Individual Submission
Internet-Draft
Intended status: Informational
Expires: 3 October 2026

N. Todd
Infinitum Nihil

1 April 2026

Monotonic Attestation Service (MAS)
draft-todd-mas-01

Abstract

This document defines the Monotonic Attestation Service (MAS), a protocol for issuing cryptographically attested, monotonically increasing sequence numbers within named namespaces. Each attestation includes a hash chain linking it to all prior entries in the namespace, providing verifiable proof of ordering and completeness. MAS is designed to complement RFC 3161 Trusted Timestamping: where RFC 3161 proves when an event occurred, MAS proves in what order and that the sequence is complete.

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 3 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Many systems require proof that events occurred in a specific order and that no events have been inserted, removed, or reordered after the fact. Wall-clock timestamps (including RFC 3161 trusted timestamps) prove temporal existence but do not guarantee causal ordering -- two events timestamped milliseconds apart may have arrived in either order, and a compromised clock can backdate events.

MAS addresses this gap by providing a monotonic counter with hash chain attestation per namespace. Each attestation includes:

* A sequence number that strictly increases and never repeats

* A cryptographic hash of the payload being attested

* A cryptographic hash linking to the previous attestation in the chain

* A digital signature from the MAS operator

The combination of monotonic sequencing and hash chaining provides two guarantees that timestamps alone cannot:

Total ordering: Event N happened before event N+1. Not "at roughly the same time" -- before. Unambiguously.

Completeness: If the verifier has attestations 1 through N with a valid hash chain, no attestations have been inserted or removed. The sequence is intact.

1.1. Relationship to RFC 3161

MAS is designed to operate alongside RFC 3161 [RFC3161], not replace it. A system MAY request both a MAS attestation (for ordering) and an RFC 3161 timestamp (for wall-clock anchoring) for the same event. The combination provides four independent guarantees:

Guarantee	Provider
Wall-clock existence	RFC 3161 TSA
Causal ordering	MAS sequence number
Sequence completeness	MAS hash chain
Authority	MAS Ed25519 signature

Table 1

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Namespace: A named scope within which sequence numbers are issued. Sequence numbers are unique and monotonically increasing within a namespace but independent across namespaces.

Attestation: A signed record binding a sequence number to a payload hash within a namespace, chained to the previous attestation.

Chain: The ordered, hash-linked sequence of all attestations within a namespace.

Operator: The entity running the MAS instance and holding the signing key.

Requester: The entity requesting an attestation for a payload.

Verifier: Any entity validating an attestation or chain of attestations.

2. Data Model

2.1. Attestation Record

An attestation record contains the following fields:

Field	Type	Description
version	unsigned integer	Protocol version. This document defines version 1.
namespace	text string	The namespace this attestation belongs to. UTF-8 encoded.
sequence	unsigned integer (64-bit)	Monotonically increasing sequence number within the namespace. Starts at 1.
payload_hash	byte string	SHA-256 hash of the payload being attested. The MAS never sees the payload itself.
previous_hash	byte string	SHA-256 hash of the previous attestation record in serialized form. For sequence 1, this is 32 zero bytes.
timestamp	unsigned integer (64-bit)	UNIX timestamp in milliseconds when the attestation was issued. This is the operator’s local time and is NOT a trusted timestamp -- use RFC 3161 for trusted wall-clock time.
signature	byte string	Ed25519 signature over the canonical serialization of all preceding fields.

Table 2

2.2. Canonical Serialization

The canonical serialization of an attestation for hashing and signing is CBOR [RFC8949] encoded as a definite-length array in field order:

```
[
  version,          // unsigned integer
  namespace,        // text string
  sequence,         // unsigned integer
  payload_hash,     // byte string (32 bytes, SHA-256)
  previous_hash,    // byte string (32 bytes, SHA-256)
  timestamp         // unsigned integer
]
```

The signature is computed over the SHA-256 hash of this serialized array. The signature itself is NOT included in the array that is hashed.

2.3. Chain Genesis

The first attestation in a namespace (sequence = 1) MUST use 32 zero bytes as the previous_hash. This is the chain genesis.

2.4. Chain Integrity

For any attestation with sequence N > 1, the previous_hash field MUST equal the SHA-256 [RFC6234] hash of the canonical serialization of the attestation with sequence N-1.

A verifier can confirm chain integrity by iterating from sequence 1 to N, verifying that each attestation’s previous_hash matches the hash of the prior record.

3. Protocol

3.1. Transport

MAS is transport-agnostic. This document defines bindings for HTTPS (Section 3.2) and Service Binding RPC (Section 3.3). Implementations MAY define additional transport bindings.

