success; see Section 10.4.

Time-Source Manipulation

An attacker can alter the gateway clock or day-rollover configuration so that accepted records are assigned to the wrong UTC day or are given misleading ingest_time values. The device is not the time authority in this profile; the gateway is. Implementations therefore mitigate this attack by maintaining a trustworthy gateway time source, monitoring rollover behavior, and treating UTC day assignment as a security-relevant operational function; see Section 10.1. Verification can detect internal inconsistencies in disclosed artifacts, but it cannot reconstruct true real-world time if the gateway time base was wrong at acceptance.

Resource Exhaustion

An attacker can flood a deployment with malformed, replayed, or excessive-rate frames in order to exhaust buffering, replay-state storage, artifact storage, or verifier computation. Implementations mitigate this attack by rejecting malformed or AEAD-invalid frames before commitment, bounding acceptance windows and retained replay state, sizing local storage and retention policy explicitly, and applying rate limits or other admission controls to untrusted senders. Relevant operational signals are described in Section 10.2 and Section 10.5. This profile does not define a complete denial-of-service defense.

Verification Scope, Completeness, and Disclosure

An attacker can rely on a successful verification result being misread as proof of dataset completeness, physical truth of measurements, or authorization for autonomous actuation. An attacker can also obtain sensitive information if disclosed bundles expose more record material than intended for the chosen disclosure class. This profile mitigates those risks only partially: verifier output is required to state the exercised verification scope, a successful result establishes internal consistency and proof binding for the disclosed bundle rather than completeness of all observed or emitted telemetry, and disclosure classes constrain what is expected to be published. Deployments that need stronger completeness, safety, or confidentiality guarantees must add external operational controls, independent observation, and access-control and retention policies that match the sensitivity of the disclosed artifacts.

Even when payloads are withheld, manifests, day artifacts, proof sidecars, and rejection-audit summaries can disclose device-activity metadata such as record counts, rejection rates, outage windows, and anchoring delay. Deployments SHOULD account for that metadata exposure in disclosure and retention policy.

12. Privacy Considerations

Telemetry payloads can include sensitive operational data. Operators should:

- * minimize personally identifiable data in committed artifacts,
- * separate identity metadata from measurement payload when possible,
- * apply retention and access controls, and
- * publish only data appropriate for the chosen disclosure class.

Privacy-preserving disclosures remain valid, but they MUST NOT be described as publicly recomputable unless Class A conditions are met.

Disclosure of verifier-facing manifests, day artifacts, proof sidecars, and rejection-audit summaries can reveal device-activity patterns, outage windows, anchoring cadence, and operational incidents even when payload values are withheld. Operators SHOULD treat these artifacts as metadata-bearing evidence and apply disclosure, retention, and access-control policy accordingly.

13. IANA Considerations

This section follows the guidance in [RFC8126] and provides the complete instructions for the Internet Assigned Numbers Authority (IANA).

13.1. Media Types Registry

IANA is requested to register the following media types in the standards tree of the "Media Types" registry in accordance with [RFC6838]:

Type name: application

Subtype name: verifiable-telemetry-day+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: see Section 11, especially profile and metadata confusion, artifact mutation, proof substitution, disclosure scope, and resource exhaustion.

Interoperability considerations: this media type identifies the authoritative day-artifact family defined by Section 4.4, Section 4.5, Section 4.6, and Section 4.7. A recipient still needs the applicable `commitment_profile_id` to interpret canonical CBOR, hash, and Merkle semantics for a verification claim.

Published specification: this document, especially Section 4.4, Section 4.5, Section 4.6, and Section 4.7.

Applications that use this media type: producers, verifiers, disclosure tools, and archival or audit systems that exchange or retain authoritative day artifacts.

Fragment identifier considerations: no fragment identifier syntax is defined by this document for application/verifiable-telemetry-day+cbor. Fragment identifiers, if present, are processed according to the +cbor structured syntax suffix rules in [RFC8949].

Additional information:

- * Magic number(s): none
- * File extension(s): none
- * Macintosh file type code(s): none

Person & email address to contact for further information: Bilal El Khatabi <elkhatabibilal@gmail.com>

Intended usage: COMMON

Restrictions on usage: none

Author: Bilal El Khatabi

Change controller: IESG

Type name: application

Subtype name: verifiable-telemetry-manifest+json

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: see Section 11, especially profile and metadata confusion, artifact mutation, stale or substituted manifests, disclosure scope, and resource exhaustion.

