OTAC 0.1.1 — Canonical Temporal Attestation Capsule (JCS/CBOR, PQC) with Test Vectors and FRAND-Z Terms

VERSION: 0.1.1 (Defensive Publication Draft)
DATE: 26 Nov 2025
AUTHORS: Project NyxQuantum (Sigitas Andrijauskas Sumlinskas, et al.)
CONTACT: NyxQuantum@proton.me

CHANGE LOG:
0.1 → 0.1.1: Added `canonical_profile_id`, `canonical_hash_alg`; domain separation tag (DST) and `tac_id`; key metadata; `shard_auth_tag`; offline verification steps; privacy guidance; COSE/JOSE interop note; clearer FRAND-Z; versioning clause for `canonical_profile_id`; recommended 64 KiB size limit; complete JSON/CBOR examples in Appendix B.

***

ABSTRACT

OTAC 0.1.1 defines a canonical temporal evidence object for critical systems. The format mandates deterministic encodings (JSON Canonicalization Scheme, RFC 8785; deterministic CBOR, RFC 8949), a common canonicalization pipeline producing identical canonical bytes across encodings, required fields (`version`, `canonical_profile_id`, `canonical_hash_alg`, `policy_id`, `sovereign_time`, `identity_anchor`, `prev_hash_link`, `evidence_vector`, `policy_digest`, `pqc_signature`, `tac_id`) and post-quantum signatures (e.g., ML-DSA / SLH-DSA). OTAC MAY be sharded via erasure codes (k-of-n) with per-shard headers and authentication. This draft includes a verification procedure, templates for test vectors, interoperability guidance (EaaS/TQT/QUIC-TAC), and FRAND-Z terms. OTAC is transport-agnostic and may be encapsulated in files, messages, or streams without changing its canonical definition.
***

TABLE OF CONTENTS

***

1. Scope and Goals

This document specifies OTAC 0.1.1: a canonical, signed temporal evidence object with offline verifiability, dual deterministic encodings (JSON/CBOR), a common canonicalization pipeline producing identical canonical bytes, optional erasure-coded sharding, and hooks for Temporal Quorum Time (TQT) and Evidence-as-a-Service (EaaS).

Non-goals: transport semantics (e.g., congestion control, multiplexing), quorum logic, or Δt policies. OTAC is transport-agnostic and may be encapsulated in files, messages, or streams without changing its canonical definition.

***

2. Terminology and Normative Language

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 RFC 2119 and RFC 8174 when, and only when, they appear in all capitals.

JSON uses UTF-8. CBOR uses deterministic encoding per RFC 8949. Time values use UTC-Z and RFC 3339.

***

3. OTAC Data Model

3.1 Required Fields (MUST)

- `version` (string) — e.g., `"0.1.1"`.
- `canonical_profile_id` (string/URN) — locks canonical ordering and normalization rules. Example: `"urn:otac:canon:2025-11-26:jcs-cbor-v1"`.
- `canonical_hash_alg` (string) — e.g., `"SHA-256"` or `"SHA3-256"`.
- `policy_id` (string) — policy profile identifier.
- `sovereign_time` (string) — UTC-Z RFC 3339 timestamp indicating attested issuance time.
- `identity_anchor` (object|string) — certificate/anchor digest, DID reference, or equivalent trust anchor.
- `prev_hash_link` (hex string) — hash of prior OTAC canonical bytes (same `canonical_hash_alg`). For genesis, `prev_hash_link` MUST be the all-zero hex string whose length equals the digest size of `canonical_hash_alg` (e.g., 64 hex zeros for SHA-256).
- `evidence_vector` (object) — domain-specific evidence payload (minimised; see §8).
- `policy_digest` (hex string) — digest over the policy snapshot used for this OTAC.
- `pqc_signature` (object) — see §5.
- `tac_id` (string/URN) — stable identifier. A common pattern is: `tac_id = "urn:otac:" + canonical_hash_alg.lower() + ":" + <hex(canonical_hash_bytes)>`.

