Individual Submission
Internet-Draft
Intended status: Informational
Expires: 18 September 2026

B. Stone
SwarmSync.AI

March 2026

AIVS: Agentic Integrity Verification Standard
draft-stone-aivs-00

Abstract

This document specifies the Agentic Integrity Verification Standard (AIVS), a portable, self-verifiable archive format for cryptographic proof of AI agent sessions. An AIVS proof bundle is a gzip-compressed tar archive containing a SHA-256 hash-chained audit log, an Ed25519 digital signature over the chain, a machine-readable manifest, and an embedded verification script that requires only the Python 3 standard library to execute. AIVS also defines AIVS-Micro: a minimal six-field attestation (~200 bytes) for continuous monitoring and API contexts where a full session bundle is not required. AIVS enables any party to independently verify that every action in an agent session is accounted for and unmodified, that no actions have been inserted, deleted, or reordered, and that the session was produced by a specific cryptographic identity — entirely offline, without installing any software beyond Python 3.

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 2 September 2026.

Copyright Notice

Table of Contents

1. 1. Introduction

1.1. 1.1 Problem Statement

AI agents increasingly perform consequential actions on behalf of humans: navigating websites, filling forms, executing JavaScript, extracting data, and making purchases. Existing observability platforms (OpenTelemetry, LangSmith, Langfuse) log these actions but provide no cryptographic guarantees that the logs are complete, unmodified, or authentic.

Regulatory frameworks mandate audit trails but do not prescribe formats:

EU AI Act Article 19 requires automatically generated logs for high-risk AI systems but specifies no format.

ISO/IEC 42001:2023 Annex A.6.2.8 requires event logging but defines no data structure.

NIST AI RMF requires documentation and audit trails but deliberately avoids prescribing formats.

This creates a gap: organizations that must prove what their AI agents did have no standard way to produce, exchange, or verify that proof.

1.2. 1.2 Design Goals

Goal	Rationale
Self-verifiable	Bundles verifiable without contacting any server, blockchain, or authority.
Portable	A single file that can be emailed, stored, or submitted to any system.
Tamper-evident	Modifying any action in the log must be detectable.
Zero dependencies	Verification requires only Python 3 standard library.
Session-level	Covers an entire session (sequence of actions).
Lightweight profile	AIVS-Micro provides ~200-byte attestation for high-frequency monitoring.
Domain-agnostic	Applicable to any AI agent performing any action.

1.3. 1.3 Relationship to Existing Standards

Standard	Scope	AIVS Relationship
W3C VC 2.0	Identity claims	AIVS bundles may be wrapped as VC credentialSubject
IETF SCITT	Supply chain transparency	AIVS bundles may be registered as SCITT signed statements
C2PA v2.2	Media asset provenance	AIVS applies the same manifest-chain concept to agent actions
Certificate Transparency RFC 6962	Append-only Merkle logs	AIVS hash chain is a simplified linear variant
EU AI Act Art. 19	Audit log requirements	AIVS is a concrete format satisfying Article 19

1.4. 1.4 Terminology

Term	Definition
Session	A bounded sequence of actions performed by a single AI agent instance, identified by a session_id.
Action	A single operation performed by the agent (e.g., navigate to URL, click element, execute JavaScript).
Audit Row	A JSON object recording one action with its inputs, outputs, timestamp, cost, and hash chain fields.
Hash Chain	A sequence of audit rows where each row’s hash depends on the previous row’s hash.
Chain Hash	A single SHA-256 hash computed over all row hashes, serving as a fingerprint of the entire session.
Proof Bundle	A .tar.gz archive containing the audit log, signature, manifest, public key, and verifier script.
AIVS-Micro	A minimal 6-field JSON proof (~200 bytes) for a single-URL scan attestation.
Identity Key	An Ed25519 keypair used to sign the chain hash.

2. 2. Hash Chain Specification

2.1. 2.1 Row Hash Computation

Each audit row is identified by a deterministic SHA-256 hash. The hash input is a colon-separated string of exactly seven fields in this order:

```
row_hash = SHA-256(
    "{row_id}:{session_id}:{action_type}:{tool_name}:{cost_cents}:{timestamp}:{prev_hash}
"
)
```

The hash is represented as a lowercase hexadecimal string (64 characters).

2.2. 2.2 Field Definitions

Field	Type	Description
row_id	Integer	Monotonically increasing row identifier (1-indexed).
session_id	String	Unique identifier for the session.
action_type	String	Classification of the action. Default: "tool_call".
tool_name	String	Namespaced tool identifier (e.g., "browser.navigate").
cost_cents	Integer	Cost of the action in cents. 0 for free actions.
timestamp	Float	Unix timestamp with fractional seconds.
prev_hash	String	row_hash of the immediately preceding row. Empty string for the first row.

2.3. 2.3 Chain Integrity Property

For a chain of N rows, modifying any field of row K invalidates row_hash[K], which invalidates prev_hash[K+1], and so on through row_hash[N]. This means:

Insertion of a row is detectable (changes all subsequent row_id values and hashes).

