Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 11 October 2026

G. G

9 April 2026

COGITATOR Witness Protocol: Cryptographically Verifiable Audit Records for AI Agent Execution

draft-noctem-cogitator-witness-protocol-00

Abstract

This document specifies the COGITATOR Witness Protocol -- a scheme for producing cryptographically verifiable, tamper-evident records of AI agent execution. A conforming implementation produces a single witness root value that any third party can independently recompute from the same inputs to verify the record was not altered after the fact.

The protocol is implementation-agnostic. The reference implementation is COGITATOR (Rust), but any language or framework can produce conforming witness bundles.

This document also describes how the COGITATOR Witness Protocol relates to the IETF SCITT architecture, with which it is designed to be used as a payload inside SCITT Signed Statements.

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 11 October 2026.

Copyright Notice

Table of Contents

1. Introduction

AI agents deployed in regulated or high-stakes environments issue tool calls with real-world consequences. Existing audit approaches -- log ingestion, post-hoc summaries, model cards -- are reconstructive. They describe what probably happened, not what provably happened. Logs are mutable. Post-hoc summaries are interpretations.

The COGITATOR Witness Protocol takes the position that agent execution should be as auditable as a compiled binary in a reproducible build system. The witness root is the runtime equivalent of a content-addressed store path: a cryptographic commitment that ties a specific output to a specific, verifiable execution.

This specification is intended to complement, not replace, the SCITT architecture (draft-ietf-scitt-architecture). SCITT provides the outer envelope: a Signed Statement registered on a Transparency Service with a verifiable Receipt. The COGITATOR Witness Protocol provides the inner payload: a structured, hash-chained record of what the agent did, attempted, and was blocked from doing.

1.1. Requirements 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 BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

2. Definitions

Run A single end-to-end execution of an agent against a set of inputs.

Tool call A discrete action dispatched by the agent to an external or internal tool.

Phantom entry A record of a tool call that was intercepted and blocked before dispatch.

Witness root A single BLAKE3 hex digest committing the entire run record.

Policy digest A SHA-256 hex digest of the policy file in effect during the run.

Canonical JSON JSON serialised according to RFC 8785 (deterministic key ordering, no insignificant whitespace).

Witness bundle The complete set of artefact files for a single run.

Signed Statement As defined in draft-ietf-scitt-architecture: a COSE-signed envelope carrying a payload and subject claim, registered on a SCITT Transparency Service.

Receipt As defined in draft-ietf-scitt-architecture: a proof of inclusion issued by a Transparency Service confirming a Signed Statement was registered on its ledger.

3. Witness Bundle Structure

A conforming witness bundle MUST contain the following files:

```
run_<id>/
+-- agent_trace.json        # Agent steps: inputs, tool requests, outputs
+-- tool_transcript.json    # All tool calls (real and phantom) with outcomes
+-- chaos_profile.json      # Fault injection schedule (may be empty)
+-- drift_report.json       # Replay mismatch report (empty if no drift)
+-- hash_chain.txt          # Per-call BLAKE3 hashes, one per line
+-- meta.json               # Witnessed metadata
+-- witness_manifest.json   # Per-file hashes and bundle hash
+-- witness_root.txt        # Single hex string -- the tamper-evident root
```

All JSON files MUST be serialised as RFC 8785 canonical JSON before hashing.

4. Schema Definitions

4.1. meta.json

```
{
  "schema_version": 4,
  "run_id": "<string>",
  "agent_id": "<string>",
  "seed": "<uint64>",
  "policy_digest": "<sha256-hex | null>",
  "started_at": "<ISO 8601 datetime>",
  "finished_at": "<ISO 8601 datetime>",
  "cogitator_version": "<semver string>"
}
```

* schema_version MUST be 4 for this version of the protocol.

* policy_digest MUST be the SHA-256 hex digest of the policy file bytes, or null if no policy was in effect.

* seed MUST be the fixed random seed used for the run, enabling deterministic replay.

4.2. tool_transcript.json

```
{
  "schema_version": 4,
  "entries": [ /* ToolCall[] */ ],
  "phantom_entries": [ /* PhantomEntry[] */ ],
  "policy_digest": "<sha256-hex | null>"
}
```

4.2.1. ToolCall Object

```
{
  "step": "<uint>",
  "tool_call_idx": "<uint>",
  "tool_name": "<string>",
  "request": { },
  "response": { },
  "chaos_fault": "<string | null>",
  "call_hash": "<blake3-hex>"
}
```

call_hash MUST be the BLAKE3 digest of the RFC 8785 canonical JSON of this object, with call_hash set to the empty string before hashing.

4.2.2. PhantomEntry Object

```
{
  "step": "<uint>",
  "tool_call_idx": "<uint>",
  "tool_name": "<string>",
  "request": { },
  "disposition": "Blocked | Phantom",
  "rule_id": "<string | null>",
  "reason": "<string | null>",
  "entry_hash": "<blake3-hex>"
}
```

* entry_hash MUST be the BLAKE3 digest of the RFC 8785 canonical JSON of this object, with entry_hash set to the empty string before hashing.

* disposition MUST be one of Blocked (tool call explicitly denied by policy) or Phantom (tool call silently observed but not dispatched).