Interoperability considerations: this media type identifies the verification manifest defined by Section 5, Section 6.4, and Appendix D. The manifest is a metadata and digest-binding surface; it is not an authoritative commitment artifact. A recipient still needs the disclosed `commitment_profile_id` and referenced artifacts to interpret and verify a claim.

Published specification: this document, especially Section 5, Section 6.4, and Appendix D.

Applications that use this media type: producers, verifiers, disclosure tools, publication services, and archival or audit systems that exchange verification manifests.

Fragment identifier considerations: no fragment identifier syntax is defined by this document for application/verifiable-telemetry-manifest+json.

Additional information:

- * Magic number(s): none

- * File extension(s): none

- * Macintosh file type code(s): none

Person & email address to contact for further information: Bilal El Khatabi <elkhatabibilal@gmail.com>

Intended usage: COMMON

Restrictions on usage: none

Author: Bilal El Khatabi

Change controller: IESG

13.2. No CBOR Tag Allocation

This document requests no new CBOR tag allocation. Commitment bytes defined by Section 4.4 forbid CBOR tags, and the authoritative day artifact defined by Section 4.6 does not require additional tag semantics for exchange.

13.3. `commitment_profile_id` Registry

IANA is requested to create the Verifiable Telemetry Ledger Commitment Profiles registry under a new Verifiable Telemetry Ledger Parameters registry group. The registration policy is Specification Required. The initial registration is `verifiable-telemetry-canonical-cbor-v1`, with this document as the reference and "deterministic CBOR canonical record and day artifact commitment profile" as the description. Deployment-private identifiers MUST begin with `x-` and MUST NOT be used as interoperable public profile identifiers.

This document requests no CoAP Content-Format entry.

14. Future Extension Points

The following items are outside the baseline profile defined by this document. They identify extension surfaces that would require separate specification before they are used as interoperability claims.

- * A future specification can register media types for exported evidence bundles or additional artifact families.
- * A future SCITT publication profile can define statement content, submission procedure, and verifier processing for published verification manifests or exported evidence bundles.
- * A future disclosure profile can standardize a universal Class B withheld-material artifact format.
- * A future disclosure profile can adopt COSE-MERKLE proof encodings.
- * A future registry policy can cover disclosure-class and anchor-status vocabularies if multiple independent specifications need shared allocation.
- * A future commitment profile can introduce domain separation for leaf or parent hashes under a distinct `commitment_profile_id`.

15. References

15.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/rfc/rfc8949>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/rfc/rfc6838>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/rfc/rfc3339>>.

15.2. Informative References

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.
- [I-D.ietf-opsawg-rfc5706bis] Claise, B., Clarke, J., Farrel, A., Barguil, S., Pignataro, C., and R. Chen, "Guidelines for Considering Operations and Management in IETF Specifications", Work in Progress, Internet-Draft, draft-ietf-opsawg-rfc5706bis-04, 16 March 2026, <<https://datatracker.ietf.org/doc/draft-ietf-opsawg-rfc5706bis/>>.
- [RFC3552] Rescorla, S. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/rfc/rfc3552>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/rfc/rfc5116.html>>.

- [RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <<https://www.rfc-editor.org/rfc/rfc8439>>.
- [RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Timestamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <<https://www.rfc-editor.org/rfc/rfc3161>>.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <<https://www.rfc-editor.org/rfc/rfc8785>>.
- [RFC8610] Bormann, C. and P. Hoffman, "Concise Data Definition Language (CDDL): A Notational Convention to Express CBOR and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/rfc/rfc8610>>.
- [RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <<https://www.rfc-editor.org/rfc/rfc9162.html>>.
- [SCITT] Birkholz, H., Delignat-Lavaud, A., Fournet, C., Deshpande, Y., and S. Lasker, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture, 2024, <<https://datatracker.ietf.org/doc/draft-ietf-scitt-architecture/>>.
- [COSE-MERKLE] Steele, O., Birkholz, H., Delignat-Lavaud, A., and C. Fournet, "COSE Merkle Tree Proofs", Work in Progress, Internet-Draft, draft-ietf-cose-merkle-tree-proofs, 2025, <<https://datatracker.ietf.org/doc/draft-ietf-cose-merkle-tree-proofs/>>.
- [OTS] OpenTimestamps Project, "OpenTimestamps Protocol and Tooling", 2016, <<https://opentimestamps.org/>>.
- [RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <<https://www.rfc-editor.org/rfc/rfc9334>>.

