Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 2 October 2026

R. Sharif
CyberSecAI Ltd

31 March 2026

Cryptographic Attestation for AI Model Lifecycle: From Training Data to Inference Output

draft-sharif-ai-model-lifecycle-attestation-00

Abstract

This document defines a cryptographic attestation framework for the complete lifecycle of artificial intelligence models, from training data provenance through model weight signing, quantization verification, deployment attestation, and per-inference output signing. The framework creates an unbroken chain of cryptographic evidence binding each inference output to the specific model version, training data, and deployment configuration that produced it.

The framework uses ECDSA P-256 digital signatures, SHA-256 hash functions, Merkle trees for corpus attestation, and JSON Web Key Sets (JWKS) for key discovery. It addresses documented threats including model distillation attacks, quantization poisoning, training data manipulation, silent model degradation, and inference output tampering.

This specification complements the Agent Trust Transport Protocol (ATTP) [draft-sharif-attp-agent-trust-transport], MCPS message signing [draft-sharif-mcps-secure-mcp], and the Agent Audit Trail format [draft-sharif-agent-audit-trail] to provide end-to-end cryptographic verification from data ingestion to consumer delivery.

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 October 2026.

Copyright Notice

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

Artificial intelligence models traverse a complex lifecycle from training data collection through model training, optional fine-tuning, quantization, deployment, and inference serving. At each stage, the model or its outputs may be tampered with, substituted, or misrepresented.

Current security measures address individual stages in isolation:

- Sigstore (OpenSSF) signs model files at rest but does not cover inference outputs or training data provenance.
- CycloneDX and SPDX document model lineage as metadata but provide no cryptographic verification.
- TLS protects model downloads in transit but not against modification after TLS termination.
- No production system provides per-inference output signing that binds outputs to specific model versions and training data.

This specification defines a unified cryptographic attestation framework that creates an unbroken chain from training data to inference output. Each stage produces a signed attestation that references the previous stage, enabling end-to-end verification.

The framework is motivated by documented incidents:

- Industrial-scale model distillation: Three AI laboratories created 24,000+ fraudulent accounts and conducted 16 million+ exchanges to extract capabilities from a frontier model (February 2026).
- Quantization poisoning: Research demonstrated 88.7% success rate for adversarial weight injection in quantized models that pass all standard security checks (2025).
- Training data poisoning: Only approximately 250 poisoned documents can compromise LLMs across all model and dataset sizes (2025).

- Supply chain compromise: A compromised PyPI package with 97 million monthly downloads intercepted AI service credentials (March 2026).

- Silent model degradation: API providers substituting lower-capability models without notification to consumers.

1.1. Scope

This specification covers:

- Cryptographic attestation of training data corpora

- Digital signing of model weights (full-precision and quantized)

- Deployment attestation binding signing keys to specific model instances

- Per-inference output signing with model provenance metadata

- Verification procedures for consumers

- Key management lifecycle

This specification does not cover model watermarking, fingerprinting, or behavioural verification methods, which operate at a different layer.

2. Terminology

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] when, and only when, they appear in all capitals, as shown here.

Training Data Attestation: A signed Merkle tree root covering the SHA-256 hashes of all documents in a training corpus.

Model Weight Signature: An ECDSA P-256 signature over the SHA-256 hash of model weight tensors.

Quantization Attestation: A signed record binding a quantized model to the full-precision model from which it was derived, including per-layer error bounds.

Deployment Attestation: A signed record binding an inference signing key to a specific model deployment, including model identity, weight hash, and hardware attestation.

Inference Signature: An ECDSA P-256 signature over an inference output, bound to model identity metadata and a verifiable timestamp.

Attestation Chain: The ordered sequence of attestations from training data through inference output, where each attestation references the hash of the previous attestation.

3. Threat Model

3.1. Threats to Training Data

TD1 - Data Poisoning: Adversary inserts malicious documents into the training corpus to manipulate model behaviour.

Research demonstrates that approximately 250 poisoned documents are sufficient to compromise models of any size.

TD2 - Unlicensed Data: Training corpus includes copyrighted material without authorisation. Multiple lawsuits with damages exceeding $4 billion are pending as of March 2026.

