
Attested Multi-Agent Conversation Logs: A Tamper-Evident Black Box for AI Governance¹

Anantha Shakthi Ganeshan Thevar
Psiog

Publius Dirac
BlueQubit

With
Apart Research

Abstract

As multi-agent AI systems assume high-stakes responsibilities, governance frameworks such as the EU AI Act demand traceable, auditable records of agent interactions. We introduce Attested Logs, an open-source Python library that serves as a tamper-evident "black box" for AI conversations. Each message is cryptographically signed (Ed25519), hash-chained (SHA-256), and anchored to trusted public keys, enabling fully offline verification of integrity, authenticity, and ordering. Inspired by aviation flight recorders and C2PA content provenance, the library supports incident forensics, regulatory compliance, and cross-organizational trust. Integrations for AutoGen (manual logging) and LangGraph (callback-based auditing) are provided, together with runnable demos demonstrating end-to-end signing, verification, and tamper detection in real LLM conversations, and 20+ passing tests covering core, crypto, chain, and verification layers.

¹ Research conducted at the [Technical AI Governance Challenge](#), 2026

1. Introduction

As multi-agent AI systems take on high-stakes tasks, the lack of trustworthy records of agent interactions creates serious governance gaps. Post-incident reconstruction often relies on incomplete logs or centralized data that may be altered or inaccessible.

Attested Logs is a lightweight Python library that brings cryptographically attested, tamper-evident logging to multi-agent conversations. Every message is signed (Ed25519), hash-chained (SHA-256), and anchored to organizational certificate authorities (similar to HTTPS CA's), enabling fully offline verification of integrity, authenticity, and ordering.

Our main contributions:

1. A lightweight, offline-verifiable ledger library forming tamper-evident chains in multi-agent conversations using Ed25519 signatures and SHA-256 linking.
2. Framework integrations (AutoGen, LangGraph) demonstrating how attested logging can be retrofitted into existing orchestration tools.
3. Runnable demos showing end-to-end signing, verification, and tamper detection, together with 20+ passing tests covering core, crypto, chain, and verification layers.

By augmenting ECMA-430 (NLIP), which standardizes multimodal agent messaging (text, structured data, binary), we embed provenance directly into conversation flow, producing self-contained audit trails for regulatory compliance (e.g., EU AI Act), incident forensics, and cross-organizational trust.

2. Related Work

This section reviews the most relevant prior work in tamper-evident logging, content provenance, AI agent communication standards, and auditability for multi-agent systems. We highlight key projects, standards, and tools, explain their contributions and limitations, and clarify the specific gaps that **Attested Logs** addresses.

2.1 Tamper-Evident Logging for AI Actions

The closest direct inspiration is the **AI Action Ledger** (GitHub: Jreamr/ai-action-ledger), an open-source tool for creating tamper-evident audit logs of AI agent actions. It records LLM calls,

tool uses, and chain steps in an append-only log using SHA-256 hash chaining to detect tampering, with a dashboard for visualization and integrity verification.

- **Strengths:** Simple, lightweight, AI-specific (e.g., LangChain/LangGraph compatibility), fully open-source.
- **Limitations:** Stores only metadata hashes (not full message content), lacks per-message cryptographic signatures from individual agents, no certificate-based trust anchoring, and no embedded proofs for offline verification of authenticity/non-repudiation.

Attested Logs builds directly on this concept but extends it to full-message signing with Ed25519 and offline verification — making it suitable for governance-grade incident forensics.

2.2 Content Provenance and Signed Metadata Standards

Content Credentials (C2PA) — developed by the Coalition for Content Provenance and Authenticity — is a widely adopted standard for embedding cryptographically signed metadata into media assets (images, video, audio) to record creation history, edits, and AI generation provenance. It uses manifests with assertions, digital signatures, and trust anchors to enable verification of origin and authenticity, even offline.

- **Relevance:** Directly analogous to our goal — applying provenance to AI-generated outputs (conversation logs instead of media). C2PA demonstrates real-world success of signed, tamper-evident records for trust in generated content.
- **Differences:** C2PA focuses on static assets (single files) rather than dynamic, multi-turn conversations. It does not address multi-agent signing chains or real-time append-only logging.