3.2. HTTPS Binding

3.2.1. Request Attestation

```
POST /attest HTTP/1.1
Content-Type: application/cbor

{
  "namespace": "com.example.orders",
  "payload_hash": <32 bytes, SHA-256 of payload>
}
```

3.2.2. Attestation Response

```
HTTP/1.1 200 OK
Content-Type: application/cbor

{
  "version": 1,
  "namespace": "com.example.orders",
  "sequence": 4713,
  "payload_hash": <32 bytes>,
  "previous_hash": <32 bytes>,
  "timestamp": 1710590400000,
  "signature": <64 bytes, Ed25519>
}
```

3.2.3. Verify Single Attestation

```
POST /verify HTTP/1.1
Content-Type: application/cbor

{
  "attestation": <attestation record>,
  "operator_public_key": <32 bytes, Ed25519 public key>
}

Response:

HTTP/1.1 200 OK
Content-Type: application/cbor

{
  "valid": true,
  "sequence": 4713,
  "namespace": "com.example.orders"
}
```

3.2.4. Verify Chain Segment

```
POST /verify-chain HTTP/1.1
Content-Type: application/cbor

{
  "attestations": [<attestation 1>, ... <attestation N>],
  "operator_public_key": <32 bytes>
}

Response (complete chain):
```

```
HTTP/1.1 200 OK
Content-Type: application/cbor

{
  "valid": true,
  "namespace": "com.example.orders",
  "start_sequence": 1,
  "end_sequence": 4713,
  "complete": true,
  "gaps": []
}
```

Response (broken chain):

```
{
  "valid": false,
  "namespace": "com.example.orders",
  "start_sequence": 1,
  "end_sequence": 4713,
  "complete": false,
  "gaps": [{"after": 2001, "before": 2003}],
  "first_break": 2002
}
```

3.2.5. Retrieve Attestation by Sequence

```
GET /attestation/{namespace}/{sequence} HTTP/1.1
```

3.2.6. Retrieve Chain Segment

```
GET /chain/{namespace}?from={start}&to={end} HTTP/1.1
```

3.2.7. Retrieve Operator Public Key

```
GET /key HTTP/1.1
```

Response:

```
{
  "algorithm": "Ed25519",
  "public_key": <32 bytes>,
  "valid_from": 1710590400000,
  "valid_until": null,
  "previous_keys": [
    {
      "public_key": <32 bytes>,
      "valid_from": 1700000000000,
      "valid_until": 1710590400000
    }
  ]
}
```

3.3. Service Binding RPC Binding

For environments where the MAS operator runs on the same infrastructure as the requester (e.g., Cloudflare Service Bindings, AWS Lambda extensions, sidecar containers), the same request/response format is used but transported via the platform's native RPC mechanism instead of HTTPS.

The serialization format (CBOR) and field semantics are identical. The transport binding simply replaces the HTTP layer with the platform's internal call mechanism.

This is the RECOMMENDED binding when the MAS is co-located with the requester, as it eliminates TLS overhead and DNS resolution.

4. Operational Considerations

4.1. Single-Writer Requirement

A MAS instance MUST guarantee that sequence numbers within a namespace are issued by exactly one writer. Concurrent writers on the same namespace would violate monotonicity.

Implementations SHOULD use a mechanism that provides single-writer semantics natively, such as:

* A serialized process with exclusive access to the counter

* A database with serializable isolation on the counter row

* A distributed coordination primitive with leader election

The specific mechanism is an implementation choice and not specified by this protocol.

4.2. Namespace Isolation

Sequence numbers in different namespaces are independent. An operator MAY host many namespaces. There is no ordering relationship between attestations in different namespaces.

4.3. Key Management

The operator SHOULD rotate signing keys periodically. When rotating, the operator MUST:

1. Publish the new public key via the /key endpoint before using it
2. Include the old key in the previous_keys array with its validity period
3. Sign a transition attestation in the old key that references the new key

Verifiers MUST use the key that was valid at the attestation's timestamp when verifying signatures.

4.4. Persistence

The operator MUST persist all attestations durably. Loss of attestation records breaks the hash chain for all subsequent records.

The operator SHOULD provide a retrieval API (Section 3.2.5, Section 3.2.6) so that verifiers can reconstruct and verify chain segments.

4.5. Clock Advisory

The timestamp field is advisory only -- it reflects the operator's local clock and is NOT a trusted timestamp. Systems requiring trusted wall-clock time SHOULD additionally obtain an RFC 3161 timestamp for the same payload.

The timestamp field exists to assist verifiers in correlating attestations with real-world time ranges. It MUST NOT be used as the sole basis for temporal claims.

5. Verification

5.1. Single Attestation Verification

To verify a single attestation, a verifier MUST:

1. Obtain the operator's public key that was valid at the attestation's timestamp

2. Reconstruct the canonical serialization (Section 2.2) from the attestation fields

3. Compute the SHA-256 hash of the canonical serialization

4. Verify the Ed25519 [RFC8032] signature over this hash using the operator's public key

A valid signature proves the operator issued this attestation with these exact field values.