Deletion of a row is detectable (breaks the prev_hash link).

Reordering of rows is detectable (changes prev_hash linkage).

Modification of any field is detectable (changes the affected row's hash and all subsequent hashes).

2.4. 2.4 Chain Hash Computation

```
If rows is empty:
    chain_hash = SHA-256(b"empty")
Else:
    combined = concatenate(row_hash[1], row_hash[2], ..., row_hash[N])
    chain_hash = SHA-256(combined.encode("utf-8"))
```

The chain hash is represented as a lowercase hexadecimal string (64 characters).

3. 3. Audit Row Schema

3.1. 3.1 Row Format

```
{
  "id":           1,
  "session_id":   "sess-abc123",
  "action_type":  "tool_call",
  "tool_name":    "browser.navigate",
  "inputs_json":  "{\"url\": \"https://example.com\"}",
  "outputs_json": "{\"title\": \"Example Domain\"}",
  "cost_cents":   0,
  "error":        "",
  "timestamp":    1710252645.123456,
  "prev_hash":    "",
  "row_hash":     "a1b2c3d4e5f6..."
}
```

3.2. 3.2 Field Specifications

Field	Type	Required	Description
id	Integer	Yes	Row identifier, 1-indexed, monotonically increasing.
session_id	String	Yes	Session identifier.
action_type	String	Yes	Action classification.
tool_name	String	Yes	Namespaced tool identifier.
inputs_json	String	Yes	JSON-encoded action inputs. Sensitive keys MUST be redacted.
outputs_json	String	Yes	JSON-encoded action outputs. MAY be truncated.
cost_cents	Integer	Yes	Action cost in cents.
error	String	Yes	Error message; empty if success.
timestamp	Float	Yes	Unix timestamp, fractional secs.
prev_hash	String	Yes	Previous row’s row_hash.
row_hash	String	Yes	This row’s computed SHA-256 hash.

3.3. 3.3 Sensitive Input Redaction

Before computing the row hash, implementations MUST redact values for input keys matching any of the following case-insensitive substrings:

password, token, api_key, secret, key, authorization, bearer, credential, passwd, passphrase

Redacted values MUST be replaced with the string "[REDACTED]".

3.4. 3.4 Output Truncation

Implementations MAY truncate outputs_json to a maximum length. The reference implementation truncates to 2000 characters. Truncation does not affect the hash chain because outputs_json is not included in the row hash computation.

4. 4. Ed25519 Signature

4.1. 4.1 Signing

```
signature_bytes = Ed25519_Sign(private_key, chain_hash.encode("utf-8"))
signature_b64   = Base64_Encode(signature_bytes)
```

4.2. 4.2 Identity Key Properties

Property	Value
Algorithm	Ed25519 (RFC 8032)
Private key size	32 bytes
Public key size	32 bytes
Storage format	Raw bytes (not PEM)
Public key repr.	64-character lowercase hexadecimal string
File permissions	0600 (owner read/write only)

4.3. 4.3 Signature File Format (session_sig.txt)

```
chain_hash:{64-char-hex-chain-hash}
signature:{base64-encoded-signature}
```

4.4. 4.4 Signature is Optional

Implementations MAY produce bundles without Ed25519 signatures. The hash chain provides tamper-evidence independent of the signature. The signature adds identity binding (proof of who produced the bundle).

5. 5. AIVS Full Bundle Format

5.1. 5.1 Archive Structure

```
aivs_proof_{session_prefix}_{unix_timestamp}.tar.gz
└── session_proof/
    ├── audit_log.jsonl       # Hash-chained action log
    ├── manifest.json         # Bundle metadata
    ├── session_sig.txt       # Ed25519 signature
    ├── public_key.pem        # Signer's public key
    └── verify.py             # Self-contained verifier (stdlib only)
```

5.2. 5.2 manifest.json

```
{
  "session_id":    "sess-abc123",
  "exported_at":   "2026-03-14T15:30:45Z",
  "action_count":  42,
  "chain_hash":    "a1b2c3d4...",
  "aivs_version":  "1.0",
  "generator":     "ExampleAgent",
  "generator_url": "https://github.com/example/agent"
}
```

5.3. 5.3 verify.py Requirements

The embedded verifier script MUST:

Verify the hash chain using only Python 3 standard library (hashlib, json, sys, pathlib).

Exit with code 0 on successful verification.

Exit with code 1 if the hash chain is broken or the signature is invalid.

Print human-readable verification results to stdout.

The embedded verifier MAY verify the Ed25519 signature if the cryptography library is available.

5.4. 5.4 Bundle Chaining (Optional)

When a session produces multiple sequential bundles, each bundle MAY reference its predecessor via previous_bundle_hash.txt, containing the SHA-256 hex digest of the prior .tar.gz file. This forms a scan chain: a tamper-evident sequence of bundles where each bundle cryptographically references its predecessor.

6. 6. AIVS-Micro

6.1. 6.1 Purpose

AIVS-Micro is a minimal single-URL scan attestation (~200 bytes) designed for use cases where a full session bundle is impractical: continuous monitoring, embedded score widgets, API responses, and DNS TXT record verification.