Appendix A. Example Verification Manifest

This appendix shows a representative verification manifest. A concrete deployment schema can carry additional operational fields, but the example below captures the verification surface described here. The `device_id`, `frame_count`, and `frames_file` fields shown here are operational summary fields for a deployment using the reference framed transport profile; they are not substitutes for canonical-record or day-artifact authority.

```
{
  "version": 1,
  "date": "2025-10-07",
  "site": "an-001",
  "device_id": "device-003",
  "frame_count": 7,
  "records_dir": "records",
  "frames_file": "frames.ndjson",
  "artifacts": {
    "block": {
      "path": "blocks/2025-10-07-00.block.json",
      "sha256": "<64 hex chars>"
    },
    "day_cbor": {
      "path": "day/2025-10-07.cbor",
      "sha256": "<64 hex chars>"
    },
    "day_json": {
      "path": "day/2025-10-07.json",
      "sha256": "<64 hex chars>"
    },
    "day_sha256": {
      "path": "day/2025-10-07.cbor.sha256",
      "sha256": "<64 hex chars>"
    },
    "day_ots": {
      "path": "day/2025-10-07.cbor.ots",
      "sha256": "<64 hex chars>"
    },
    "day_ots_meta": {
      "path": "day/2025-10-07.ots.meta.json",
      "sha256": "<64 hex chars>"
    },
    "x-deployment-summary": {
      "path": "deployment/summary.json",
      "sha256": "<64 hex chars>"
    }
  }
},
```

```
"anchoring": {
  "policy": { "mode": "warn" },
  "channels": {
    "ots": { "enabled": true, "status": "verified" },
    "tsa": {
      "enabled": false,
      "status": "skipped",
      "reason": "disabled"
    },
    "peers": {
      "enabled": false,
      "status": "skipped",
      "reason": "disabled"
    }
  },
  "overall": "success"
},
"verification_bundle": {
  "disclosure_class": "A",
  "commitment_profile_id": "verifiable-telemetry-canonical-cbor-v1",
  "checks_executed": [
    "bundle_disclosure_validation",
    "day_artifact_validation",
    "verification_manifest_validation",
    "record_level_recompute",
    "batch_metadata_validation",
    "day_digest_binding",
    "ots_verification"
  ],
  "checks_skipped": [
    { "check": "tsa_verification", "reason": "disabled" },
    { "check": "peer_quorum_verification", "reason": "disabled" }
  ]
},
"verifier": {
  "verification": {
    "commitment_profile_id": "verifiable-telemetry-canonical-cbor-v1",
    "disclosure_class": "A"
  },
  "channels": {
    "ots": { "enabled": true, "status": "verified" }
  },
  "overall": "success"
}
```

Appendix B. Illustrative Conformance Vector Bundle

The figure below is a reader-oriented sketch of the verifiable-telemetry-canonical-cbor-v1 conformance-vector bundle shape and naming only. It is illustrative and is not itself a complete machine-readable vector corpus.

Wrapped hexadecimal values in this appendix are presentation-only; a verifier or implementer should concatenate adjacent lines without inserting whitespace.

The fixtures below are reader-oriented logical record sketches, not the authoritative CBOR array encoding. They show the current post-projection logical content and the active commitment profile identifier, while the published vector corpus carries the exact encoded bytes and digests.

