Zanii: Proof-of-Action for AI Agents

A transparency log for verifiable AI-agent behavior

Zanii · 2026 · v1

Abstract

Software agents increasingly act on behalf of people and organizations: they call tools, move money, and touch private data with little or no human in the loop. Yet the only record of what an agent did is a log its own operator can silently edit, so accountability rests on trusting whoever keeps the record. We propose a transparency log for agent actions. Each agent holds a cryptographic identity and receives scoped, expiring authority from its owner through a signed delegation certificate. Every action becomes an Ed25519-signed, hash-chained *receipt* that records only a hash of the payload, never the payload itself. Receipts are appended to a Merkle transparency log in the manner of Certificate Transparency; the log periodically signs a compact commitment to its entire history and writes that commitment to a public blockchain, so not even the log operator can rewrite the past. Anyone can then verify, offline and without trusting any party, what an agent did, under whose authority, and that the record was neither altered nor reordered. We call this primitive *proof-of-action*.

1. Introduction

Bitcoin showed that a ledger can be trustworthy without a trusted keeper [1], and Ethereum generalized that ledger into a platform for arbitrary state [2]. Both solve a problem about *value*: who owns what, and who may move it. The rise of autonomous software agents raises a different problem, about *behavior*: an agent invokes a payment API, sends an email as you, or reads a customer record, and afterward the only evidence is an application log. That log is mutable by the very party who would be blamed if the action were improper. When an auditor, a counterparty, or a court later asks "what did your agent actually do, and who authorized it?", the answer "trust our database" is not an answer.

The observation of this paper is that the machinery built to make the web's certificate authorities accountable, Certificate Transparency [3], solves exactly this shape of problem, and can be pointed at agent actions instead of certificates. A certificate authority can, in principle, issue a fraudulent certificate; CT makes every issued certificate appear in a public, append-only, cryptographically verifiable log, so misissuance is detectable. An agent operator can, in principle, misreport what its agents did; a transparency log of signed agent actions, anchored to a public blockchain, makes misreporting detectable in the same way.

We combine three well-understood components, none of them novel in isolation: digital signatures and scoped delegation for *authority*, a Merkle transparency log for *tamper-evident ordering*, and periodic on-chain anchoring for *global immutability*. The contribution is their composition into a single primitive for agent accountability, and a protocol precise enough that a receipt can be verified by anyone, with no call back to us.

2. Design goals

- **No trusted operator.** Verification must never require trusting Zanii, the log, or the agent's operator. The math is the trust.
- **Offline-verifiable.** A receipt plus a small proof must be checkable with no network access and no registry lookups.
- **Payload-private.** The log records only a hash of an action's payload; the underlying data never leaves the operator's systems.
- **Least authority, provable.** An agent must never be able to act beyond what its owner delegated, and that boundary must be checkable.
- **Cheap and portable.** Anchoring must cost a fraction of a cent, and an agent's entire history must export as one self-contained, independently verifiable file.

3. Identity

Every participant, owner, delegating team, agent, and the log itself, is an Ed25519 keypair. An identity is expressed as a `did:key` decentralized identifier:

```
did:key:z<base58btc(0xed 0x01 || publicKey32)>
```

The defining property is that the identifier *is* the public key: to verify a signature you decode the key directly from the DID, with no registry, no lookup, and no certificate authority. All hashing in the protocol is SHA-256, and all signatures are Ed25519, each computed over the RFC 8785 JSON Canonicalization Scheme (JCS) [4] form of the relevant object, UTF-8 encoded. Canonicalization matters: it guarantees that two systems serialize the same logical object to the same bytes, so a signature made by one verifies byte-for-byte on another.

