AURA Core Specification

Version: 1.0 (Draft)
Status: Normative Specification
Date: February 2026
Author: Romain Benabdelkader

Abstract

AURA Core defines a minimal evidentiary manifest format for digital content.

Its purpose is to establish verifiable facts about a digital asset:

- existence at a given point in time
- integrity (content hash)
- attribution claims (who declares what)
- temporal assertions (when the declaration was made)

AURA Core is deliberately minimal.

It is designed to be:

- evidentiary only
- non-executive
- non-enforcing
- platform-independent
- usable without any central authority

Table of Contents

---

1. Scope

This specification defines the **AURA Core manifest format**.

AURA Core specifies how factual claims about a digital asset can be expressed, signed, and verified.

1.1 What AURA Core Defines

AURA Core defines:

- The minimal conceptual structure of an evidentiary manifest
- The cryptographic requirements for verification
- The factual claims that can be asserted
- The relationship between manifests and digital assets

1.2 What AURA Core Does NOT Define

AURA Core intentionally **does not** define:

- identity verification mechanisms
- legal legitimacy or ownership determination
- rights enforcement mechanisms
- usage monitoring systems

- licensing frameworks
- platform integration requirements
- specific file formats or serialization

These concerns are **explicitly out of scope**.

---

2. Design Principles

AURA Core is designed according to the following principles:

2.1 Minimality

Only facts strictly required for evidentiary purposes are included in the specification.

2.2 Neutrality

No execution, no enforcement, no policy logic.

The manifest is a declaration of facts, not an instruction to act.

2.3 Portability

Verification MUST be possible without reliance on any specific platform, service, or operator.

2.4 Survivability

The manifest format MUST remain usable even under partial or fragmented adoption.

A manifest created today MUST be verifiable decades from now, regardless of:

- whether the creator is still active
- whether any operator still exists
- whether the original platform is available

2.5 Explicit Limitations

AURA Core makes no claim beyond what can be cryptographically proven.

All limitations are documented and acknowledged.

---

3. Core Claims

An AURA manifest **MAY** assert the following facts:

3.1 Existence Claim

That a specific digital asset existed at a declared point in time.

What this proves:

- The declaring entity was aware of this specific asset
- The asset content (as represented by its hash) existed when the manifest was created

What this does NOT prove:

- That the asset was created at that time
- That this is the first or only declaration
- That the declaring entity has any rights to the asset

3.2 Integrity Claim

That the digital asset has specific content, represented by a cryptographic hash.

What this proves:

- The exact binary content that was declared
- Any modification to the asset will be detectable

What this does NOT prove:

- That this is the original or authoritative version
- That no other versions exist
- That the content is authentic or unmodified from an earlier state

3.3 Attribution Claim

That an entity controlling a specific cryptographic key declares association with the asset.

What this proves:

- Someone with access to the private key created this declaration

- The declaration has not been tampered with

What this does NOT prove:

- The legal identity of the key holder
- The legitimacy of their claim
- That they are the creator, owner, or rights holder

3.4 Temporal Assertion

That the declaration was made at or before a certain time.

What this proves (with trusted timestamping):

- The manifest existed at the timestamp authority's recorded time

What this proves (without trusted timestamping):

- Only that the declaring entity claims a certain time

What this does NOT prove:

- The exact moment of creation
- That earlier declarations don't exist
- That the timestamp is accurate (without third-party corroboration)

---

4. Manifest Structure

4.1 Conceptual Components

An AURA manifest is a structured document containing **at minimum**:

1. **asset_hash** – cryptographic hash of the digital asset
2. **signing_key** – public key of the declaring entity
3. **declaration** – the factual claims being made
4. **signature** – cryptographic signature binding all components

4.2 Format Flexibility

AURA Core does **not** mandate:

- A specific serialization format (JSON, XML, CBOR, etc.)
- A specific encoding (UTF-8, binary, etc.)
- A specific structure or field names

Implementations **MAY** choose any format that preserves the conceptual requirements.

4.3 Extensions

Additional fields **MAY** be included as extensions.