TD3 - Data Provenance Falsification: Provider claims training data meets certain quality or licensing criteria without cryptographic proof.

3.2. Threats to Model Weights

MW1 - Weight Theft: Adversary exfiltrates model weights for unauthorised use or competitive advantage. The first criminal conviction for AI-related economic espionage occurred in January 2026.

MW2 - Weight Tampering: Adversary modifies model weights to insert backdoors or degrade performance.

MW3 - Silent Substitution: Provider silently replaces a high-capability model with a lower-capability model to reduce serving costs.

3.3. Threats to Quantization

QZ1 - Quantization Poisoning: Adversary produces a malicious quantized model that appears benign in full precision but contains adversarial behaviour in the quantized representation. Demonstrated at 88.7% success rate.

QZ2 - Quantization Misrepresentation: A quantized model is falsely claimed to be derived from a specific full-precision model.

3.4. Threats to Inference

IF1 - Output Tampering: Intermediary modifies inference outputs after generation and before delivery to the consumer.

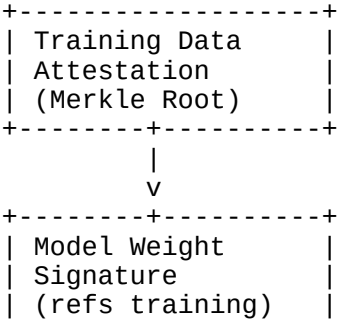
IF2 - Output Replay: Adversary replays a previously generated output for a different query.

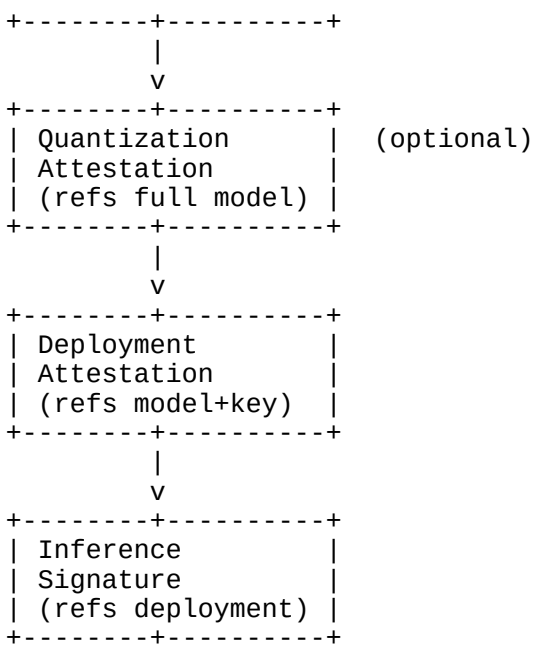
IF3 - Model Impersonation: Adversary serves outputs from a different model (including distilled copies) while claiming they are from the original model.

IF4 - Timestamp Falsification: Provider backdates or postdates inference timestamps.

4. Attestation Chain Architecture

The attestation chain creates a linked sequence of signed records:





Each attestation contains:

- A reference (hash) to the previous attestation in the chain
- Stage-specific metadata
- An ECDSA P-256 signature over the attestation content
- A timestamp

Verification proceeds by walking the chain from any point back to the training data attestation, verifying each signature and hash reference.

5. Training Data Attestation

5.1. Corpus Hashing

Each document in the training corpus is hashed individually:

```
document_hash = SHA-256(document_bytes)
```

Document hashes are assembled into a Merkle tree. The tree is constructed with SHA-256 as the hash function, using the method specified in RFC 6962 Section 2.1 (certificate transparency Merkle tree).

5.2. Attestation Format

```
{
  "attestation_type": "training_data",
  "version": "1.0",
  "corpus_id": "gpt4-train-2026-q1",
  "merkle_root": "<SHA-256 Merkle root>",
  "document_count": 15000000,
  "total_tokens": 1200000000000,
  "license_attestation": {
    "all_licensed": true,
    "license_types": ["CC-BY-4.0", "Apache-2.0", "public-domain"],
    "audit_date": "2026-03-01T00:00:00Z"
  },
  "poison_screening": {
    "method": "statistical_outlier_detection",
```

```
      "screening_date": "2026-03-01T00:00:00Z",
      "documents_flagged": 0
    },
    "timestamp": "2026-03-15T10:00:00Z",
    "issuer": "training-pipeline-ca",
    "signature": "<ECDSA-P256-signature>"
  }
```

5.3. Inclusion Proofs

For any individual document, the provider can generate a Merkle inclusion proof demonstrating that the document was part of the attested corpus, without revealing the entire corpus. This enables selective disclosure for audit and legal purposes.

