EMET: An Authority-Incapable Byte-Level Integrity Witness for AI-Assisted Work

Zain Dana Harper
Seattle, WA | ORCID 0009-0001-7175-5393
https://github.com/HarperZ9/emet

July 2026

Abstract

Recent verifiers for AI-generated outputs are converging on a single direction: *richer learned models* that emit calibrated confidence, multi-category error diagnoses, and reasoning chains. This paper describes the opposite bet, shipped and frozen at version 1.0.0. EMET is a deliberately *minimal, model-free, byte-level* integrity witness whose verdict is a pure function of recomputed bytes, with no language model in the loop. Its central design constraint is not richer output but an *authority-incapable output type*: a closed verdict lattice {MATCH, DRIFT, UNVERIFIABLE} that, by construction, cannot express TRUSTED, APPROVED, or SAFE. We argue that a verifier which cannot grant authority is adversarially simpler to reason about: there is no model to persuade and no token that can launder a byte-level match into a permission. We specify the witness as a frozen, normative contract (RFC 2119 keywords, a pinned output grammar, a minimal trusted computing base), and we report a conformance harness in which four independent implementations—a Python reference plus Rust, Node.js, and Go ports, the latter two written against the specification and vectors alone—agree on 35 core vectors in continuous integration. We are explicit about what this does and does not establish: same-author agreement demonstrates internal consistency and implementability, *not* independent re-derivability, which we name as an open, falsifiable community deliverable. The artifact, specification, conformance vectors, and four implementations are public.

1 Introduction

A verifier of AI-assisted work faces a recurring temptation: to grow richer. Each added output category—a confidence score, a severity label, a suggested fix—makes the verifier more useful and, quietly, more authoritative. The risk is structural rather than incidental: a verifier that can emit APPROVED or SAFE is a target for the same pressure tactics that affect any authority, and an LLM-in-the-loop verifier can be persuaded, jailbroken, or confused in ways a byte comparison cannot.

EMET takes the inverse position. It is an *externally-anchored* integrity layer that decides exactly one question—*do these bytes re-derive to the same hash as an authorized anchor?*—and refuses every adjacent question. Its output type is a closed lattice of three values: MATCH (re-derivation agreed), DRIFT (a confirmed difference), and UNVERIFIABLE (the check could not be completed). The lattice is closed in the formal sense: no implementation may define, emit, or accept any other verdict, and in particular may not emit any value asserting authority or permission. Inability is never reported as trust; it is reported as UNVERIFIABLE with a stable machine reason code.

This is the load-bearing distinction we draw against the current trend. Where concurrent work builds *more expressive* verifiers—structured reasoning agents with process rewards [1], provenance-native traces with per-document influence [2], governed context vaults [3]—EMET

trades expressiveness for a property those systems cannot offer by construction: *the verifier cannot be turned into an authority*, because the authority words are not in its vocabulary. The contribution is not a better judgment of meaning (EMET is, by design, semantically blind); it is a witness whose verdict is a pure function of recomputed bytes, so there is no learned channel through which persuasion can be applied. We argue this property by construction and name adversarial evaluation of it as an open deliverable (Section 7).

The paper makes four contributions:

1. **Conceptual:** an authority-incapable witness, formalized as a closed verdict lattice and six testable boundary invariants (Section 3).
2. **Technical:** a minimal-TCB byte-level witness with a portable, content-addressed receipt format that travels off-machine where anchors cannot (Sections 3–4).
3. **Methodological:** a frozen, normative specification refined by independent-from-reference implementations, where each port surfaced specification under-determination that the conformance vectors then pinned (Section 5).
4. **Integrity:** an explicit, falsifiable scoping of what same-author agreement proves and what it does not, with the different-author implementation named as an open challenge (Section 7).

2 Background and related work

Learned and structured verifiers. SEVA [1] trains a verification agent with reinforcement learning to emit evidence alignments, reasoning chains, calibrated confidence, and a six-category error diagnosis. ProvenAI [2] decomposes transparency in multi-hop question answering into answer correctness, citation fidelity, and per-document influence under leave-one-resource-out intervention. Both build *richer, learned* verifiers; EMET builds a *poorer, deterministic* one, on the principle that a verifier with no learned component has no persuasive surface.

Provenance and context governance. ContextNest [3] formalizes context governance—provenance, version identity, integrity, point-in-time reconstruction—as a layer beneath retrieval-augmented generation. Minos [4] performs LLM-driven provenance backward tracking for forensic analysis. These systems are natural *consumers* of a leaf witness: EMET can supply the byte-level integrity fact that a governance stack invokes, without competing with the governance layer itself.