4. Delegation

Money needs no notion of agency, you own a coin and you spend it. Agents do: an agent acts *for* a principal, and must not exceed what that principal allowed. Zanii represents authority as a **delegation certificate**, signed by the *issuer* over the JCS of the certificate without its signature:

```
{
  "v": 1,
  "issuer":  "did:key:z6Mk…",
  "subject": "did:key:z6Mk…",
  "scopes":  ["stripe.*", "gmail.send"],
  "exp":     "2026-08-01T00:00:00.000Z",
  "sig":     "ed25519:…"
}
```

A **scope** is an exact string (`gmail.send`) or a wildcard prefix (`stripe.*`, where `a.*` covers `a` , `a.b` , and `a.b.c`). Certificates compose into a **delegation chain**, root first, e.g. organization → team → agent. A

chain is *valid* for agent `A` at time `t` against an expected owner `O` if and only if:

1. every certificate's signature verifies against its `issuer` key;
2. `chain[0].issuer = O` (when the verifier pins an owner);
3. `chain[i+1].issuer = chain[i].subject` for all links;
4. `chain[last].subject = A`;
5. every certificate satisfies `t < exp`;
6. no certificate's hash is in the verifier's revocation set (§9).

The chain's **effective scopes** are the last certificate's scopes, intersected with every ancestor's, so a delegate can only *narrow* authority, never widen it. The result is a provable permission slip that travels with every action and always resolves back to a human principal.

5. Action receipts

The atomic unit is the **receipt**: one signed record per action. It is analogous to a Bitcoin transaction, but instead of transferring value it attests behavior.

```
{
  "v": 1,
  "agent_id":     "did:key:z6Mk…",
  "delegation":   [ /* cert chain, root first */ ],
  "action":       "tool_call",
  "target":       "stripe.charges.create",
  "payload_hash": "sha256:…",
  "ts":           "2026-07-02T16:00:00.000Z",
  "prev":         "sha256:… | null",
  "sig":          "ed25519:…"
}
```

The signature is Ed25519 by the agent's own key (the key inside `agent_id`) over the JCS of the receipt without `sig`. Two design choices carry most of the weight. First, `payload_hash` is a hash of the action's data, never the data, so the log and its proofs leak nothing: the email body, the customer record, the amount, all stay with the operator, while the receipt still binds to that exact payload. Second, `prev` is the hash of the agent's previous receipt (or `null` for the first), forming a per-agent hash chain: removing, reordering, or altering any receipt breaks the linkage of every receipt after it.

6. Receipt verification

A receipt `R` verifies at time `t` if and only if:

1. `R.sig` verifies against the public key inside `R.agent_id`;
2. `R.delegation` is a valid chain for `R.agent_id` at `R.ts` (§4);
3. `R.target` is covered by the chain's effective scopes;

4. in a sequence, `R.prev` equals the hash of the preceding receipt.

This check is *pure*: no network, no trusted party, no clock beyond the timestamps in the objects themselves. It establishes that a specific, authorized agent attested a specific action. What it does not yet establish is that the receipt is part of an immutable, ordered history; that is the job of the log.

7. The transparency log

Receipts are appended to an **append-only Merkle tree**, following the construction of Certificate Transparency (RFC 6962 [3], RFC 9162 [5]). The Merkle Tree Hash (MTH) of a list of leaves `D` is defined with explicit domain separation between leaves and interior nodes:

```
MTH({})       = SHA-256()
MTH({d(0)})   = SHA-256(0x00 || d(0))
MTH(D[n])     = SHA-256(0x01 || MTH(D[0:k]) || MTH(D[k:n])),  n > 1
```

where a leaf's data is the JCS bytes of the full receipt (including its signature), and `k` is the largest power of two strictly less than `n`. The `0x00` prefix on leaves and `0x01` on nodes is not decorative: it gives second-preimage resistance, preventing an attacker from passing an interior node off as a leaf or vice versa.

Periodically the log publishes a **Signed Tree Head** (STH), a compact, signed commitment to its entire history at a given size:

```
{ "v": 1, "log_id": "did:key:z6Mk…", "size": 12345,
  "root": "sha256:…", "ts": "…", "sig": "ed25519:…" }
```

The STH's signature is Ed25519 by the log's key over the JCS of the STH without `sig`; the log's identity is itself a `did:key`. The root is a single 32-byte hash that fixes the exact content and order of every receipt in the tree: change one bit of one receipt and the root changes.

8. Inclusion and consistency proofs

The Merkle structure buys two proofs, each logarithmic in the size of the log.