6. Model Weight Signing

6.1. Weight Hashing

For models small enough to hash in their entirety:

```
weight_hash = SHA-256(concatenation of all weight tensors
                      in canonical order)
```

For large models where single-hash computation is impractical, a per-layer Merkle tree is constructed:

```
layer_hash[i] = SHA-256(weight_tensor[i])
weight_merkle_root = MerkleRoot(layer_hash[0..N])
```

6.2. Signing Format

```
{
  "attestation_type": "model_weights",
  "version": "1.0",
  "model_id": "gpt-4-turbo-2026-03-31",
  "model_version": "2026.03.31",
  "weight_hash": "<SHA-256 or Merkle root>",
  "hash_method": "merkle_per_layer",
  "layer_count": 96,
  "parameter_count": 1800000000000,
  "architecture": "transformer_decoder",
  "training_attestation_hash": "<SHA-256 of training attestation>",
  "training_run_id": "run-2026-q1-final",
  "timestamp": "2026-03-20T00:00:00Z",
  "issuer": "model-signing-ca",
  "signature": "<ECDSA-P256-signature>"
}
```

The training_attestation_hash field links this attestation to the training data attestation, creating the second link in the chain.

7. Quantization Verification

7.1. Quantization Attestation

When a model is quantized (e.g., to GGUF, AWQ, or GPTQ format), the quantization process produces an attestation:

```
{
  "attestation_type": "quantization",
  "version": "1.0",
  "source_model_id": "gpt-4-turbo-2026-03-31",
  "source_weight_hash": "<SHA-256 of full-precision weights>",
  "source_weight_signature": "<signature from Section 6>",
```

```
  "quantization_method": "gguf-q4_k_m",
  "quantized_hash": "<SHA-256 of quantized file>",
  "layer_manifest": [
    {
      "layer": "attention.0.weight",
      "source_hash": "<SHA-256>",
      "quantized_hash": "<SHA-256>",
      "max_absolute_error": 0.0012,
      "mean_squared_error": 0.00003
    }
  ],
  "model_weight_attestation_hash": "<SHA-256 of weight attestation>",
  "quantization_tool": "llama.cpp-b4567",
  "quantization_tool_hash": "<SHA-256 of quantization tool binary>",
  "timestamp": "2026-03-25T00:00:00Z",
  "issuer": "quantization-ca",
  "signature": "<ECDSA-P256-signature>"
}
```

7.2. Error Bounds

The layer_manifest includes per-layer error metrics between the full-precision and quantized weights. Verifiers can check that error bounds are within acceptable tolerances. Abnormally high errors on specific layers may indicate adversarial modification during quantization.

Implementations SHOULD define maximum acceptable error thresholds per quantization method. Quantized models exceeding these thresholds SHOULD be rejected.

8. Deployment Attestation

When a model is deployed to an inference endpoint, the deployment process generates a signed attestation binding the inference signing key to the specific model:

```
{
  "attestation_type": "deployment",
  "version": "1.0",
  "deployment_id": "deploy-gpt4t-us-east-001",
  "model_id": "gpt-4-turbo-2026-03-31",
  "weight_hash": "<SHA-256 of deployed weights>",
  "weight_attestation_hash": "<SHA-256 of weight or quant attestation>",
  "inference_key_id": "model-gpt4t-2026-03-31-001",
  "inference_public_key": {
    "kty": "EC",
    "crv": "P-256",
    "x": "<base64url>",
    "y": "<base64url>"
  },
  "hardware_attestation": {
    "platform": "nvidia-h100-confidential",
    "attestation_quote": "<base64>"
  },
  "jwks_endpoint": "https://api.provider.com/.well-known/model-keys",
  "deployment_timestamp": "2026-03-31T00:00:00Z",
  "issuer": "deployment-ca",
  "signature": "<ECDSA-P256-signature>"
}
```

The inference_public_key is the key that will sign all inference outputs from this deployment. Publishing it via JWKS enables consumers to verify outputs without contacting the provider directly.