Extensions **MUST NOT**:

- Break verification of the core manifest
- Introduce mandatory dependencies
- Require execution or enforcement logic

Extensions **SHOULD**:

- Be clearly namespaced
- Be documented separately
- Be optional to implement

4.4 Informative Example

The following JSON structure is **informative only** and represents one possible implementation:

```
{
  "asset": {
    "hash": "sha3-256:abc123..."
  },
  "issuer": {
    "public_key": "ed25519:def456..."
  },
  "declaration": {
    "declared_at": "2026-02-08T10:30:00Z"
  },
  "signature": {
    "algorithm": "ed25519",
    "value": "ghi789..."
  }
}
```

This is **not normative**. Implementations may differ significantly.

---

5. Signatures

5.1 Signature Requirement

Manifests **MUST** be signed using a cryptographic key controlled by the declaring entity.

5.2 What the Signature Proves

The signature proves:

- Control of the private key at the time of signing
- That the signed content has not been altered

5.3 What the Signature Does NOT Prove

The signature does **not** prove:

- The legal identity of the key holder
- That the key holder has rights to the asset
- That the key holder created the asset
- That the key has not been compromised

5.4 Algorithm Requirements

Implementations **SHOULD** use well-established signature algorithms such as:

- Ed25519 (recommended)
- ECDSA with appropriate curves
- RSA with appropriate key sizes

Weak or deprecated algorithms **MUST NOT** be used.

5.5 Signature Scope

The signature **MUST** cover at minimum:

- The asset hash
- The public key identifier

- The core declaration fields

The signature **MAY** cover additional fields, but this is implementation-specific.

---

6. Temporal Assertions

6.1 Self-Asserted Timestamps

Manifests **MAY** include a self-asserted timestamp.

Self-asserted timestamps have **limited evidentiary value** because:

- They can be backdated
- They cannot be independently verified
- They rely entirely on the claiming entity's honesty

6.2 Trusted Timestamping (Optional)

AURA Core supports **optional** trusted timestamping using independent Time Stamping Authorities (TSAs), such as those defined in **RFC 3161**.

6.3 How Trusted Timestamping Works

When used, the timestamp token **MUST**:

1. Cryptographically bind the manifest hash to the TSA response
2. Come from an independent third-party authority
3. Be verifiable without contacting the TSA again

6.4 Properties of Trusted Timestamps

Trusted timestamping:

- ✅ Increases evidentiary strength
- ✅ Provides independent temporal proof
- ✅ Does not introduce execution or enforcement
- ✅ Does not create dependency on a single authority

Multiple timestamp authorities **MAY** be used for redundancy.

6.5 Timestamp Limitations

Even with trusted timestamping:

- The timestamp proves when the **manifest** was created, not when the **asset** was created
- Earlier undisclosed declarations may exist
- The timestamp authority itself could be compromised (though this is detectable)

6.6 Informative Timestamp Field Example

An implementation **MAY** represent a trusted timestamp as:

```
{
  "timestamp": {
    "authority": "https://freetsa.org/tsr",
    "token": "base64_encoded_rfc3161_response",
    "signed_at": "2026-02-08T10:30:15Z"
  }
}
```

This structure is **informative and non-normative**.

---

7. Anchors

7.1 Purpose of Anchors

Manifests **MAY** reference external anchors to provide additional context for verifiers.

7.2 Types of Anchors (Non-Normative)

Examples include:

- **DNS records** (e.g., TXT record at `_aura.example.com`)
- **Well-known endpoints** (e.g., `https://example.com/.well-known/aura-key`)
- **Operator attestations** (e.g., directory entry at a service)
- **Web-of-trust assertions** (e.g., PGP key signatures)
- **Blockchain records** (e.g., transaction hash on a public ledger)

7.3 Anchor Properties

Anchors:

- ✅ Increase contextual confidence
- ✅ Provide multiple verification paths
- ✅ Are entirely optional
- ❌ Are **not** authoritative
- ❌ Do **not** replace cryptographic verification

7.4 Anchor Trust

Verifiers **MAY** choose which anchors (if any) to trust.