An **inclusion proof** for a receipt at leaf index `i` in a tree of size `n` is the *audit path*: the `O(log n)` sibling hashes along the path from that leaf to the root. A verifier recomputes candidate roots by folding the leaf hash with each sibling and checks that the result equals the root in a signed STH. For a log of one million receipts, that is roughly twenty hashes, not a million. This is how a receipt is proven to be *in* the log without downloading the log.

A **consistency proof** between an earlier size `m` and a later size `n` is a set of at most `⌈log2 n⌉ + 1` hashes demonstrating that the size-`n` tree is an append-only *extension* of the size-`m` tree, that the earlier history was preserved intact and only grew. Together, inclusion and consistency proofs make deletion and reordering mathematically detectable: any such tampering yields a root that no longer matches the signed, and later anchored, tree head.

A receipt is **fully proven** when it verifies per §6, its leaf has a valid inclusion proof against an STH, and that STH's signature verifies against the log's DID.

9. On-chain anchoring

Signatures make the log's claims *attributable*, but not *singular*. A dishonest log could sign two different histories and show each to a different party, a "split-view" attack, since it holds the key and can sign both. To remove this, the log periodically writes its current STH outside its own control, onto a public blockchain.

An **anchor** is an EIP-1559 transaction on a public EVM chain whose calldata is the JCS bytes of the STH; the anchor record stores the transaction hash and block number as its reference. Once anchored, the root is a single value visible to everyone on a ledger the log does not control, and changing it would require rewriting the blockchain itself. A receipt at leaf index `i` is **anchored** once any anchor exists with `size > i`; consistency proofs then tie every later tree back to the anchored one. Third-party verification is direct: fetch the transaction's calldata from the chain, decode the STH, verify its signature, and demand an inclusion proof against it.

This is the same defense Bitcoin uses, made cheap. Bitcoin makes history expensive to rewrite through proof-of-work; Zanii does not run a chain of its own, it *rents* an existing chain's immutability by committing a 32-byte root for a fraction of a cent per anchor. An "average" verifier need not understand any of this: they open the anchor transaction on a public block explorer, a site the operator does not run, and see the log's fingerprint recorded there, permanently, at a known time.

> *Deployment note.* The protocol is chain-agnostic; any public EVM chain suffices, with a mainnet the production target so that rewriting history carries real economic cost. A `file` backend (a local append-only file) exists for development and is explicitly *not* tamper-proof.

10. Revocation

Authority must be withdrawable, and the *time* of withdrawal must itself be provable. A **revocation record** embeds the full delegation certificate being revoked and is signed by that certificate's *issuer*, the only party that could have granted the authority in the first place:

```
{ "v": 1, "type": "revocation", "cert": { /* full cert */ },
  "ts": "…", "sig": "ed25519:…" }
```

The log appends valid revocation records as their own Merkle leaves and adds the certificate's hash to its revoked set. Thereafter any delegation chain containing the revoked certificate is invalid (§4, rule 6): the log rejects new receipts under it, and verification of existing ones flags it. Key rotation is not a separate mechanism but a composition: issue a new certificate to the new key, then revoke the old one.

11. Cross-organization receipts

When an agent of organization X interacts with an agent of organization Y, neither organization's word should be the record. An **A2A receipt** is a single statement signed by *both* parties over an identical signing base (the receipt with both signature fields removed), so the two signatures are order-independent and neither countersigns the other. The receipt's hash enters *both* agents' hash chains, and the log advances both heads atomically on ingest. The result is one neutral proof of a bilateral interaction that both sides, and any third party, can verify, aligned with the emerging ERC-8004 model for trustless agent identity [6].

12. Reputation and payments

Reputation in Zanii is *evidence*, not assertion. A reputation view over an agent reports only its verifiable history, receipt count, activity span, cross-party interaction count, distinct counterparties, revocation status, and anchored coverage, and every figure resolves to receipts anyone can re-verify from the agent's exported audit bundle. Where a numeric rating is wanted, it is posted through the standard ERC-8004 reputation registry with a link back to that verifiable bundle, so the score is *backed by* history rather than claimed.