Constrained verifiable agents. *Making Failure Safe* [5] shifts agent output from free-form code to typed, rule-checked configurations. EMET shares the "shift toward the checkable" instinct but applies it at the integrity layer rather than the code-generation layer.

Foundations. EMET rests on content addressing and Merkle/tamper-evident logging, capability-based security (the witness holds no key and takes no action), and the long line of work on reproducible builds. Its departure is not cryptographic novelty but the deliberate removal of the authority channel from the output type.

3 Design

3.1 The closed verdict lattice

Every integrity judgment EMET emits is exactly one of {MATCH, DRIFT, UNVERIFIABLE}. This enumeration is closed: an implementation **must not** emit TRUSTED, APPROVED, SAFE, or any value asserting authority or permission. Absence of drift is reported as MATCH (re-derivation agreed) or UNVERIFIABLE (no raw-byte anchor), never as trust. This is the first of six boundaries—*facts, not authority*—encoded directly in the output type.

Auxiliary judgments are likewise closed and never map to an authority word: coherence emits {COHERENT, VIEW_DIFFERS_FROM_SOURCE}; corroboration emits {CORROBORATED, QUARANTINE_REAL a marker scan emits a non-negative integer count.

3.2 Hashing and identity

The identity of an artifact is SHA-256 over its *exact raw bytes*. An implementation **must not** normalize, transcode, or canonicalize before hashing: line endings, encoding, and whitespace are part of the artifact, and detecting a CRLF rewrite is a feature. Bytes are read through a raw channel (`read_binary`), never a mediated view; with no raw channel the result is UNVERIFIABLE.

3.3 UNVERIFIABLE, never TRUSTED

If an implementation cannot obtain raw bytes, find an anchor, or complete a check, it **must** report UNVERIFIABLE with a *stable machine reason code* (not prose) and **must not** substitute a default, a cached value, or a trust assertion. The governed codes are closed: `E_NOT_FOUND`, `E_NO_RAW_CHANNEL`, `E_NO_ANCHOR`, `E_NO_CORPUS`, `E_NO_CORPUS_VERSION`, `E_NO_SECOND_READ_PATH`, `E_LOG_CORRUPT`. Inability is never trust, and never a difference.

3.4 Exit codes as the verdict class

The exit code is the verdict class as an integer: 0 (the property held: all MATCH, COHERENT, CORROBORATED, INTACT); 1 (a negative finding: DRIFT or divergence); 2 (could not be checked: UNVERIFIABLE); 3 (markers detected); 64 (usage error). For multiple targets, a confirmed difference dominates an inability to check (exit 1 dominates exit 2). The exit code is data plus an advisory signal; it allows and denies nothing—enforcement is a downstream decision on owner-controlled infrastructure.

3.5 The six boundaries as testable invariants

The design thesis is encoded as six invariants, each falsifiable by a test:

1. **Facts, not authority**—the lattice is closed; no codepath emits outside it or produces TRUSTED.
2. **Attests, never adjudicates a model-safety decision**—EMET operates only on artifact, byte, and provenance facts; no command takes a model-safety or content decision as input.
3. **Outside, never inside**—targets are read by raw bytes; EMET must not be hosted by, or routed through, the system it audits.
4. **Advisory by default**—a verdict is data plus an exit code; EMET allows, denies, blocks, or enforces nothing of its own accord.
5. **Re-derivable**—every verdict is reproducible from the same specification version, corpus version, and bytes; no secret, no held key.
6. **Zero actuation on the audited target**—EMET writes only to its own implementation-private stores (the anchor store, the hash-chained log, the redacted `<file>.refused` copy); the operator is the single actuator over the audited world.

3.6 Trusted computing base

The core depends only on the language runtime and standard library (CPython plus `hashlib`, `json`, `os`, `subprocess`) and adds no third-party runtime dependency. Optional adapters (sign-

ing, SARIF or in-toto emission) live in separate packages; the minimal-TCB guarantee applies to the named core only.