In these reader-oriented sketches, kind is shown using the lowercase public JSON projection label. In authoritative CBOR commitment bytes, the same semantic family carries the numeric discriminator defined by the current profile; for the fixtures below, custom.raw corresponds to Custom=250.


```
commitment_profile_id:
  verifiable-telemetry-canonical-cbor-v1

fixture record_a:
  pod_id: "00000000000000065"
  fc: 1
  ingest_time: 1772366400
  pod_time: null
  kind: custom.raw
  payload.temp_c: 21.5

fixture record_b:
  pod_id: "00000000000000066"
  fc: 2
  ingest_time: 1772367000
  pod_time: null
  kind: custom.raw
  payload.temp_c: 22.0

fixture record_c:
  pod_id: "00000000000000067"
  fc: 3
  ingest_time: 1772367600
  pod_time: null
  kind: custom.raw
  payload.temp_c: 22.5

class-a-bundle-v1:
  disclosure_class: A
  commitment_profile_id: verifiable-telemetry-canonical-cbor-v1
  required_artifact_1: records/<record-id>.cbor
  required_artifact_2: day/YYYY-MM-DD.cbor
  required_artifact_3: day/YYYY-MM-DD.cbor.ots
  required_artifact_4: day/YYYY-MM-DD.ots.meta.json
  verifier_check_1: bundle_disclosure_validation
  verifier_check_2: day_artifact_validation
  verifier_check_3: record_level_recompute
  verifier_check_4: batch_metadata_validation
  verifier_check_5: day_digest_binding
  verifier_check_6: ots_verification
```

Figure 4

The published machine-readable vector set carries the exact canonical bytes, digests, expected roots, and the applicable `commitment_profile_id`. This appendix remains illustrative and is not an authoritative conformance corpus.

Appendix C. Minimal Device Requirements

This appendix records the minimum expectations for a constrained device that emits framed telemetry under the reference framed transport profile in Section 4.1. It is descriptive of that deployment model, not a claim that all conforming deployments share identical hardware or use the same transport profile.

- * The device is not required to sign assertions or to emit SCITT statements.
- * The device **MUST** be able to produce framed transport messages consistent with Section 4.1, including the nonce construction and uniqueness requirements defined there.
- * Nonce generation **MUST** rely on a cryptographically strong pseudorandom source or an equivalent construction with explicit seed discipline. Weak or predictable generator state is out of profile.
- * The device or an immediately adjacent trusted transport component **MUST** preserve enough durable state to avoid nonce reuse and replay ambiguity across the deployment-defined connectivity window.
- * The device is not required to maintain UTC wall-clock time. If the device carries a local timestamp, that field is deployment input and not the source of UTC day boundaries in this profile.
- * If uplink availability is intermittent, accepted-but-unsubmitted telemetry **MUST** either be durably buffered or be retransmittable from durable local state according to deployment policy.
- * This document does not require one fixed storage budget. The practical tradeoff is between the configured connectivity window, frame rate, local retention policy, and the gateway's replay/admission contract.

In the deployment model assumed here, the intermittency assumption primarily applies on the device-to-gateway path. The gateway is expected to maintain UTC time for ingest_time assignment, day-artifact rollover, and verifier-facing day labels; operational guidance for gateway time synchronization appears in Section 10. External timestamping or publication channels can also be delayed, but those delays do not change the device-side framing requirements.

Appendix D. Verification Manifest CDDL

This appendix gives a representative Concise Data Definition Language (CDDL) ([RFC8610]) shape for the verification manifest. It captures the verification surface described here and leaves room for deployment-specific additions, provided those additions do not remove or change the verification semantics defined here.

Check names are drawn from the standardized vocabulary defined in Section 6.4; deployment-specific extensions MUST use separate names beginning with x- that do not redefine those identifiers.