We adapt C2PA-style embedded signatures and trust models to sequential message logs.

2.3 AI Agent Communication Standards: ECMA-430 (NLIP)

The **Natural Language Interaction Protocol (NLIP)**, standardized as ECMA-430 (core message format) and related specifications (ECMA-431 HTTP binding, ECMA-434 security profiles, etc.), published by Ecma International in December 2025, defines a vendor-neutral, multimodal envelope protocol for human-agent and agent-to-agent communication. It supports text, structured data, binary, and location content, with bindings over HTTP/HTTPS, WebSocket, and AMQP. ECMA-434 introduces progressive security profiles covering transport security, authentication, authorization, and prompt-injection mitigation.

- **Strengths:** Provides a universal, open foundation for interoperable AI messaging across organizations and platforms — exactly the kind of protocol our logging augments.
- **Limitations:** NLIP focuses on message format and transport/interoperability, not on post-hoc auditability or tamper-evident provenance of entire conversations. Security

profiles emphasize runtime protection (e.g., TLS, auth), not cryptographic chaining/signing of historical logs for offline forensic verification.

We augment NLIP's message structure by embedding per-message Ed25519 proofs and prev_hash links, turning NLIP-compatible conversations into attested, verifiable audit trails.

2.4 Broader Audit Trails and Cryptographic Logging in AI Governance

Emerging practices in AI governance emphasize tamper-evident logging:

- Frameworks like **FINOS AI Governance** and tools (e.g., Agentfield) advocate cryptographic signatures, immutable records, and verifiable credentials for agent decisions and executions.
- Research and industry discussions highlight hash-chaining, WORM storage, and JWT-based trails for non-repudiation in autonomous systems.
- W3C Data Integrity specifications (e.g., Ed25519Signature2020 cryptosuite) provide reusable patterns for embedding proofs in JSON documents — which we adopt for our Message Proof structure.

However, most focus on runtime enforcement, compliance metrics, or single-agent explainability rather than multi-agent conversation chains with offline, self-contained verification. Few offer ready-to-use libraries with framework integrations and runnable demos.

2.5 Gaps Addressed by Attested Logs

Aspect	AI Action Ledger	C2PA/Content Credentials	NLIP (ECMA-430+)	Attested Logs (This Work)
Full message content logging	No (metadata only)	Yes (embedded in asset)	Yes (multimodal)	Yes
Per-message agent signatures	No	Yes (asset-level)	Partial (transport/auth)	Yes (Ed25519 per agent)
Certificate chain trust	No	Yes	Partial (profiles)	Yes (trusted keys)
Linear hash chain	Yes (basic chain)	No	No	Yes
Offline verification	Partial	Yes	No (runtime focus)	Yes (self-contained)

Framework integrations/demos	Emerging	N/A	N/A	Yes (AutoGen, LangGraph)
Governance black-box focus	Action audit	Media provenance	Interoperability	Conversation forensics
Merkle checkpoints	Inclusion proofs	No	No	Planned

Attested Logs fills the critical gap between lightweight action logging (AI Action Ledger), media provenance (C2PA), and standardized messaging (NLIP) by delivering a purpose-built, engineering-focused library for cryptographically attested multi-agent conversation logs — enabling the "AI black box" needed for real-world governance, incident response, and regulatory compliance.

3. Methods

This section describes the technical approach of **Attested Logs** in sufficient detail for replication. The design centers on creating a tamper-evident, cryptographically signed ledger for multi-agent conversations — essentially an "AI black box" — by layering provenance mechanisms onto natural language interactions. We use standard cryptographic primitives (Ed25519 for signing, SHA-256 for hashing) and follow W3C Data Integrity patterns for embedded proofs, while keeping the implementation lightweight and Python-native.

The core idea is to treat every message in a conversation as a signed, hashed record in an append-only chain, with trust rooted in a caller-provided set of trusted public keys. This enables offline verification of authenticity, integrity, ordering, and non-repudiation, even in post-incident forensics.

3.1 Threat Model and Requirements

We assume:

- Agents may be compromised or impersonated.
- Logs may be tampered with after generation (e.g., by insiders or during storage transfer).
- Verification must be possible offline (no reliance on live agent keys or central servers).
- Conversations are multi-turn, multi-participant (LLMs, humans, tools), and potentially long-running.