3.2 Optional Fields (MAY)

- `shard_header` — `{index, total, params}` for k-of-n shards.
- `shard_auth_tag` — per-shard MAC/tag (or signature) to authenticate shards before reassembly.
- `privacy_stamp` — e.g., `{epsilon, delta, k, scope}`.
- `lineage_id` — identifier linking related OTACs across a process or pipeline.
- `tqt_meta` — time-quorum provenance (sources, weights, path-asymmetry bounds).

- `quantum_entropy_provenance` — record for randomness source and health (e.g., quantum or hardware entropy providers).
- `compliance_stamp` — profile(s) met (e.g., DORA/IEC-62443 mapping IDs).
- `content_type` — short type for `evidence_vector` (e.g., `"pop"`, `"hec"`, `"ml-stage"`).
- `key_rotation_info` (object) — hints about previous keys, future keys, and rotation policy identifiers used by the deployment.

3.3 Encoding Constraints \& Limits (MUST/SHOULD)

- Deterministic JSON per RFC 8785; deterministic CBOR per RFC 8949.
- All strings MUST be NFC Unicode; timestamps MUST be UTC-Z (RFC 3339).
- Numbers MUST be finite; NaN/Inf are forbidden.
- A **recommended** (non-normative) maximum OTAC size is 64 KiB; larger payloads SHOULD use external references with hash pointers in `evidence_vector`.
- Hashes MUST specify algorithm via `canonical_hash_alg`.

`canonical_profile_id` SHOULD correspond to a published profile whose definition is itself integrity-protected (e.g., by a signed document, a separate OTAC, a transparency log, or an equivalent mechanism). Verifiers SHOULD maintain an allow-list of trusted `canonical_profile_id` values. New `canonical_profile_id` values MAY be introduced in future profiles without invalidating existing OTAC chains, provided verifiers explicitly trust each profile identifier they accept.

Downgrade protection: verifiers MUST NOT automatically fall back to a less strict `canonical_profile_id` without an explicit policy.

***

4. Canonicalization Pipeline (JCS/CBOR → Canonical Bytes) (MUST)

Both JSON and CBOR encodings feed a common normalization pipeline identified by `canonical_profile_id`, yielding identical canonical bytes for the same logical object.

Let `canonical_bytes` be the output of the normalization pipeline. Let `canonical_hash = H(canonical_bytes)` using `canonical_hash_alg`. The signature input MUST be:

`DST || canonical_bytes`

where `DST = "OTAC-0.1.1/signature"` (domain separation tag, UTF-8).

The `prev_hash_link` MUST be computed over the canonical bytes of the previous OTAC using `canonical_hash_alg`. The canonical hash bytes refer to the digest of these canonical bytes under `canonical_hash_alg`.

***

5. Post-Quantum Signatures \& Key Metadata

`pqc_signature` (object) MUST contain:

- `alg` ∈ { `"ML-DSA"`, `"SLH-DSA"` } (algorithm identifier, aligned with NIST ML-DSA / SLH-DSA families).
- `key_format` ∈ { `"COSE"`, `"JOSE"`, `"RAW"` } — how the public key is expressed if included.

- `key_id` (string) — implementer’s key identifier.
- `pubkey_fingerprint` (hex, e.g., SHA-256 of public key).
- `valid_from`, `valid_to` (RFC 3339, OPTIONAL).
- `sig` (bytes/hex) — signature over `DST || canonical_bytes`.

Signatures MUST be generated inside a PQC-capable Secure Element / HSM meeting suitable assurance (e.g., ISO/IEC 19790 / FIPS 140-3 class), with forward-secure rotation and validity windows where applicable. Implementations MUST use cryptographically secure random number generators suitable for PQC key and signature generation.

The post-quantum signature MUST cover at least `version`, `canonical_profile_id`, `canonical_hash_alg`, `policy_id`, `prev_hash_link`, `identity_anchor`, `policy_digest`, and any `quantum_entropy_provenance` present, as they are part of the canonical bytes.

Revocation and emergency key rotation MUST be handled at system level (e.g., via CRL/OCSP-like or transparency mechanisms); OTAC can carry `valid_to` and `key_rotation_info` hints but does not define revocation semantics.

