QUADNITION PROTOCOL – GOLD SPEC v1.0

Charter · Technical Annex · Merkle v2 · Verifier · Acceptance Checklist

1. OFFICIAL CHARTER (PUBLIC FACING)

Quadnition is a cryptographically verifiable human-in-the-loop protocol.

It proves that a real human created a piece of media at a real time, on a real device, and that the record has not been silently tampered with since.

Quadnition establishes four coordinated proofs across thought, sound, image, and machine state, bound together into a single session record (the SessionBundle).

Quadnition does not claim to detect consciousness or subjective awareness. It proves origin, integrity, continuity, timing, and human involvement – nothing more, nothing less.

Everything needed to independently verify a Quadnition session is provided in this document: the Technical Annex v1.0, the reference verifier, and sample sessions.

2. TECHNICAL ANNEX v1.0

2.1 Core Session Structure

At session start:

• t0 = session start time (monotonic clock).
• device_seed = per-device random value.
• device_key = symmetric key, unique per device.

Define:
session_id = BLAKE3(device_seed || t0)

The session_id binds all proofs and logs for this capture.
The device_key is used for HMAC-based chains and seals.

2.2 The Four Proofs

THOUGHT-PROOF (Human-in-the-loop)

Goal: Prove that a real human responded to unpredictable challenges in real time.

Inputs:
• C – challenge prompt (cryptographically random).
• R – human response (typed or spoken, normalized).
• Δt – time between C displayed and R received.
• F – device fingerprint.
• session_id.

Transform:
H_thought = HMAC(device_key, C || R || Δt || F || session_id)

Constraint: Δt must fall within a human reaction-time window.

SOUND-PROOF (Audio continuity)

Goal: Prove that the audio stream was captured live and not overdubbed or spliced.

Inputs:
• audio.wav – raw or PCM audio stream.
• timing_marks – embedded markers at fixed intervals.
• mic_id – microphone identifier.

Transform:
H_sound = BLAKE3(audio_chunks || timing_marks || mic_id)

IMAGE-PROOF (Video continuity)

Goal: Prove continuity and authenticity of the video stream.

Inputs:
• frames – raw / near-raw frames.
• sensor_id – camera sensor identifier.
• exposure_i, shutter_metadata_i – per-frame metadata.

Per-frame:
frame_hash_i = BLAKE3(frame_i || sensor_id || exposure_i || shutter_metadata_i)

Aggregate:
H_image = BLAKE3(frame_hash_0 || … || frame_hash_n)

MACHINE-PROOF (Append-only log)

Goal: Bind the sequence of events as an append-only chain.

Inputs:
• event_i – JSON event payload.
• seal_0 – initial nonce.
• device_key – symmetric key.

Transform:
seal_0 = device_nonce
seal_i = HMAC(device_key, seal_{i-1} || event_i)

Any modification to event_i or their order breaks the chain.

2.3 SessionBundle Format

A Quadnition session is delivered as a SessionBundle containing:

• video.mp4 – encoded video stream.
• audio.wav – raw/PCM audio stream.
• eventlog.json – machine log.
• manifest.json – top-level metadata and commitments.

manifest.json structure (conceptual):


```
{
"session_id": "...",
"hashes": {
"video": "",
"audio": "",
"log": ""
},
"proofs": {
"thought": "H_thought",
"sound": "H_sound",
"image": "H_image",
"machine": "H_machine"
},
"timing": {
"t_start": "...",
"t_end": "...",
"drift_max": 150
},
"merkle": {
"version": 2,
"canon": "JCS",
"hash": "sha256",
"root": "...",
"index": 1,
"path": ["..."]
}
}
```


2.4 Verification Procedure

A reference verifier performs:

1. Unpack the SessionBundle (zip).
2. Recompute BLAKE3 hashes for video, audio, and eventlog; compare against manifest.hashes using constant-time comparison.
3. Recompute H_thought, H_sound, H_image, and the machine-proof seal chain.
4. Verify Merkle v2 inclusion for the chosen leaf (e.g., manifest) using version, root, index, and path.
5. Check timing constraints: t_end >= t_start; duration within policy; drift_max below threshold.
6. Return PASS or FAIL with explicit reasons.

Example output:

PASS: all proofs consistent
FAIL: video hash mismatch
FAIL: machine seal chain broken at event 12
FAIL: thought-proof timing window violated

2.5 Formal Claims and Non-Claims

If verification returns PASS, then with high probability:

• Audio, video, and machine log come from a single continuous session. • A real human participated by responding to unpredictable challenges in real time. • Any modification above a minimal threshold to audio, video, or log would cause verification to fail. • Session timing is consistent with a live capture between t_start and t_end. • Cross-modal references (events vs. media timestamps) are internally consistent.

Quadnition explicitly does NOT claim:

• To detect or measure consciousness, sentience, or qualia. • To guarantee that the human was truthful or acting with good intent. • To secure devices already compromised at hardware or OS level. • To prevent coercion or social engineering.

3. MERKLE v2 SPECIFICATION

Merkle v2 is the tree construction used for Quadnition segment and receipt commitments.

Goals:
• Deterministic, canonical leaves.
• Collision-resistant node construction.
• Clear, versioned odd-leaf policy.
• Interoperable across languages and platforms.

Canonicalization:
• Values are encoded using a JCS-style canonicalization (sorted keys, stable formatting).

Hashing:
• leaf_hash = sha256(0x00 || canon(leaf_bytes))
• node_hash = sha256(0x01 || left_hash || right_hash)

Odd-leaf policy:
• RFC6962-style promotion: if a level has an odd count, the last node is promoted unchanged.

Versioning:
• version = 2
• canon = "JCS"
• hash = "sha256"

Public API (conceptual):
• buildTree(leaves, version=2)
• buildProof(leaves, index, version=2)
• verifyProof(leafValue, index, path, rootHex, version=2)

4. VERIFIER OVERVIEW

The reference verifier is a CLI tool (quad_verify.py) that:

• Accepts a SessionBundle zip file.
• Unpacks it into a temporary directory.
• Re-hashes video, audio, and eventlog (BLAKE3 or SHA-256 as configured).
• Replays the machine seal chain for structural integrity.
• Verifies Merkle v2 inclusion proofs where present.
• Performs basic timing and structural sanity checks.
• Emits a single PASS or FAIL with a human-readable explanation.

Example:

```
$ python quad_verify.py session_bundle.zip
PASS: all structural checks passed
```

5. ACCEPTANCE CHECKLIST (GOLD CRITERIA)

For a Quadnition implementation to be considered GOLD, the following must hold:

• Deterministic leaves:
– JCS-style canonicalization used for all Merkle v2 leaves.
– No raw JSON.stringify for hashed structures.

• Collision hygiene:
– Leaf and node hashes are domain-separated (0x00 and 0x01 prefixes).

– Odd-leaf promotion is explicitly versioned as version: 2.

• Interop clarity:
– Each Merkle-bearing segment declares {version, canon, hash}.
– Golden vectors are published (leaves, expected root, inclusion paths).

• Verifier UX:
– quad_verify.py (or equivalent) reconstructs the root and validates at least three inclusion proofs from public fixtures.
– PASS/FAIL messages are explicit and reproducible across machines.

• Transparency:
– The Charter and Technical Annex PDFs are hashed (SHA-256) and the hashes published.
– Roots and key anchors are optionally signed (e.g. Ed25519) and logged in a transparency or receipt log.

When all of the above are satisfied, Quadnition can be independently verified,
ported, and red-teamed without ambiguity, and the protocol is suitable for public scrutiny.