Different verifiers may have different trust requirements.

AURA Core does **not** mandate any specific anchor or trust model.

---

8. Explicit Non-Goals

AURA Core does **NOT** aim to:

8.1 Prove Ownership or Legitimacy

The manifest proves a declaration was made, not that the declaration is legitimate or legally valid.

8.2 Prevent Copying, Scraping, or Reuse

The manifest is informational. It does not technically prevent any action.

8.3 Enforce Compliance or Policy

The manifest contains no executable logic. It does not and cannot enforce anything.

8.4 Replace Judicial or Regulatory Processes

The manifest provides evidence for legal processes, but does not replace them.

8.5 Guarantee Adoption or Dominance

AURA Core is designed to be useful even with partial adoption. It does not require universal acceptance to provide value.

8.6 Solve Identity or Trust

AURA Core deliberately does not define identity verification. This is left to ecosystem implementations.

---

9. Threat Model

9.1 Assumed Limitations

AURA Core explicitly acknowledges the following limitations:

9.1.1 Keys May Be Compromised

If a private key is compromised:

- An attacker can create valid manifests
- Past manifests remain valid
- Key rotation or revocation requires external mechanisms

9.1.2 Anchors May Be Spoofed or Centralized

External anchors:

- Can be controlled by single entities
- May be spoofed or falsified
- Introduce trust dependencies

9.1.3 Timestamps May Be Disputed

Without third-party corroboration:

- Self-asserted timestamps can be backdated
- Timestamp authorities can be compromised
- Clock skew may cause inconsistencies

9.1.4 Incomplete Coverage

Many assets will circulate without any AURA manifest.

The absence of a manifest proves nothing.

9.2 What AURA Core Provides

Despite these limitations, AURA Core:

- ✅ Reduces evidentiary uncertainty
- ✅ Makes factual claims explicit and verifiable
- ✅ Constrains the space of dispute
- ✅ Provides a portable evidentiary artifact

9.3 What AURA Core Does NOT Provide

AURA Core:

- ❌ Does not eliminate disputes
- ❌ Does not guarantee completeness
- ❌ Does not prove which declaration is "correct"
- ❌ Does not determine legal outcomes

---

10. Relationship to Implementations

10.1 Specification vs. Implementation

This specification defines a **format**, not an **implementation**.

10.2 Implementation Freedom

Tools, services, operators, or platforms **MAY**:

- Implement AURA Core in any programming language
- Use any serialization format
- Add extensions or additional fields
- Provide user-facing services

10.3 Verification Independence

Implementations **MUST NOT**:

- Require reliance on any specific platform for verification
- Introduce mandatory dependencies on operators or services
- Break backward compatibility with core verification
- Claim ownership of the standard

10.4 Conformance Requirement

An implementation conforms to AURA Core if:

1. It can produce manifests verifiable by other implementations
2. It can verify manifests produced by other implementations
3. It respects the core claims and limitations
4. It does not introduce mandatory execution or enforcement

---

11. Conformance

11.1 Conforming Manifest

A manifest conforms to AURA Core if it:

1. Contains a cryptographic hash of the asset
2. Contains or references a public key
3. Contains a signature verifiable with that key
4. Does not require execution or enforcement for verification

11.2 Conforming Implementation

An implementation conforms to AURA Core if it:

1. Can create conforming manifests
2. Can verify conforming manifests created by other implementations
3. Does not require platform-specific dependencies for verification
4. Respects the explicit limitations and non-goals

11.3 Extensions and Conformance

Extensions do not affect conformance as long as:

- Core verification remains possible
- Extensions are optional
- Unknown extensions can be safely ignored

---

12. Security Considerations

12.1 Key Management

Implementers should follow established best practices for:

- Key generation (sufficient entropy)
- Key storage (encrypted, backed up)
- Key rotation (when compromise is suspected)
- Key revocation (external mechanisms)

12.2 Hash Algorithm Selection

Implementers should use:

- SHA-256 or SHA3-256 (minimum)
- Stronger algorithms as they become standardized
- Avoid deprecated algorithms (MD5, SHA-1)

12.3 Signature Verification

Verifiers must:

- Check signatures cryptographically
- Validate hash matches asset
- Not trust unverified anchors
- Apply appropriate skepticism to temporal claims

---

13. IANA Considerations

This document has no IANA actions.