5.2. Chain Verification

To verify a chain segment from sequence S to sequence E, a verifier MUST:

1. Verify each individual attestation per Section 5.1

2. For each attestation with sequence N > S, verify that previous_hash equals SHA-256 of the canonical serialization of attestation N-1

3. If S = 1, verify that attestation 1 has previous_hash equal to 32 zero bytes

If all verifications pass, the chain segment is complete and unmodified.

5.3. Gap Detection

If a verifier has attestations with non-contiguous sequence numbers (e.g., 100, 101, 103), this indicates either:

* Attestation 102 was not provided (incomplete disclosure)

* Attestation 102 never existed (operator error)

The verifier SHOULD report the gap and MUST NOT treat the chain as complete.

5.4. Fork Detection

If a verifier encounters two different attestations with the same namespace and sequence number but different content, this indicates a fork -- the operator (or an attacker) issued conflicting attestations. This is a critical integrity violation.

A verifier that detects a fork MUST reject both attestations and SHOULD alert the relying party.

6. Interaction with RFC 3161

A system that uses both MAS and RFC 3161 [RFC3161] for the same event has two independent attestations:

1. MAS attestation: proves ordering and chain completeness

2. RFC 3161 timestamp token: proves wall-clock existence

These can be bundled together as a dual attestation:

```
{
  "mas_attestation": <MAS attestation record>,
  "rfc3161_token": <RFC 3161 timestamp token (DER-encoded)>,
  "binding_hash": <SHA-256 of (MAS canonical serialization ||
                   RFC 3161 token)>
}
```

The binding_hash cryptographically links the two attestations, preventing an attacker from mixing attestations from different events.

7. Security Considerations

7.1. Operator Trust

MAS requires trust in the operator. A compromised operator can issue false attestations, skip sequence numbers, or maintain a shadow chain. This is analogous to the trust model of RFC 3161 TSAs.

Mitigation: requesters MAY submit the same payload to multiple independent MAS operators and compare sequence assignments. Disagreement indicates compromise.

7.2. Denial of Service

An attacker flooding the MAS with requests could exhaust sequence space or degrade performance. Operators SHOULD implement rate limiting per requester.

The 64-bit sequence space supports 2^64 attestations per namespace, which is sufficient for all practical purposes.

7.3. Replay

Attestation responses are deterministic for a given input and state. An attacker replaying an old attestation request will receive a new attestation with a new sequence number, not a replay of the old response. The hash chain prevents inserting the replayed response into the chain.

7.4. Namespace Squatting

Namespaces are first-come-first-served within an operator. Operators SHOULD implement namespace registration and access control to prevent unauthorized use.

7.5. Quantum Resistance

Ed25519 is not quantum-resistant. Future versions of this specification MAY define additional signature algorithms (e.g., ML-DSA / CRYSTALS-Dilithium) for post-quantum security. The version field enables algorithm negotiation.

8. IANA Considerations

This document has no IANA actions at this time. A future version may request registration of a well-known URI (e.g., /.well-known/mas) and a media type for MAS attestations.

9. References

9.1. 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, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

9.2. Informative References

[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, <https://www.rfc-editor.org/rfc/rfc3161>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/rfc/rfc6234>.

[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/rfc/rfc8032>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

Appendix A. Reference Implementation

A reference implementation is available as a Cloudflare Durable Object with Service Binding RPC transport. The implementation uses:

* ed25519-dalek for Ed25519 signatures
* sha2 for SHA-256 hashing
* ciborium for CBOR serialization
* Durable Object SQLite for counter persistence and attestation storage

The reference implementation is approximately 300 lines of Rust.

Appendix B. Example Chain

B.1. Genesis (sequence 1)

```
{
  "version": 1,
  "namespace": "com.example.orders",
  "sequence": 1,
  "payload_hash": "a1b2c3...",
  "previous_hash": "0000000000000000000000000000000000
                    0000000000000000000000000000000000",
  "timestamp": 1710590400000,
  "signature": "ed25519:..."
}
```

B.2. Sequence 2

```
{
  "version": 1,
  "namespace": "com.example.orders",
  "sequence": 2,
  "payload_hash": "d4e5f6...",
  "previous_hash": "sha256(canonical_serialization(
                    attestation_1))",
  "timestamp": 1710590400050,
  "signature": "ed25519:..."
}
```

B.3. Verification

A verifier with both records confirms:

1. Attestation 1: previous_hash is 32 zero bytes (genesis). Signature valid.

2. Attestation 2: previous_hash equals SHA-256 of attestation 1's canonical serialization. Signature valid.

3. Sequence is contiguous (1, 2). Chain is complete.

Appendix C. Acknowledgments

The concept of hash-chained monotonic attestation draws from Certificate Transparency [RFC9162], blockchain consensus mechanisms, and the Merkle tree structures underlying distributed ledger technologies. MAS simplifies these into a single-writer model suitable for centralized trust contexts where distributed consensus is unnecessary.

Author's Address

Norman Todd
Infinitum Nihil
Oxford, Mississippi
United States of America
Email: nt@infinitum-nihil.com