4.3. hash_chain.txt

A newline-delimited text file. Each line is the call_hash or entry_hash of one record in the order they were produced, interleaving real calls and phantom entries chronologically.

4.4. witness_manifest.json

```
{
  "files": {
    "agent_trace.json": "<blake3-hex>",
    "tool_transcript.json": "<blake3-hex>",
    "chaos_profile.json": "<blake3-hex>",
    "drift_report.json": "<blake3-hex>",
    "hash_chain.txt": "<blake3-hex>",
    "meta.json": "<blake3-hex>"
  },
  "bundle_hash": "<blake3-hex>"
}
```

* Each file hash is the BLAKE3 digest of the raw file bytes.

* bundle_hash is the BLAKE3 digest of the RFC 8785 canonical JSON of the files object.

4.5. witness_root.txt

A single line containing the BLAKE3 hex digest of the RFC 8785 canonical JSON of the complete witness_manifest.json object (including bundle_hash). This is the only value that needs to be published for a third party to verify the entire bundle.

5. Witness Root Computation

The witness root is computed as follows:

1. Serialise each bundle file to RFC 8785 canonical JSON (or raw bytes for text files).

2. Compute BLAKE3(file_bytes) for each file to produce the files map.

3. Compute BLAKE3(RFC8785(files)) to produce bundle_hash.

4. Serialise witness_manifest.json including bundle_hash.

5. Compute BLAKE3(RFC8785(witness_manifest)) to produce witness_root.

6. Write witness_root as a single lowercase hex string to witness_root.txt.

A verifier replays steps 1-6 from the bundle files and asserts the computed root matches the published witness_root.txt.

6. Policy Protocol

The policy layer is optional. If no policy file is provided, all tool calls are implicitly allowed and policy_digest MUST be null in both meta.json and tool_transcript.json.

When a policy file is provided:

1. The policy file bytes MUST be SHA-256 digested before any run begins.

2. The digest MUST be committed to meta.json and tool_transcript.json before the first tool call.

3. Every tool call MUST be evaluated against the policy before dispatch.

4. Blocked or phantomed calls MUST produce a PhantomEntry committed to the witness chain.

5. The CallHistory MUST be updated after every verdict, including blocked calls.

6.1. Policy Verdicts

Verdict	Tool dispatched?	Agent receives	Chain entry
allow	Yes	Real tool response	ToolCall
block	No	blocked: true	PhantomEntry(Blocked)
phantom	No	blocked: true	PhantomEntry(Phantom)

Table 1

7. Deterministic Replay

A conforming implementation MUST support replay mode. Given the original agent_trace.json, chaos_profile.json, policy file (identified by policy_digest), and seed, a replay run MUST produce an identical witness_root to the original run. Any deviation MUST be reported as a DriftIssue in drift_report.json.

8. Conformance

An implementation is conforming if:

1. It produces all required bundle files.

2. All JSON files are RFC 8785 canonical before hashing.

3. The witness root computation follows the algorithm in Section 6 exactly.

4. Phantom entries are produced for all blocked and phantomed calls.

5. The policy digest is committed before the first tool call when a policy is in effect.

6. Replay of the same inputs produces an identical witness root.

9. Integration with SCITT

The COGITATOR Witness Protocol is designed to be used as a payload within the SCITT architecture (draft-ietf-scitt-architecture). The RECOMMENDED integration pattern is:

1. After a run completes, the witness bundle is produced as specified in this document.

2. The witness_root value and run_id from meta.json are embedded as the payload of a COSE-signed SCITT Signed Statement, with the subject header set to the agent identifier.

3. The Signed Statement is registered with a SCITT Transparency Service via the SCRAPI interface (draft-ietf-scitt-scrapi).

4. The resulting Receipt is stored alongside the witness bundle.

This provides two independently verifiable guarantees: the SCITT Receipt proves the witness root was registered at a specific time; the witness root proves the bundle contents have not been altered since computation.

10. Relation to Regulatory Frameworks

The COGITATOR Witness Protocol is designed to address requirements of:

* EU AI Act (2024) Articles 12 and 9 -- tamper-evident record-keeping and risk management for high-risk AI.

* FCA AI and machine learning guidance -- audit trails for automated decision systems in financial services.

* NIST AI RMF (2023) -- traceability and accountability for AI systems.

11. Versioning

This document describes protocol version 1.0, corresponding to schema_version 4 in bundle files. Breaking changes will increment both version numbers together.

12. IANA Considerations

This document has no IANA actions.

13. Security Considerations

The integrity guarantees of the witness root depend on the collision resistance of BLAKE3. As of the date of this document, BLAKE3 is not known to have practical collision attacks.

The protocol does not provide confidentiality. Witness bundles may contain sensitive agent inputs and outputs. Implementors are responsible for access controls on bundle storage and transmission.

The policy digest commits to the policy file bytes but does not authenticate the provenance of the policy file itself. Deployments requiring policy provenance guarantees should sign policy files independently before committing the digest.

14. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

15. Informative References

[I-D.draft-ietf-scitt-architecture]
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-22, October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-22>.

[I-D.draft-ietf-scitt-scrapi]
Birkholz, H. and J. Geater, "SCITT Reference APIs", Work in Progress, Internet-Draft, draft-ietf-scitt-scrapi-07, March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-scrapi-07>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

Author’s Address

George G
Email: george.g@tuta.io