3.7 Tamper-evident audit chain

The audit log is JSON-lines. Each entry carries `kind`, `fact`, `prev`, and `chain`, where

$$\texttt{chain} = \texttt{SHA256}\big(\texttt{prev} + \texttt{kind} + \texttt{canonical\_json}(\texttt{fact})\big),$$

`prev` is the prior entry's chain value, and the genesis `prev` is 64 zeros. Binding `kind` means relabeling an entry's operation (e.g. `anchor`→`refuse`) is itself tamper. `audit` recomputes the chain *and* checks linkage: each stored `prev` must equal the prior entry's chain, so a forged-but-internally-consistent re-chained suffix is caught, not only a single-entry edit.

Canonical JSON across languages. `canonical_json` is the serialization with keys sorted, "`, `" and "`: `" separators, and `ensure_ascii` escaping, UTF-8-encoded before concatenation. A language's default encoder does *not* produce this form: Go's `encoding/json` HTML-escapes `< > &`, and neither Go nor JavaScript's `JSON.stringify` matches the spacing or sorts keys. A conforming implementation must therefore hand-roll or post-process its serializer. Pinning this byte form is what lets an auditor re-derive a chain that another implementation wrote.

4 The portable witness receipt

The anchor store is implementation-private: a raw anchor/VERIFY verdict cannot leave the machine that produced it. The *portable witness receipt* (`emet-witness-receipt/v1`) is the standardized, cross-implementable form that can travel, so a different party re-derives and checks it on their own machine with zero shared state, zero trust in the producer, and zero network access.

A receipt is a JSON object carrying one or more EMET verdicts plus the re-derivation method. Its content address `receipt_id` = `SHA256` over the canonical JSON of the receipt with the `receipt_id`, `signature`, and `witness` fields excluded. The witness block (implementation name and that implementation's own artifact-of-record hash) is excluded from the address because it is producer identity, intrinsically per-implementation; including it would make the same subject/verdict/spec/timestamp hash to different identifiers across implementations, defeating the guarantee. The guarantee: the same subject bytes, specification version, corpus version, and issuance timestamp yield a byte-identical `receipt_id` across conforming implementations; tampering any addressed field re-hashes to a different identifier.

The offline verifier `emet check` is stateless (no anchor store, no log, no network, no shared key). It re-derives `receipt_id` and compares (mismatch → RECEIPT_TAMPERED), optionally re-hashes live subject bytes (`--recompute-from-paths`), and returns a member of the closed receipt lattice {RECEIPT_VALID, RECEIPT_TAMPERED, RECEIPT_UNVERIFIABLE}, with TAMPERED dominating UNVERIFIABLE as in Section 3.4. A receipt asserts a *fact of re-derivation*, never authority: RECEIPT_VALID means the receipt's content address is intact at check time; it maps to no authority word.

5 Multi-language conformance

Conformance is defined Extensionally: an implementation conforms at a given specification version if and only if it satisfies every normative **must** and produces the expected verdict and exit code for every vector in `conformance/vectors.json` at the stated corpus version.

Four implementations pass the core vectors in continuous integration: the Python reference, plus Rust, Node.js, and Go ports. The Node.js and Go ports were written *against the specification and the vectors alone*, not the reference code. Each port surfaced specification underdetermination that the vectors then pinned. The clearest example is the marker count: the specification initially left `count` implicit, and an independent reimplementation forced the rule—a non-overlapping leftmost scan in corpus order—which the vectors `refuse-three-markers` and `refuse-repeated-marker-occurrence-count` now pin jointly. This is the methodological contribution: the specification was refined by the act of independent implementation, and the vectors are the artifact of that refinement.

The byte-hash core (anchor, verify, coherence, corroborate, audit) depends only on SHA-256 and exact bytes—no corpus, no key, no clock. Marker-dependent output (refuse, the marker census) is versioned data: two implementations agree on a marker verdict only when pinned to the same corpus version, which each marker-dependent output echoes.

5.1 What the corpus covers

Capability	Vectors	Languages that pass
Core (byte-hash, exit split, JSON envelope, marker path, audit chain)	35	Python, Rust, Node.js, Go
Portable witness receipt (`emet check`)	5	Python, Rust, Node.js (Go: pending
Stripped-credential rebind (experimental)	4	Python (cross-language: follow-on)
Total	44	—

6 Threat model and honest limits

We model an adversary who controls some part of the path between a model (or its operators) and ground truth: the ability to alter artifacts, substitute a presented view for the source, inject false authority into context, tamper one read channel, or edit an accountability log. The adversary does *not* control the operator, the external verifier of record, or the mathematics of SHA-256. Under STRIDE, the load-bearing class is elevation of privilege: an adversary who could make EMET assert authority, adjudicate a model decision, or run inside the audited system would convert the witness into an authority. The six boundaries are designed to preclude this by construction—EMET's output vocabulary contains no authority token and its verdict is a pure function of recomputed bytes—rather than by adversarial testing, which we name as an open deliverable. We also distinguish this from a separate concern the design does *not* resolve: a downstream integrator remains free to *treat* a MATCH as a permission, so the property established here is that EMET cannot *emit* authority, not that a consumer cannot misuse an integrity fact as one.