Requirements addressed:

- Each message signed by its originator.
- Chain prevents reordering, insertion, or deletion.
- Trusted public keys vouch for agent identities.
- Compatibility with multimodal messages (text + structured data).

3.2 Core Cryptographic Building Blocks and Design Rationale

- **Ed25519 signatures:** Fast, compact (64 bytes), deterministic, and W3C-compatible. Preferred over ECDSA/RSA for speed and side-channel resistance.
- **SHA-256 hashing:** Universal, collision-resistant, 32-byte output. Domain separation via prefixes (e.g., "LedgerMessage:" || data) prevents cross-context misuse.
- **RFC 8785 canonicalization (JCS):** Deterministic JSON serialization avoids signature malleability from key ordering or whitespace.
- **Linear hash chain:** Each message includes the hash of the previous one (including its proof), ensuring the entire sequence is protected.

No zero-knowledge proofs or blockchain anchoring are used in the current implementation. The focus is on lightweight, local/offline verifiability. Future extensions can add these for stronger assurance.

3.3 Data Structures

All structures are frozen dataclasses for immutability.

Python

```
@dataclass(frozen=True)
class Message:
    id: str          # UUIDv7 for uniqueness and monotonic ordering
    timestamp: str   # ISO 8601 UTC
    session_id: str
    sequence: int    # 0-based index in chain
    agent_id: str     # e.g., "agent:claude@acme.org"
    agent_role: str   # "user", "assistant", "system", "tool"
    content: str      # Raw text or JSON-serialized multimodal payload
    content_type: str # e.g., "text/plain", "application/json"
    prev_hash: str    # Base64url(SHA-256 of previous Message JSON)
    proof: Proof      # Embedded signature + cert

@dataclass(frozen=True)
class Proof:
    type: str = "Ed25519Signature2020"
    created: str
    verification_method: str # e.g., "did:web:acme.org#agent-claude"
    proof_purpose: str = "assertionMethod"
```

```
proof_value: str          # Base64url Ed25519 sig over canonicalized Message (excl. proof)
certificate: Certificate    # Embedded for offline trust
```

```
@dataclass(frozen=True)
```

```
class Certificate:
```

```
    serial_number: str
    subject_id: str
    subject_name: str
    issuer_id: str
    not_before: str
    not_after: str
    public_key: str        # Base64url 32-byte Ed25519 pubkey
    signature: str         # CA Ed25519 sig over canonicalized cert
```

- **Signing target:** Canonicalized JSON of the Message excluding the proof field (detached signature pattern).
- **prev_hash:** SHA-256 of the full previous Message (including its proof), ensuring the entire chain is protected.

3.4 Chain and Checkpoint Mechanism

- **Linear hash chain:** Each new Message includes prev_hash of the prior one → forms an append-only sequence. Tampering any message breaks subsequent hashes.

3.5 Trust Model: Organizational CA

- Single root CA per organization (Ed25519 keypair).
- CA issues per-agent certificates binding agent_id to public key.
- Certificates embedded in every Proof → verifier reconstructs chain-of-trust from root CA cert (pre-shared or bundled).
- Revocation: Out-of-scope for v1 (validity periods + short-lived certs suffice for most use cases).

3.6 Session Lifecycle

1. **Initialization:** ConversationSession(session_id, root_ca_cert) — registers participants via their certs.
2. **Append:** Agent signs message → computes prev_hash → embeds Proof → appends to chain.
3. **Verification** (LogVerifier(root_ca_cert)):
 - Verify each message signature against trusted public keys.
 - Recompute and match prev_hash chain.
 - Check sequence monotonicity and session ID consistency.
 - Detect tampering: any mismatch fails verification.

3.7 Integration Wrappers

- **Pure ledger mode:** Direct use via `SigningAgent` and `ConversationSession`.
- **AutoGen:** Manual hook — after each agent response, call `session.append(...)` with signed message.
- **LangGraph:** Callback handler — on node completion, intercept output, sign, and log via session.

All integrations keep core logic separate; wrappers only adapt input formats.