***

6. Sharding / Erasure Coding (k-OF-n)

Sharding employs standard (n, k) erasure codes (e.g., Reed-Solomon or Locally Repairable Codes, LRC). Each shard MUST include `{index, total, params}` in `shard_header`. A `shard_auth_tag` (per-shard MAC or signature) SHOULD authenticate shards before reassembly.

Any k valid shards MUST reconstruct the original canonical bytes, whose digest MUST match the stored hash used for signature verification. Recovery MUST validate all per-shard tags and then verify the whole OTAC signature.

***

7. Interoperability (TQT, EaaS, QUIC-TAC, PoP-Cam; COSE/JOSE)

- **TQT**: When available, `tqt_meta` SHOULD list sources (e.g., NTS/PTP/NTP/other), trust weights, and path-asymmetry bounds.
- **EaaS**: Custody services MAY ingest OTACs, verify timestamps versus TQT, reseal them with PQC signatures, and retain erasure-coded copies under tenant isolation.
- **QUIC-TAC / PoP-Cam**: Systems MAY reference `tac_id` or embed summaries of `evidence_vector`; full OTAC records can remain off-path where appropriate.
- **COSE/JOSE**: Implementations MAY wrap `pqc_signature` in COSE_Sign1 or JWS (detached); `sig` MUST always be computed over `DST || canonical_bytes`.

OTAC is intended to be compatible with Remote ATtestation procedureS (RATS) architectures as a format for Evidence or Attestation Results, without redefining RATS interaction models.

***

8. Security \& Privacy Considerations

- **Replay**: `prev_hash_link` establishes chain continuity; chains MUST be strictly monotonic in intended contexts. In low-state or multi-origin deployments, `evidence_vector` SHOULD include a nonce and/or a context-specific sequence number to help detect replays within a short time window, in addition to `prev_hash_link`.
- **Canonicalization pitfalls**: `canonical_profile_id` and a unified pipeline

avoid cross-encoding ambiguities. Verifiers SHOULD restrict themselves to known, integrity-protected profiles.
- **Key rotation**: Enforced at EaaS/service layer; fields `valid_from`, `valid_to`, and `key_rotation_info` act as hints only (see §5).
- **Minimization**: `evidence_vector` SHOULD contain only necessary data; sensitive payloads SHOULD be referenced via hash pointers and, where appropriate, stored or encrypted out-of-band.
- **Differential Privacy**: Budgets (`privacy_stamp`) MUST be tracked and enforced externally; OTAC only carries the stamp.
- **Entropy provenance**: Post-quantum signatures SHOULD use a high-assurance entropy source; when present, `quantum_entropy_provenance` MAY record information about entropy origin and health for forensic and audit purposes.

Implementations MUST use cryptographically secure random number generators suitable for PQC key and signature generation (see also §5).

***

9. Offline Verification Procedure (Normative)

Given an OTAC object:

1. Validate field presence and lengths; check `version`, `canonical_profile_id`, `canonical_hash_alg`.
2. Canonicalize to bytes via the pipeline referenced by `canonical_profile_id`.
3. Compute `tac_id = "urn:otac:%s:%s" % (canonical_hash_alg.lower(), hash(canonical_bytes))`; compare to provided `tac_id`.
4. Recompute `prev_hash_link` (if not genesis) from the previous OTAC’s canonical bytes and compare.
5. Build `DST || canonical_bytes`; verify `pqc_signature.sig` using `alg`, `pubkey_fingerprint` (and provisioned public-key material).
6. If sharded, first authenticate shards via `shard_auth_tag`, reassemble, then repeat steps 2–5 on the reassembled OTAC.
7. Optionally validate `tqt_meta` against local TQT policy.

A reduced verifier profile for resource-constrained devices MAY implement steps 1–3 and 5 only, delegating heavy checks (e.g., full chain validation, DP budget checks, compliance mapping) to a backend.