9. Inference Output Signing

9.1. Signing Process

For each inference request, the inference engine:

1. Generates the output (text, embeddings, structured data)

2. Constructs the signature payload:

```
{
  "output_hash": "<SHA-256 of raw output bytes>",
  "model_id": "gpt-4-turbo-2026-03-31",
  "model_version": "2026.03.31",
  "weight_hash": "<SHA-256 of deployed weights>",
  "deployment_id": "deploy-gpt4t-us-east-001",
  "key_id": "model-gpt4t-2026-03-31-001",
  "timestamp": 1711900000,
  "nonce": "<unique-per-request>",
  "request_hash": "<SHA-256 of input prompt>",
  "sequence_number": 847293
}
```

3. Signs the payload with the deployment's ECDSA P-256 private key.

4. Attaches the signature as HTTP headers:

```
X-Inference-Signature: <base64-DER-encoded signature>
X-Inference-Key-ID: model-gpt4t-2026-03-31-001
X-Inference-Model-ID: gpt-4-turbo-2026-03-31
X-Inference-Weight-Hash: sha256:a7c3f8e2...
X-Inference-Timestamp: 1711900000
X-Inference-Nonce: <unique-nonce>
X-Inference-Sequence: 847293
```

9.2. Streaming Responses

For streaming responses (Server-Sent Events), the signature covers the complete accumulated output and is sent as the final SSE event:

```
event: inference-signature
data: {"signature":"...","key_id":"...","timestamp":...}
```

Implementations MAY also sign individual chunks with running hashes for incremental verification.

9.3. Input-Output Binding

When the request_hash field is present, the signature cryptographically binds a specific input to a specific output. This provides non-repudiation: the provider cannot deny generating a specific output for a specific input, and the consumer cannot claim a different input produced the output.

10. Key Management

10.1. Key Hierarchy

The framework uses a three-tier key hierarchy:

Tier 1 - Provider Root Key:
Long-lived (years). Stored in HSM with FIPS 140-3 Level 3+. Signs Tier 2 keys. Published in provider's root JWKS.

Tier 2 - Model Signing Keys:
Medium-lived (months). Signs model weight attestations and deployment attestations. One per model family.

Tier 3 - Inference Signing Keys:
Short-lived (days to weeks). Generated per deployment instance. Signs inference outputs. Published in deployment JWKS.

10.2. Key Rotation

Tier 3 keys SHOULD be rotated at least monthly. When a key is rotated:

- The new key is added to the JWKS before use.
- The old key remains in the JWKS for a grace period (RECOMMENDED: 30 days) to allow verification of recent outputs.
- After the grace period, the old key is removed from the JWKS but retained in the provider's archive for historical verification.

10.3. Key Discovery

Consumers discover provider public keys via JWKS:

GET /.well-known/model-keys

The response follows RFC 7517 (JSON Web Key) with extensions for model identity metadata.

10.4. Revocation

If an inference signing key is compromised:

- The key is immediately removed from the JWKS.
- A signed revocation notice is published at:
GET /.well-known/model-key-revocations
- Consumers checking the JWKS will not find the revoked key and MUST reject outputs signed with it.

11. Verification Procedures

11.1. Inference Output Verification

A consumer verifying an inference output:

1. Extracts X-Inference-Key-ID from the response.
2. Fetches the provider's JWKS from /.well-known/model-keys.
3. Locates the key matching the key_id.
4. Reconstructs the signature payload from the response body and headers.
5. Verifies the ECDSA signature.
6. Validates the timestamp is within an acceptable window (RECOMMENDED: 300 seconds).
7. Validates the nonce has not been seen before.