6.2. 6.2 Micro Proof Format

```
   {
     "url":                  "https://example.com",
     "dom_hash":             "sha256:a1b2c3d4e5f6...",
     "timestamp":            "2026-03-14T10:22:01.000000000Z",
     "signature":            "ed25519:BASE64_ENCODED_SIGNATURE",
     "scanner_version_hash": "sha256:def456...",
     "scan_origin":          "local"
   }
```

6.3. 6.3 Canonical Payload for Signing

```
   {url}|{dom_hash}|{timestamp}|{scanner_version_hash}|{scan_origin}
```

6.4. 6.4 Micro Proof Verification

```
1. Reconstruct canonical payload:
   payload = "{url}|{dom_hash}|{timestamp}|{scanner_version_hash}|{scan_origin}"

2. Decode signature:
   sig_b64 = micro_proof["signature"].removeprefix("ed25519:")
   sig_bytes = Base64_Decode(sig_b64)

3. Verify:
   Ed25519_Verify(public_key, sig_bytes, payload.encode("utf-8"))

4. If verification succeeds: PASS
5. If verification fails:   FAIL
6. If signature == "unsigned": SKIP
```

7. 7. Verification Algorithm

7.1. 7.1 Hash Chain Verification (REQUIRED)

```
  Input:  audit_log.jsonl
  Output: PASS or FAIL with row number

  prev_hash = ""
  for each row in audit_log.jsonl (ordered by id):
      expected = SHA-256(
          "{row.id}:{row.session_id}:{row.action_type}:"
          "{row.tool_name}:{row.cost_cents}:{row.timestamp}:{prev_hash}"
      )
      if row.row_hash != expected:
          FAIL at row.id
      prev_hash = row.row_hash

  PASS: all N rows verified
```

7.2. 7.2 Ed25519 Signature Verification (OPTIONAL)

```
Input:  session_sig.txt, public_key.pem
Output: PASS, FAIL, or SKIP

1. Parse chain_hash and signature from session_sig.txt
2. Parse public key hex from public_key.pem
3. If no signing key configured: SKIP
4. Reconstruct Ed25519PublicKey from raw bytes
5. Verify: Ed25519_Verify(public_key, signature,
           chain_hash.encode("utf-8"))
6. If verification succeeds: PASS
7. If verification fails:    FAIL (exit 1)
8. If cryptography library unavailable: SKIP with notice
```

7.3. 7.3 Exit Codes

Code	Meaning
0	Hash chain verified. Signature verified (if available).
1	Hash chain broken OR signature invalid.

8. 8. Security Considerations

8.1. 8.1 What AIVS Proves

Integrity: The sequence of actions has not been modified since the bundle was created.

Completeness: No actions have been inserted or deleted from the chain.

Ordering: Actions occurred in the recorded sequence.

Identity (with signature): The bundle was produced by the holder of a specific Ed25519 private key.

8.2. 8.2 What AIVS Does NOT Prove

Truthfulness: AIVS does not prove that the recorded inputs/outputs actually occurred. A malicious agent could fabricate actions and produce a valid chain.

Timeliness: Timestamps are self-reported. Implementations requiring trusted timestamps SHOULD layer RFC 3161 on top.

Key authenticity: AIVS does not include a PKI or certificate chain. The public key in the bundle is self-asserted.

8.3. 8.3 Threat Model

Threat	Mitigated By
Post-hoc modification of log	Hash chain (Section 2)
Deletion of actions	Sequential prev_hash chaining
Insertion of actions	Sequential row_id + prev_hash
Reordering of actions	prev_hash depends on previous hash
Impersonation of agent identity	Ed25519 signature (Section 4)
Exposure of sensitive inputs	Mandatory redaction (Section 3.3)
Replay of old proof bundle	session_id + timestamp uniqueness

9. 9. IANA Considerations

This document defines no new IANA registries. The following existing standards are referenced: SHA-256 (FIPS 180-4), Ed25519 (RFC 8032), JSON (RFC 8259), gzip (RFC 1952).

10. References

10.1. Normative References

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996, <https://www.rfc-editor.org/rfc/rfc1952>.

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

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

10.2. Informative References

[I-D.rosenberg-oauth-aauth] Rosenberg, J., "OAuth for Agentic AI Authorization", Work in Progress, Internet-Draft, draft-rosenberg-oauth-aauth-01, 2026, <https://datatracker.ietf.org/doc/html/draft-rosenberg-oauth-aauth-01>.

[I-D.stone-vcap] Stone, B., "VCAP: Verified Commerce for Agent Protocols", Work in Progress, Internet-Draft, draft-stone-vcap-00, 2026, <https://datatracker.ietf.org/doc/html/draft-stone-vcap-00>.

[RFC3161] Adams, C., "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, August 2001, <https://www.rfc-editor.org/rfc/rfc3161>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

Author’s Address

Ben Stone
SwarmSync.AI
Email: benstone@swarmsync.ai