Payments are a **convention, not a rail**: a receipt with `action: "payment"` and a target such as `payment.x402` or `payment.ap2`, whose payload carries the payment rail's own settlement reference. Zanii proves *who initiated and received* a payment; the rail settles the money. This lets agent-to-agent commerce inherit the same tamper-evident, anchored accountability as every other action.

13. Security analysis

We consider four adversaries: a malicious agent, a malicious operator (including us), a network attacker, and a colluding log.

Forging a receipt. To produce a receipt that verifies as some agent `A`, an adversary must produce a valid Ed25519 signature under `A`'s key. Ed25519 provides roughly 128-bit security, so absent the private key the probability of forging a single signature is on the order of `2^−128 ≈ 3 × 10^−39`. Delegation bounds the blast radius of a stolen key: a compromised agent can still act only within its unexpired, unrevoked scopes, and revocation plus short expiries cap the exposure window.

Altering logged history. Changing, deleting, or reordering any receipt changes the leaf or node hashes above it and therefore the Merkle root. Because the root is signed by the log (forging that signature is again `~2^−128`) and, once anchored, fixed on a public chain, undetected tampering of anchored history reduces to rewriting the blockchain, whose cost is the chain's own security budget.

Split view. A colluding log could sign two histories, but the moment either is checked against the single on-chain anchor, the fork is exposed: at most one root can match the anchored value. Continuous auditors (monitors that fetch each new STH and demand a consistency proof to the last) shrink the detection window to the anchoring interval. Thus the log's remaining power is limited to *delaying* an entry or *withholding* a proof, both of which are visible as a failure to produce a valid proof, never to silently rewrite the past.

Privacy of the analysis. None of the above requires exposing payloads, because verification operates entirely on hashes, signatures, and proofs.

14. Privacy

The log stores, per action, only `payload_hash`, the action's `target`, timestamps, identities, and signatures. The action's data never leaves the operator, yet the receipt still binds to it: to later prove what happened, the operator discloses the payload and anyone recomputes its hash and matches it against the anchored receipt. Identities are `did:key` public keys, which are pseudonymous unless independently linked to a real-world entity. Selective disclosure is natural: an audit bundle can be scoped to a single agent, and a payload revealed only to those who need it.

15. Related work

Certificate Transparency [3, 5] pioneered append-only Merkle logs with public auditing for TLS certificates; Zanii adopts its tree construction and proof algorithms and applies them to agent actions, adding identity, scoped delegation, and on-chain anchoring. **Sigstore/Rekor** [7] applies transparency logging to software artifacts. The **Ethereum Attestation Service** provides on-chain attestations but places data on-chain; Zanii keeps payloads off-chain and anchors only a root. **ERC-8004** [6] defines trustless-agent identity and reputation registries, with which Zanii's identities and reputation feedback align. Foundational to all of these are Merkle's hash trees [8] and the Bitcoin [1] and Ethereum [2] ledgers, whose core insight, a record made trustworthy by structure rather than by a trusted keeper, this work carries into agent behavior.

16. Conclusion

We have described proof-of-action: a way to make what AI agents do provable to anyone, without trusting the parties involved. An agent's authority is a signed, scoped, expiring delegation that resolves to a human principal; each action is a signed, hash-chained receipt that reveals only a hash of its data; receipts live in an append-only Merkle log whose signed head is anchored to a public blockchain; and every claim, who did what, under whose authority, in what order, unaltered, is checkable offline by anyone. None of the parts are new. Their composition gives AI agents something they have lacked and increasingly need: a record they cannot fake, and that no one has to be trusted to keep.

References

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. [2] V. Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2014. [3] B. Laurie, A. Langley, E. Kasper. *Certificate Transparency*. RFC 6962, 2013. [4] A. Rundgren, B. Jordan, S. Erdtman. *JSON Canonicalization Scheme (JCS)*. RFC 8785, 2020. [5] B. Laurie, E. Messeri, R. Stradling. *Certificate Transparency Version 2.0*. RFC 9162, 2021. [6] *ERC-8004: Trustless Agents*. Ethereum ERC draft, 2025. [7] Sigstore. *Rekor: Software Supply Chain Transparency Log*. 2021. [8] R. C. Merkle. *Protocols for Public Key Cryptosystems*. IEEE S&P, 1980.