***

10. Test Vectors (Templates)

10.1 Vector A (JSON → Canonical Bytes)

Input JSON (pre-canonicalization):

```json
{
  "version": "0.1.1",
  "canonical_profile_id": "urn:otac:canon:2025-11-26:jcs-cbor-v1",
  "canonical_hash_alg": "SHA-256",
  "policy_id": "POLICY-EXAMPLE-A",
  "sovereign_time": "2025-11-26T12:00:00Z",
  "identity_anchor": "did:example:ABC123",
  "prev_hash_link": "00...00",
  "evidence_vector": {
    "type": "demo",
    "hash": "FILL"
```

```
  },
  "policy_digest": "FILL",
  "quantum_entropy_provenance": "FILL",
  "pqc_signature": {
    "alg": "ML-DSA",
    "key_format": "COSE",
    "key_id": "KEY1",
    "pubkey_fingerprint": "FILL",
    "sig": "FILL"
  },
  "tac_id": "urn:otac:sha-256:FILL"
}
```

Canonical bytes (hex) — per §4:

`FILL`

Canonical hash (hex) — per `canonical_hash_alg`:

`FILL`

10.2 Vector B (CBOR → Canonical Bytes)

Input CBOR (diagnostic or hex):

```text
FILL
```

Canonical bytes (hex) — per §4:

`FILL`

Canonical hash (same as JSON path above):

`FILL`

***

11. FRAND-Z Licensing Terms (Summary)

- Royalty-free, worldwide, non-exclusive license to Essential Claims needed to implement OTAC 0.1.1 conformant with this document.
- Reciprocity: each implementer grants an equivalent royalty-free license to its Essential Claims covering OTAC 0.1.1.
- Defensive suspension: license may be suspended against entities initiating offensive patent actions against conformant implementations.
- No trademark rights; provided “AS IS”.

The full license text is provided in `LICENSE.txt` in the reference package.

***

12. References

Normative

- RFC 8785 — JSON Canonicalization Scheme (JCS).

- RFC 8949 — Concise Binary Object Representation (CBOR) — including deterministic encoding rules.
- RFC 3339 — Date and Time on the Internet: Timestamps.
- RFC 2119 / RFC 8174 — Key words for use in RFCs to Indicate Requirement Levels.

Informative

- RFC 8915 — Network Time Security (NTS).
- RFC 9334 — Remote ATtestation procedureS (RATS) Architecture.
- NIST PQC standards: ML-DSA (Dilithium), SLH-DSA (SPHINCS+).
- Erasure Codes (Reed-Solomon / LRC) — general literature.
- ISO/IEC 19790 — Security requirements for cryptographic modules (HSM/SE context).
- COSE/JOSE families for key and signature container formats.
- PTP (IEEE-1588v2), NTP — time sources for TQT in EaaS contexts.

***

Appendix A — Minimal `verify.py` (Template)

```python
#!/usr/bin/env python3
# OTAC 0.1.1 – minimal verifier (template)
# Implement JCS/CBOR + the canonicalization pipeline selected by canonical_profile_id.

import json, sys, hashlib

DST = b"OTAC-0.1.1/signature"

def canonicalize_to_bytes(obj):
    # TODO: Implement per canonical_profile_id (JCS/CBOR → normalized bytes)
    raise NotImplementedError

def h(data, alg="SHA-256"):
    return getattr(hashlib, alg.lower().replace("-", ""))(data).hexdigest()

def main():
    data = json.load(sys.stdin)
    cbytes = canonicalize_to_bytes(data)
    print("CANONICAL_BYTES_HEX:", cbytes.hex())
    alg = data.get("canonical_hash_alg", "SHA-256")
    print("HASH:", h(cbytes, alg))
    print(
        "TAC_ID:",
        "urn:otac:%s:%s"
        % (alg.lower(), h(cbytes, alg)),
    )
    # Signature verification occurs with provisioned PQC public key material (out of scope here).

if __name__ == "__main__":
    main()
```

***

Appendix B — Vector Files Layout

```text
/otac-0.1.1/
  OTAC-0.1.1.pdf
  OTAC-0.1.1.md        (this specification, optional alongside PDF)
  examples/
    genesis.json       (genesis OTAC, prev_hash_link = 0x00…)
    plc_event.json     (HEC-Sentinel PLC event)
    ml_stage.json      (ML pipeline stage – Dataset Guard)
    cold_chain.json    (cold-chain trace)
  tools/
    verify_standalone.py
    README.md
  LICENSE.txt
```

— END —