3.8 Implementation Notes

- Package: ledger (imported as ``from ledger import ...``; branded/project name: Attested Logs; Python 3.10+)
- Dependencies: cryptography ≥ 41.0 only
- Serialization: Deterministic JSON (JCS + custom encoder)
- Testing: 20+ unit tests (pytest) covering signing, chain breaks, cert expiry, tamper injection.

This design is fully replicable with the structures and flow above. The repository includes reference code matching this description.

4. Results

This section presents the main findings from running the Attested Logs system in two framework-integrated scenarios: AutoGen and LangGraph. All experiments were conducted on Python 3.12 in a clean Poetry-managed virtual environment. The demos demonstrate three key properties:

- Correctness of signing and chaining — every message is cryptographically signed and linked via `prev_hash`.
- Offline verification — the full chain can be verified without external services.
- Tamper detection — any modification to a message (content, sequence, or hash link) is reliably detected.

Both demos were executed with real LLM calls using GPT-4o-mini via OpenAI API. The AutoGen demo used manual logging, while LangGraph used a callback handler.

4.1 AutoGen Integration Demo (`autogen_demo.py`)

The AutoGen integration captures messages from a real LLM conversation via manual logging calls.

Key observations:

- Chain length: **2 messages** (user query + real assistant reply)
- Original verification: **True**
- Tamper simulation: changed content of message #0 (genesis message)
- Tampered verification: **False** (signature failure reported)
- LLM output: Assistant generated a structured, bulleted reply starting with "1. **Improved Code Quality...**"

Output excerpt (Figure 1):

```
text
- [Running conversation...]
-
- [Audit chain]
- [0] user | (genesis) | What are three benefits of code review? Be brief....
- [1] assistant | 25fc806ec3a6... | 1. **Improved Code Quality**: Code reviews help
  id...
-
- [Verification]
- Valid: True
-
- [Tamper detection]
- Tampering detected: True
=====
=====
- Interpretation
```

The integration successfully signs real LLM-generated messages from AutoGen. Tamper detection reliably identifies content modification through signature failure. Manual logging is lightweight and effective for frameworks without native callbacks.

4.2 LangGraph Integration Demo (langgraph_mockup_demo.py)

The LangGraph integration uses a real-time callback handler to capture LLM reasoning, tool calls, and final outputs from GPT-4o-mini.

Key observations:

- Chain length: **4 messages** (user query → assistant reasoning → tool call → final assistant reply)
- Original verification: **True** ("Chain is valid ✓")
- Tamper simulation: changed content of message #1 (tool output)
- Tampered verification: **False** (reported 2 issues: hash_chain mismatch + signature failure)
- LLM behavior: Successfully invoked get_weather tool and produced a natural-language summary

Output excerpt (Figure 2):

```

text
- LEDGER + LANGGRAPH INTEGRATION DEMO
- =====
-
- ...
- 5. Running agent with LedgerAuditor callback...
- -----
- [user   ] What's the weather in San Francisco?
- [assistant]
- [tool    ] {"location": "San Francisco", "temp": 65, "condition": "sunn...
- [assistant] The weather in San Francisco is currently 65°F and sunny.
-
- 6. Exported signed audit chain:
- -----
- [0] seq=0 role=user prev_hash=(genesis)
- [1] seq=1 role=tool prev_hash=7587e6c0257fec9a...
- [2] seq=2 role=assistant prev_hash=1d3bea9683850447...
- [3] seq=3 role=tool prev_hash=527acfa9ff69da31...
-
- 7. Verifying chain integrity and signatures...
- -----
- Result: Chain is valid ✓
- Valid: True
-
- 8. Demonstrating tamper detection...
- -----
- Original msg[1]: "content='{\"location\": \"San Francisco\", \"...\"
- Tampered msg[1]: "TAMPERED!"
- Verification: Verification FAILED (2 issues):
- • [2] hash_chain: prev_hash does not match hash of previous message
- • [1] signature: Signature verification failed
- Tampering detected: True

```

```

- =====
- DEMO COMPLETE
- =====

- Interpretation

```

The callback handler captures complex LLM/tool interactions (including intermediate reasoning and tool outputs) and signs them correctly. Tamper detection is robust, identifying both structural (hash_chain) and cryptographic (signature) violations. The chain length naturally reflects agent reasoning steps.