We state the residual surface plainly:

- **Denylist incompleteness.** Marker detection is a known-signature denylist; absence of a marker is not absence of injection.
- **Trust-root regress.** A compromised execution substrate re-derives a compromised self-hash consistently; `selftest` proves integrity only relative to an uncompromised substrate. An external verifier must be the check of record for EMET itself; EMET must not be its own root of trust.
- **Single-read-path collusion.** Corroboration detects divergence across disjoint reads; an adversary who tampers every read path identically is not caught at the byte layer.
- **Semantic blindness.** EMET judges bytes and provenance, never meaning; a semantically harmful artifact with intact bytes passes VERIFY by design.
- **Coerced operator.** The single actuator is assumed uncoerced; coercion is out of model.

7 Discussion: what agreement proves

The four implementations share an author. We are therefore explicit about the epistemic status of the conformance result. Same-author agreement across four languages demonstrates *internal consistency* (the specification does not contradict itself across its surfaces) and *implementability* (the specification can be realized in four runtimes from its text). It does *not* demonstrate independent re-derivability, which is a stronger claim: that a *different* author, reading only the specification and the vectors, can produce an implementation that passes them. That implementation is an open, named deliverable, and no party should treat re-derivability as proven until it exists.

This distinction is the spine of the artifact's integrity. A frozen contract plus production-grade reference implementations is a legitimate 1.0 deliverable; claiming the external re-derivability that only a different author can confer would be exactly the in-band overclaim that EMET exists to strip from other systems. We therefore frame the different-author implementation as a *falsifiable community challenge*: a single implementation from `SPEC.md` alone, in any language, passing `conformance/vectors.json`, converts re-derivability from an assertion into a demonstrated fact. The artifact is structured to make that challenge cheap to attempt and hard to game—a frozen specification, public vectors, a from-spec check command, and four public implementations to consult only after a genuine independent attempt.

EMET is a leaf witness, not a governance system. A governance stack that decides which artifacts are approved, current, and attributable [3] can invoke EMET for the byte-level integrity fact and retain the authority decision at its own layer; the two concerns are separable by design.

8 Reproducibility

The frozen specification (`SPEC.md` v1.0.0, normative, RFC 2119), the conformance vectors, the four implementations (Python, Rust, Node.js, Go), and the from-spec conformance runner (`conformance/run.py`) are public under MPL-2.0 at `https://github.com/HarperZ9/emet`. A reader can re-derive any core verdict from the specification and vectors alone, and can check any implementation against the corpus without reading reference code.

Availability

EMET, its specification, conformance vectors, and the four implementations are at github.com/HarperZ9/emet (MPL-2.0).

References

[1] Yuan, A., Nian, Y., Zhang, H., Su, Z., and Zhao, Y. *SEVA: Self-Evolving Verification Agent with Process Reward for Fact Attribution.* arXiv:2606.29713, 2026.

[2] Faizan, M. and Alharthi, D. *ProvenAI: Provenance-Native Traces of Evidence in Generated Answers.* arXiv:2606.26449, 2026.

[3] Sulpovar, M., Konsynski, B. R., Kanchwala, Q., and Goodhart, G. *ContextNest: Verifiable Context Governance for Autonomous AI Agents.* arXiv:2607.02116, 2026.

[4] Wang, J., Li, Z., Wang, Z., Shen, X., and Zhang, F. *Minos: A Multi-Agent Collaborative Framework for Provenance-Based Backward Tracking.* arXiv:2607.00440, 2026.

[5] Chen, B. *Making Failure Safe: A Constrained, Verifiable Agent Framework for Open-Web Data Collection.* arXiv:2607.00035, 2026.

[6] Bradner, S. *Key words for use in RFCs to indicate requirement levels.* RFC 2119, IETF, 1997.

[7] NIST. *Secure Hash Standard (SHS)*. FIPS PUB 180-4, 2015.

[8] Reproducible Builds Project. *Reproducible Builds: a set of software development practices.* https://reproducible-builds.org, 2026.