Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2026

J. Dembowski
20 April 2026

Proof-of-Behavior Protocol for Autonomous AI Agents
draft-dembowski-agentledger-proof-of-behavior-00

Abstract

Autonomous AI agents execute actions — file writes, API calls, shell commands — on behalf of human principals. No standard mechanism exists for a third party to verify that an agent acted within its declared behavioral rules, that policy enforcement occurred before execution (not after), or that the action log has not been tampered with after the fact.

This document defines the Proof-of-Behavior (PoB) protocol: a minimal, language-agnostic standard for tamper-evident audit trails and pre-execution policy enforcement in AI agent systems. The protocol specifies a signed receipt format, a hash-chain linking scheme, a policy gate contract, and a cross-agent reference mechanism.

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 22 October 2026.

Copyright Notice

Table of Contents

1. Introduction

AI agent frameworks (LangChain, AutoGen, CrewAI, and others) expose lifecycle hooks that allow external code to observe agent actions. These hooks are sufficient for logging but do not provide cryptographic guarantees. A log written by the agent process being audited is not independently verifiable: the same software that produced the actions also produces the evidence.

The Proof-of-Behavior protocol addresses three distinct problems:

1. *Tamper evidence*: Once an action is recorded, neither the agent nor its operator can modify or delete the record without detection.

2. *Pre-execution enforcement*: Policy decisions must be recorded before the action executes. A DENIED record proves the agent was blocked; the absence of such a record proves the policy was not consulted.

3. *Cross-agent attribution*: In multi-agent systems, an orchestrator should be able to verify that a downstream agent completed a specific action under a specific policy, without trusting the downstream agent's in-process state.

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 [RFC2119] [RFC8174].

2. Terminology

Receipt: A signed JSON object recording a single agent action. Immutable once finalized.

Chain: An ordered, append-only sequence of receipts, each cryptographically linked to its predecessor.

Policy Gate: A pre-execution evaluation step that produces either an ALLOW or DENY verdict before any action is forwarded to the agent.

DENIED Receipt: A receipt with status "denied", written to the chain before the action is attempted. Proves the policy gate ran and rejected the action.

agent_id: The raw bytes of the agent's Ed25519 public key, encoded as lowercase hexadecimal. Serves as both identity and verification key.

principal_id: An opaque string identifying the human or organizational principal that authorized this agent's keypair. Format is binding-specific.

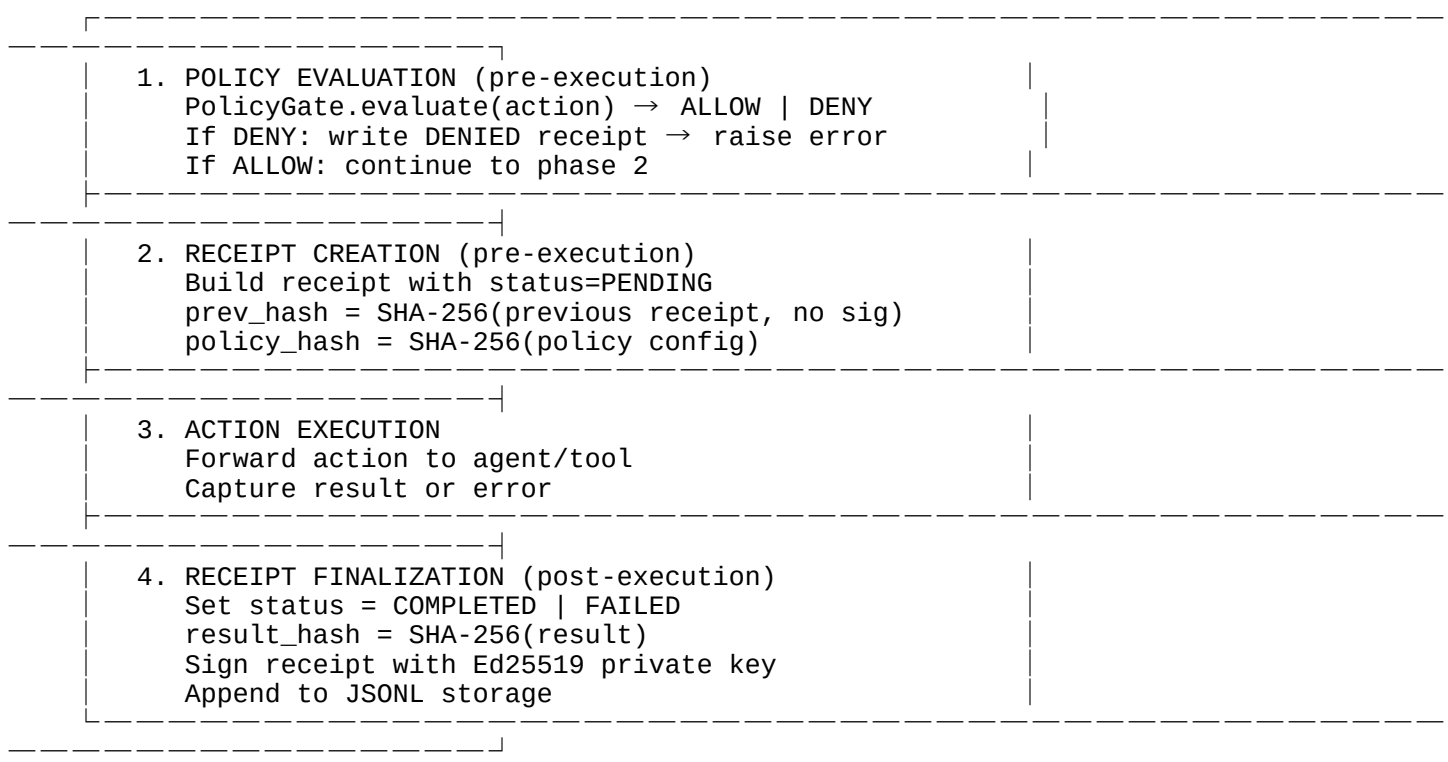
policy_hash: SHA-256 hash of the policy configuration, embedded in