8. Optionally validates the weight_hash against known-good model versions.

11.2. Full Chain Verification

For full chain verification (e.g., for regulatory audit):

1. Verify the inference signature (Section 11.1).
2. Retrieve the deployment attestation for the deployment_id.
3. Verify the deployment attestation signature.
4. Verify the weight_hash in the deployment attestation matches the inference signature's weight_hash.
5. Retrieve the model weight attestation referenced by the deployment attestation.
6. Verify the model weight attestation signature.
7. Retrieve the training data attestation referenced by the model weight attestation.
8. Verify the training data attestation signature.
9. Each step verifies that the hash reference in the current attestation matches the actual hash of the referenced attestation.

12. Integration with ATTP and MCPS

This framework integrates with the broader agent security protocol suite:

- ATTP [draft-sharif-attp-agent-trust-transport]: Agent passports carry a reference to the model's deployment attestation when the agent uses a specific model. Consumers can verify both the agent's identity and the model's provenance.
- MCPS [draft-sharif-mcps-secure-mcp]: MCP tool call responses can include inference signatures when the tool invokes a model. The MCPS message signature covers both the tool response and the embedded inference signature.
- Agent Audit Trail [draft-sharif-agent-audit-trail]: Inference signatures are recorded in the hash-chained audit trail, creating a tamper-evident log of which models generated which outputs.

13. Regulatory Mapping

EU AI Act:

- Article 12 (Record-keeping): The attestation chain provides cryptographic evidence of the complete model lifecycle.
- Article 13 (Transparency): Training data attestation documents data sources and licensing.
- Article 15 (Cybersecurity): Per-inference signing protects output integrity.
- Article 50 (AI system identification): Inference signatures identify the specific model.

NIST AI 100-1 (AI Risk Management Framework):

- Map 1.1 (Context establishment): Deployment attestation

documents the operational context.
- Govern 1.3 (Processes and procedures): The attestation chain provides verifiable process documentation.

EU Cyber Resilience Act:
- Connected product security: Models deployed as services are products under the CRA; inference signing provides the required security properties.

14. Security Considerations

- Side-channel attacks on signing keys: Mitigated by hardware-secured key storage (HSM, TPM, TEE).

- Signing oracle attacks: An attacker with API access can generate signed outputs for arbitrary inputs. Mitigated by rate limiting and input-output binding.

- Merkle tree collision attacks: SHA-256 provides 128-bit collision resistance, sufficient for current threat models.

- Quantum computing: ECDSA P-256 is not quantum-resistant. Future revisions will define profiles for post-quantum signature algorithms (ML-DSA per NIST FIPS 204).

- Attestation chain forgery: An attacker who compromises the Tier 1 root key can forge the entire chain. Mitigated by HSM storage and multi-party key ceremonies.

- Timestamp manipulation: Mitigated by integration with trusted timestamping services (RFC 3161).

15. IANA Considerations

This document requests registration of the following HTTP headers:

- X-Inference-Signature
- X-Inference-Key-ID
- X-Inference-Model-ID
- X-Inference-Weight-Hash
- X-Inference-Timestamp
- X-Inference-Nonce
- X-Inference-Sequence

This document requests registration of the following well-known URI:

- /.well-known/model-keys
- /.well-known/model-key-revocations

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

16.2. Informative References

[draft-sharif-attp-agent-trust-transport]
Sharif, R., "Agent Trust Transport Protocol (ATTP)", Internet-Draft draft-sharif-attp-agent-trust-transport, March 2026.

[draft-sharif-mcps-secure-mcp]
Sharif, R., "MCPS: Cryptographic Security Layer for MCP", Internet-Draft draft-sharif-mcps-secure-mcp, March 2026.

[draft-sharif-agent-audit-trail]
Sharif, R., "Agent Audit Trail Format", Internet-Draft draft-sharif-agent-audit-trail, March 2026.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013.

[SIGSTORE] OpenSSF, "Model Signing with Sigstore", https://openssf.org/blog/model-signing, 2025.

[CYCLONEDX] OWASP, "CycloneDX ML-BOM Capabilities", https://cyclonedx.org/capabilities/mlbom/, 2025.

Authors' Addresses

Raza Sharif
CyberSecAI Ltd
London, United Kingdom

Email: contact@agentsign.dev
URI: https://cybersecai.co.uk