Future extensions may require:

- Media type registration (e.g., `application/aura+json`)
- Well-known URI registration (e.g., `/.well-known/aura-key`)

---

14. References

14.1 Normative References

- **RFC 3161**: Time-Stamp Protocol (TSP)
- **RFC 8032**: Edwards-Curve Digital Signature Algorithm (EdDSA)
- **FIPS 202**: SHA-3 Standard

14.2 Informative References

- **The Impossible Proof**: Romain Benabdelkader (2026)
- **RFC 7515**: JSON Web Signature (JWS)
- **C2PA**: Coalition for Content Provenance and Authenticity

---

Appendix A: Design Philosophy

AURA Core derives its strength from restraint.

By deliberately **not** attempting to solve:

- identity verification
- rights enforcement
- usage monitoring
- legal validity

...AURA Core remains:

- neutral (no policy embedded)

- portable (no platform lock-in)
- durable (no external dependencies)
- non-capturable (no single point of control)

This minimalism is not a weakness. It is the core design strategy.

Appendix B: Comparison with Other Standards

B.1 Versus C2PA

Aspect	AURA Core	C2PA
Scope	Initial claim only	Full provenance chain
Embedding	External file	Embedded in asset
Complexity	Minimal	Comprehensive
Dependencies	None	Coalition infrastructure

Relationship: Complementary. AURA manifest can reference C2PA data.

B.2 Versus NFTs

Aspect	AURA Core	NFT
Cost	Free	Blockchain fees
Speed	Instant	Block confirmation
Format	Any	Chain-specific
Dependencies	None	Blockchain infrastructure

Relationship: Orthogonal. AURA can be anchored on blockchain optionally.

B.3 Versus Digital Signatures

Aspect	AURA Core	Plain Signature
Structure	Defined manifest	Arbitrary data

Purpose	Evidentiary claim	General authentication
Interoperability	Standard format	Format-agnostic

Relationship: AURA uses digital signatures but adds structure and semantics.

Appendix C: Informative Use Cases

C.1 Copyright Disputes

Scenario: Two parties claim authorship of a work.

AURA's role:

- Each party can present timestamped manifests
- Factual timeline becomes verifiable
- Does not determine legal outcome, but constrains dispute

C.2 AI Training Opt-Out

Scenario: Creator wants to signal exclusion from training datasets.

AURA's role:

- Manifest can include opt-out declaration (extension)
- Platform can verify declaration
- Does not technically prevent use, but documents intent

C.3 Archival Preservation

Scenario: Institution wants to preserve provenance metadata.

AURA's role:

- Manifest provides cryptographic proof of content at time of acquisition
- Remains verifiable decades later
- No dependency on original platforms

Appendix D: Frequently Asked Questions

Q: Is AURA a blockchain?
A: No. AURA uses standard cryptography without requiring blockchain. Blockchain anchoring is optional.

Q: Does AURA prevent AI from using my work?
A: No. AURA documents your declaration. Enforcement depends on legal frameworks and platform compliance.

Q: Can I verify AURA offline?
A: Yes. Verification requires only the manifest, asset, and public key. No network required.

Q: What if someone forges a manifest?
A: They would need access to the private key. Forgery without the key is cryptographically infeasible.

Q: Can AURA be used for confidential content?
A: Yes. The manifest references the hash, not the content itself. The asset can remain private.

Document Status

This is a draft specification.

Feedback is welcome via:

- GitHub: https://github.com/romainbenabdelkader/AURA-STANDARD
- Email: [contact]

Version History:

Version	Date	Changes
1.0-draft	2026-02-08	Initial public draft

License

Acknowledgments

This specification reflects principles developed in *The Impossible Proof* and incorporates feedback from early implementers and the wider community.

Special thanks to contributors who have provided technical review and implementation experience.

END OF SPECIFICATION