the signed receipt payload. Ensures policy cannot be substituted without breaking signatures.

prev_hash: SHA-256 hash of the preceding receipt's canonical form (signature field excluded). Null for the first receipt in a chain (genesis receipt).

3. Protocol Overview

The protocol operates in four phases for each agent action:



The critical property of phase 1: the DENIED receipt is written to persistent storage and signed before the error is raised. The agent process cannot proceed to phase 3 without passing phase 1, and cannot suppress the phase 1 record.

4. Receipt Format

A receipt is a JSON object [RFC8259] with the following top-level fields.

4.1. Required Fields

receipt_id (string): A UUIDv4 string uniquely identifying this receipt.

chain_id (string): Equal to agent_id. Identifies the chain this receipt belongs to.

agent_id (string): Lowercase hex encoding of the agent's Ed25519 raw public key (64 hex characters, 32 bytes).

principal_id (string): Opaque identifier for the authorizing principal. Format is defined by the binding in use.

timestamp (string): ISO 8601 UTC timestamp of receipt creation (e.g., "2026-04-20T10:00:00.000000+00:00").

prev_hash (string or null): SHA-256 hex digest of the canonical form of the preceding receipt (signature excluded). MUST be null for the first receipt in a chain.

schema_version (string): Protocol version. This document defines version "0.1".

action (object): Describes the agent action. See Section 4.2.

signature (string): Ed25519 signature over the canonical form of the receipt (signature field excluded), encoded as lowercase hex (128 hex characters, 64 bytes).

4.2. Action Object

type (string): One of: "tool_call", "llm_invoke", "decision", "cross_agent".

framework (string): The agent framework in use. RECOMMENDED values: "langchain", "autogen", "crewai", "custom".

tool_name (string or null): Name of the tool invoked. REQUIRED when type is "tool_call".

status (string): One of: "pending", "completed", "failed", "denied".

payload_hash (string or null): SHA-256 hex digest of the canonical JSON encoding of the action input. MAY be null.

result_hash (string or null): SHA-256 hex digest of the canonical JSON encoding of the action result. MUST be null when status is "pending" or "denied".

error (string or null): Human-readable error message. SHOULD be set when status is "failed" or "denied".

policy_hash (string or null): SHA-256 hex digest of the policy configuration active at the time of evaluation. MUST be set when a policy gate is in use. Embedded in the signed payload.