The relative-path rule below excludes absolute paths and parent-directory traversal so that manifests remain portable across verifier hosts.

```
verification-manifest = {
  "version": 1,
  "date": tstr,
  "site": tstr,
  "device_id": tstr,
  "frame_count": uint,
  "records_dir": relative-path,
  ? "frames_file": relative-path,
  "artifacts": artifacts,
  "anchoring": anchoring,
  "verification_bundle": verification-bundle,
  ? "verifier": verifier-result,
}

artifacts = {
  "block": artifact-ref,
  "day_cbor": artifact-ref,
  "day_json": artifact-ref,
  "day_sha256": artifact-ref,
  ? "day_ots": artifact-ref,
  ? "day_ots_meta": artifact-ref,
  ? "peer_attest": artifact-ref,
  ? "tsa_info": artifact-ref,
  ? "tsa_tsr": artifact-ref,
  * tstr => artifact-ref
}

artifact-ref = {
  "path": relative-path,
  "sha256": hex64
}
```

```
anchoring = {
  ? "policy": { * tstr => any-data },
  "channels": {
    ? "ots": channel-status,
    ? "tsa": channel-status,
    ? "peers": channel-status
  },
  ? "overall": tstr,
  * tstr => any-data
}

verification-bundle = {
  "disclosure_class": disclosure-class,
  "commitment_profile_id": tstr,
  "checks_executed": [* check-name],
  "checks_skipped": [* skipped-check],
  * tstr => any-data
}

check-name = standardized-check-name / extension-check-name

standardized-check-name =
  "bundle_disclosure_validation" /
  "verification_manifest_validation" /
  "day_artifact_validation" /
  "record_level_recompute" /
  "batch_metadata_validation" /
  "day_digest_binding" /
  "ots_verification" /
  "tsa_verification" /
  "peer_quorum_verification"

extension-check-name = tstr .regexp "^x-[A-Za-z0-9._-]+$"

skipped-check = {
  "check": check-name,
  "reason": tstr
}

verifier-result = {
  ? "policy": { * tstr => any-data },
  ? "verification": {
    "commitment_profile_id": tstr,
    "disclosure_class": disclosure-class,
    * tstr => any-data
  },
  ? "checks": { * tstr => bool },
  ? "checks_executed": [* check-name],
```

```

    ? "checks_skipped": [* skipped-check],
    ? "channels": {
      ? "ots": channel-status,
      ? "tsa": channel-status,
      ? "peers": channel-status
    },
    ? "overall": tstr,
    * tstr => any-data
  }

channel-status = {
  ? "enabled": bool,
  "status":
    "verified" / "pending" / "missing" / "failed" / "skipped",
  ? "reason": tstr,
  ? "detail": tstr,
  * tstr => any-data
}

disclosure-class = "A" / "B" / "C"
relative-path = text .regex "^(?!/)(?!.*(?:^|/)\.\.\.(?:/|$)).+ $"
hex64 = text .regex "[0-9a-f]{64}"
any-data =
  nil / bool / int / float / tstr / bstr /
  [* any-data] / { * tstr => any-data }

```

Appendix E. Rejection Audit Record Shape

This appendix gives a representative shape for rejection audit records. These records are operator-audit evidence. They are not accepted telemetry, not Merkle leaves, and not required inputs to the baseline public recomputation claim.

```

{
  "device_id": "pod-001",
  "fc": 7,
  "reason": "duplicate",
  "observed_at_utc": "2026-04-25T12:00:00Z",
  "frame_sha256": "<64 lowercase hex chars>",
  "source": "replay"
}

```

Figure 5

Payload binding rejection reasons use protocol-neutral names. They describe mismatches between a decrypted logical payload and its enclosing frame header regardless of the payload serialization used by a deployment.

```
rejection-audit-record = {  
  "device_id": tstr,  
  "fc": uint / nil,  
  "reason": rejection-reason,  
  "observed_at_utc": tstr,  
  "frame_sha256": hex64,  
  "source": "parse" / "decrypt" / "replay"  
}
```

```
rejection-reason =  
  "line_too_long" /  
  "invalid_json" /  
  "not_dict" /  
  "missing_frame_fields" /  
  "unexpected_frame_fields" /  
  "invalid_hdr" /  
  "invalid_frame_types" /  
  "missing_hdr_fields" /  
  "unexpected_hdr_fields" /  
  "invalid_hdr_types" /  
  "dev_id_range" /  
  "msg_type_range" /  
  "fc_range" /  
  "flags_range" /  
  "invalid_ingest_profile" /  
  "unsupported_flags" /  
  "unknown_device" /  
  "missing_salt8" /  
  "invalid_base64" /  
  "salt8_length" /  
  "ck_up_length" /  
  "nonce_length" /  
  "tag_length" /  
  "empty_ciphertext" /  
  "ciphertext_too_large" /  
  "nonce_salt_mismatch" /  
  "nonce_fc_mismatch" /  
  "decrypt_failed" /  
  "payload_device_id_mismatch" /  
  "payload_fc_mismatch" /  
  "duplicate" /  
  "out_of_window"
```

Acknowledgments

Early structural drafts of this document were prepared with AI writing assistance. All technical content, design decisions, and normative requirements were reviewed by the author.

The author thanks the OpenTimestamps project for the public calendar infrastructure used during validation.

The author thanks Joe Clarke for OPSDIR review feedback that improved the operational considerations and deployment guidance.

Author's Address

Bilal El Khatabi
Morocco
Email: elkhatabibilal@gmail.com