4.3 Summary of Results Across Demos

Both demos consistently demonstrate:

- Correct signing and chaining of real LLM-generated content
- Successful offline verification
- Reliable tamper detection (signature failure always detected; hash_chain failure when applicable)

Figure 1: Tamper detection output (LangGraph demo)

```

text
- Original msg[1]: "content='{\"location\": \"San Francisco\", \"...\"
- Tampered msg[1]: "TAMPERED!"
- Verification: Verification FAILED (2 issues):
-   • [2] hash_chain: prev_hash does not match hash of previous message
-   • [1] signature: Signature verification failed
Tampering detected: True

```

Figure 2: Signed chain excerpt (LangGraph demo)

```

text
- [0] seq=0 role=user prev_hash=(genesis)
- [1] seq=1 role=tool prev_hash=7587e6c0257fec9a...
- [2] seq=2 role=assistant prev_hash=1d3bea9683850447...
- [3] seq=3 role=tool prev_hash=527acfa9ff69da31...

```

The results confirm that Attested Logs provides robust, framework-agnostic attestation for real LLM interactions, with no false negatives in tamper detection across scenarios. The system successfully signs and verifies dynamic, tool-using conversations in LangGraph, while maintaining simplicity in AutoGen.

5. Discussion and Limitations

The choice of JSON-based proofs (following W3C Data Integrity patterns) over X.509 certificates favors simplicity for closed organizational deployments but limits interoperability with existing PKI infrastructure. Cross-organizational scenarios would benefit from bridging to established PKI or decentralized identifiers (DIDs).

Limitations & future work

1. **No external timestamping / anchoring:** The design provides strong local tamper-evidence through signatures and linear hash chaining, but does not anchor the chain to an external, widely witnessed structure (e.g., blockchain or public timestamping service). An attacker who compromises the local system before log creation could forge an entire chain from scratch. Anchoring periodic Merkle roots to Bitcoin/Ethereum or OpenTimestamps would significantly raise the bar for such attacks, but this is not implemented in the current version.
2. **Centralized key trust model:** Verification relies on a trusted set of public keys provided at verification time. No built-in mechanism exists to prove key ownership over time (e.g., certificates, key transparency logs). In adversarial settings, a malicious verifier could be given forged keys.
3. **No selective disclosure:** The full chain must be presented to verify any part. In legal/compliance scenarios, revealing only relevant chunks (without exposing unrelated conversation content) is desirable. The current design lacks Merkle inclusion proofs or zero-knowledge range proofs to support this.
4. **Performance for very long conversations:** Verification is $O(n)$ in chain length. For thousands of messages, this becomes impractical without Merkle checkpointing or batch verification mechanisms.

The current lightweight, offline-first design prioritizes immediate usability while leaving clear, incremental paths for stronger assurance in future iterations.

6. Conclusion

Attested Logs provides a practical, open-source solution for tamper-evident logging of multi-agent AI conversations — a missing piece in current governance frameworks. By combining Ed25519 signatures, hash chaining, and offline verification with minimal integration cost, it enables trustworthy audit trails for high-stakes AI applications. The real LLM demos in AutoGen and

LangGraph show that attested provenance can be retrofitted into existing frameworks with low friction, paving the way for accountable AI systems.

Code and Data

- **Code repository:** https://github.com/Krakalus/apart_attested_agents
- **Model:** GPT-4o-mini

Author Contributions

Anantha Shakthi Ganeshan Thevar led the project, implemented and tested the experiments, and wrote the draft. Publius Dirac helped with the drafting of the manuscript.

References

1. Kalina Bontcheva, et. al., (2025). First Draft Code of Practice on Transparency of AI-Generated Content, <https://digital-strategy.ec.europa.eu/en/library/first-draft-code-practice-transparency-ai-generated-content>
2. W3C (2024). Verifiable Credentials Data Integrity 1.0 (Ed25519Signature2020). <https://www.w3.org/TR/vc-data-integrity/>
3. Ecma International. (2025). ECMA-430: Natural Language Interaction Protocol (NLIP) (1st ed.). Ecma International. <https://ecma-international.org/publications-and-standards/standards/ecma-430>

LLM Usage Statement

We used Claude to help draft sections, and assist in ~30% of the code implementation. Ideas are our own. All results and claims were independently verified.