4.3. Example Receipt

```
{
  "action": {
    "error": null,
    "framework": "langchain",
    "payload_hash": "e3b0c44298fc1c149afbf4c8996fb924...",
    "policy_hash": "3fa7188e44ea5c896b2a5d1068de1246...",
    "result_hash": "9f86d081884c7d659a2feaa0c55ad015...",
    "status": "completed",
    "tool_name": "web_search",
    "type": "tool_call"
  },
  "agent_id": "6c24573e8a4f2b9c...",
  "chain_id": "6c24573e8a4f2b9c...",
  "cross_agent_ref": null,
  "prev_hash": "a3f4b2c1d0e9...",
  "principal_id": "agent@example.com",
  "receipt_id": "550e8400-e29b-41d4-a716-446655440000",
  "schema_version": "0.1",
  "signature": "4a5b6c7d...",
  "timestamp": "2026-04-20T10:00:00.000000+00:00"
}
```

5. Canonicalization

The canonical form of a receipt is used for both signing and hash computation. It MUST be produced as follows:

1. Serialize the receipt as JSON.

2. Exclude the "signature" field entirely.

3. Sort all object keys lexicographically at every nesting level (recursive).

4. Emit with no insignificant whitespace (separators "," and ":").

5. Encode as UTF-8 bytes.

This is equivalent to JCS (JSON Canonicalization Scheme) with recursive key sorting.

Implementations MUST use this canonical form consistently. Any deviation produces a different byte sequence and an invalid signature.

6. Hash Chain

Each receipt's prev_hash links it to its predecessor:

prev_hash[n] = SHA-256(canonical_form(receipt[n-1]))

where canonical_form excludes the signature field.

The genesis receipt (first in chain) MUST have prev_hash = null.

Verification of the chain proceeds from genesis to tip:

1. Assert receipt[0].prev_hash == null.
2. For i = 1..N: assert receipt[i].prev_hash == SHA-256(canonical_form(receipt[i-1])).
3. For each receipt: verify Ed25519 signature over canonical_form(receipt[i]).

A verifier that observes a valid chain at time T can store the (receipt_id, hash) pair and later detect any rewrite of the chain history, even if the verifier has no access to the signing key.

7. Signing

Each receipt MUST be signed using Ed25519 [RFC8032] as follows:

1. Produce the canonical form of the receipt (signature field excluded).
2. Sign the UTF-8 bytes using the agent's Ed25519 private key.
3. Encode the 64-byte signature as lowercase hexadecimal.
4. Set receipt.signature to this value.

The verifying party uses the agent_id field (which encodes the Ed25519 public key) to verify each signature independently.

Agent identity MUST be established before any receipt is written. The Ed25519 keypair MUST be generated fresh for each agent instance unless a persistence mechanism is in use (see Section 12).

8. Policy Gate

The policy gate is an external component that evaluates each action before execution. It operates as follows:

```
result = policy.evaluate(action_type, tool_name, payload)

if result.verdict == DENY:
    write_denied_receipt(action_type, tool_name, result.reason, policy_hash)
    raise PolicyViolationError
else:
    # proceed to execution
```

The DENIED receipt MUST be written to persistent storage before PolicyViolationError is raised. Implementations MUST NOT proceed to execution if the DENIED receipt write fails.

The policy_hash field in every receipt (ALLOW and DENY) MUST contain the SHA-256 hash of the policy configuration. This ensures that an auditor can verify not only that the policy ran, but that the specific policy configuration active at the time matches any externally stored policy definition.

The external observer pattern is RECOMMENDED: the policy gate and receipt chain SHOULD be constructed outside the agent process and injected via callback registration. This places the enforcement logic in a different trust domain from the agent being audited.

9. Cross-Agent References

When agents collaborate, handoffs SHOULD be recorded as cross-agent receipts. A cross-agent receipt references a specific receipt in another agent’s chain.

9.1. Cross-Agent Reference Object

```
{
  "target_agent_id": "273170e7...",
  "ref_receipt_id": "550e8400-...",
  "status": "pending"
}
```

target_agent_id: The agent_id (Ed25519 public key hex) of the referenced agent.

ref_receipt_id: The receipt_id of the specific receipt being referenced.

status: One of: "pending", "confirmed".

9.2. Resolution

To resolve a cross-agent reference, the verifying agent:

1. Locates the referenced agent's chain (by target_agent_id).

2. Finds the receipt with the matching receipt_id.

3. Verifies the Ed25519 signature on that receipt using target_agent_id as the public key.

4. Confirms the receipt status is "completed".

Signature verification in step 3 is REQUIRED. A cross-agent reference that resolves without signature verification provides no tamper-evidence guarantee for the referenced chain.

10. Checkpoint Hashes

For long-running chains, implementations MAY write periodic checkpoint records to enable efficient partial verification.

A checkpoint record is NOT a receipt. It MUST include:

* "checkpoint": true

* "at_receipt_id": receipt_id of the last receipt in the batch

* "receipt_count": number of receipts covered

* "cumulative_hash": SHA-256 of the concatenated canonical forms of all receipts from genesis through at_receipt_id

* "signature": Ed25519 signature over the checkpoint record (signature field excluded)

Partial verification from a checkpoint proceeds by:

1. Verifying the checkpoint's cumulative_hash against the covered receipts.

2. Verifying the chain tail (receipts after the checkpoint) independently.

11. Storage

Implementations SHOULD store receipts in newline-delimited JSON (JSONL) format: one JSON object per line, UTF-8 encoded, LF line endings.

Checkpoint records are interleaved with receipts in the same file. A parser MUST distinguish receipts from checkpoints by the presence of "checkpoint": true.

Storage MUST be append-only during normal operation. Implementations SHOULD use OS-level file locking or equivalent to prevent concurrent writes.

12. Key Persistence

Agent keypairs SHOULD be persisted across restarts to maintain a continuous chain. Implementations SHOULD:

1. Store the Ed25519 private key in a file with permissions 0400 (owner read-only).

2. Store the public identity (agent_id, principal_id, binding_signature) separately with permissions 0600.

3. Verify that the loaded keypair matches the chain’s agent_id before resuming.

A chain whose agent_id does not match the loaded keypair MUST NOT be extended.

13. Security Considerations

13.1. Chain Rewrite

An attacker with write access to the JSONL file can rewrite the entire chain history with freshly computed hashes and re-signed receipts, provided they control the signing key. Mitigations:

* External checkpointing: publish SHA-256(checkpoint) to an append-only external log at session boundaries. Any rewrite diverges from the published value.

* Bilateral co-signing: a separate signing service (in a different trust domain from the agent) co-signs each receipt. A rewrite requires compromising both keys.

* On-chain anchoring: commit checkpoint hashes to a public blockchain.

13.2. Key Compromise

A compromised private key allows an attacker to forge receipts that appear valid. Mitigations include key revocation registries and hardware security modules for key storage. These are out of scope for this document.

13.3. Same Trust Domain

The in-memory verify() operation uses the same keypair that signed the receipts. This proves internal consistency but does not protect against an attacker who has already replaced the in-memory chain state. Verifiers MUST use verify_from_disk() with the expected public key, not in-process verify().

13.4. Policy Bypass

The policy gate can be bypassed if the agent constructs a ReceiptChain instance without a policy, or calls finalize_last() directly without a preceding append(). Implementations SHOULD restrict access to the ReceiptChain constructor and enforce that policy is set at chain construction time.

14. IANA Considerations

This document has no IANA actions.

15. References

15.1. Normative References

15.2. Informative References

16. Normative References

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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/info/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>.

Appendix A. Receipt Schema Summary

The following table summarizes all receipt fields.

Field	Type	Required	Description
receipt_id	string (UUID)	REQUIRED	Unique receipt identifier
chain_id	string (hex)	REQUIRED	Equal to agent_id
agent_id	string (hex)	REQUIRED	Ed25519 public key, 64 hex chars
principal_id	string	REQUIRED	Authorizing principal identifier
timestamp	string (ISO 8601)	REQUIRED	UTC creation time
prev_hash	string (hex) or null	REQUIRED	SHA-256 of previous receipt
schema_version	string	REQUIRED	Protocol version ("0.1")
action.type	string (enum)	REQUIRED	Action type
action.framework	string	REQUIRED	Agent framework
action.tool_name	string or null	CONDITIONAL	Required for tool_call

action.status	string (enum)	REQUIRED	pending/completed/failed/ denied
action.payload_hash	string (hex) or null	OPTIONAL	SHA-256 of input
action.result_hash	string (hex) or null	OPTIONAL	SHA-256 of output
action.error	string or null	OPTIONAL	Error description
action.policy_hash	string (hex) or null	RECOMMENDED	SHA-256 of policy config
signature	string (hex)	REQUIRED	Ed25519 signature, 128 hex chars

Table 1

Appendix B. Reference Implementation

A reference implementation in Python (MIT licensed) is available at: https://github.com/dembovvski/agentledger

The implementation covers: AgentIdentityImpl (Ed25519), ReceiptChainImpl (JSONL storage, checkpoint hashes), DenylistPolicy/ AllowlistPolicy/ CompositePolicy, integrations for LangChain/AutoGen/ CrewAI, CLI verifier.

Author’s Address

Jakub Dembowski
Email: dembowski@proton.me