
OPEN ARTIFICIAL INTELLIGENCE INTEGRITY SENTINEL

OpenAIIS

Universal Digital Trust,
Governance and Safety Protocol

TECHNICAL PROTOCOL SPECIFICATION

V5.0

Tarun Sethi

October 2026

Contents

Abstract	2
1. Protocol Principles	2
1.1. Normative terminology	2
2. Actors and Trust Domains	2
3. Universal Claim Representation	3
3.1. Canonical encoding	4
3.2. Cryptographic signing	4
3.3. Content commitment	4
3.4. Typed payload commitment	4
3.5. Signature and proof version integrity	5
4. Universal Claim Types	5
4.1. Extensibility	6
4.2. Extension-type profiles	6
5. Trust, Authority and Delegation	6
5.1. Scoped authorization	6
5.2. Responsibility acceptance	7
5.3. Revocation and recovery	7
5.4. Signed approval and authenticated ordering	7
6. Consent-Governed Identity Attestations	7
6.1. Consent-intent binding	7
6.2. Identity lifecycle	8
6.3. Authenticated lifecycle resolution	8
7. Artifact Attestations and Transfer	8
7.1. Offer cancellation and replay resistance	8
8. Disputes and Governance State	8
9. Accountable Software and Autonomous-Agent Actions	9
9.1. Signed release	9
9.2. Governance chain	9
9.3. Evidence chain	9
9.4. Request authentication	9
9.5. Authorization of consequential effects	9
10. Verifiable Safety Evidence	10
10.1. Security-relevant relationships	10
10.2. Incident-advisory extension	10
10.3. Trust-state reassessment	10
11. Authenticated Registry History	10
11.1. Inclusion verification	11
11.2. Signed checkpoints	11
11.3. Witnessing and consistency	11
11.4. Authenticated lifecycle index	11
11.5. Durable claim registration and checkpoint consistency	12
12. Portable Verification	12
12.1. Verification procedure	12

12.2. Decision semantics	12
12.3. Registry HTTP interface	12
12.4. Verification response semantics	13
13. Bitcoin Historical Commitments	13
13.1. Historical-proof verification	14
13.2. Bitcoin assurance classes	14
13.3. Historical time and trust state	14
14. Decentralized Network and Consensus	15
14.1. Node responsibilities	15
14.2. Proof-of-work ordering	15
14.3. Admission and abuse prevention	15
14.4. Economic model	15
15. Interoperability	15
15.1. Format-specific interoperability	16
16. Privacy, Data Protection and Evidence Handling	16
16.1. Deliberate content disclosure	16
17. Conformance and Independent Acceptance	16
18. Protocol Conclusion	17
Appendix A — Exact Core Wire Identifiers	17
Appendix B — Core Verifier Invariants	18

Abstract

OpenAIIS defines a universal framework for cryptographically verifiable digital claims, governed authority, accountable responsibility and security-sensitive trust decisions.

The protocol establishes a common representation for signed assertions concerning digital artifacts, identities, organizations, autonomous systems, supporting evidence and changes in their authority or trust relationships. These assertions are authenticated using public-key cryptography, bound to exact content commitments and incorporated into an independently verifiable transparency history.

Governance is expressed through typed consent, authority, responsibility and lifecycle records. Safety-relevant evidence can alter subsequent trust decisions without rewriting historical claims. Independent verifiers evaluate record integrity, authenticated history, issuer authority, consent, evidence and applicable policy to produce reproducible decisions.

The protocol supports independent checkpoint verification, external historical anchoring through Bitcoin and a decentralized network architecture without a native currency.

The fundamental OpenAIIS operation is the transformation of signed digital claims and their governance history into independently verifiable trust evidence.

1. Protocol Principles

OpenAIIS defines five interconnected properties.

Provenance. Every registered claim binds a particular issuer to a precise digital assertion and cryptographic content commitment.

Governance. Identity, consent, responsibility and delegation are evaluated through explicit, typed and authenticated relationships.

Accountability. Consequential digital actions may be linked to their executing actor, controlling operator, responsible principal and supporting evidence.

Safety. Authenticated changes in authority, revocation, disputes and security evidence can be incorporated into subsequent verification decisions.

Independent verification. Records, transparency proofs and trust decisions can be evaluated separately from the service through which the records were produced.

The protocol separates three distinct operations:

1. Registration establishes an authenticated claim and commits it to verifiable history.
2. Resolution evaluates related claims, authority, consent and lifecycle events.
3. Decision applies the relying party's policy to the resolved evidence.

Successful registration alone never grants permission to perform an action.

1.1. Normative terminology

The terms **MUST**, **MUST NOT**, **SHOULD** and **MAY** describe protocol conformance requirements. A cryptographic requirement is satisfied through deterministic verification. An authority requirement is satisfied through an authenticated and explicitly trusted relationship. A policy requirement is satisfied when the verifier's configured conditions are met. These evaluations are recorded separately.

2. Actors and Trust Domains

The OpenAIIS model distinguishes the following actors.

Actor	Protocol responsibility
Issuer	Signs a typed digital assertion
Subject	Person, artifact, agent or entity identified by a claim
Principal	Accepts responsibility or delegates authority
Operator	Controls or administers a delegated system
Agent	Executes an operation under authenticated authority
Consent signer	Authorizes a defined identity-related intent
Evidence issuer	Attests to committed supporting evidence
Transparency operator	Maintains an append-only authenticated record history
Witness	Independently observes and authenticates checkpoints
Node	Validates and replicates registry history under network rules
Relying party	Evaluates evidence and makes a policy decision

Trust is specific to an actor, key, scope and intended operation. An issuer trusted for research credentials does not automatically possess authority over software publication. An agent authorized to publish software does not automatically possess authority to change its own delegation.

Each relying party **MUST** establish its trusted issuer keys, authority mappings, acceptable evidence and relevant verification policy independently.

3. Universal Claim Representation

The central protocol object is the Universal Signed Claim, represented as a canonical signed registry record and an associated typed payload. A claim contains the following fields:

Field	Meaning
profile	Exact signed-record format identifier
record_type	Typed claim category
subject_id	Identifier of the claimed subject
issuer_id	Identifier of the signing issuer
issuer_key_id	Cryptographic signing-key identifier
content_sha256	SHA-256 commitment to the content
payload_schema	Identifier of the typed payload schema
payload_sha256	SHA-256 commitment to the canonical typed payload
authority_refs	References to supporting authority records
evidence_refs	References to supporting evidence records
consent_refs	References to authenticated consent records
related_record_refs	References to lifecycle or related records
claimed_at	Issuer-declared timestamp
signature	Ed25519 signature authenticating the claim body

All four reference arrays are sorted, unique and bounded. A reference consists of an authenticated SHA-256 record commitment. References cannot independently confer authority; their targets must be retrieved and evaluated under the rules applicable to their types.

3.1. Canonical encoding

The core JSON profile uses sorted object keys, compact JSON separators, ASCII-safe canonical serialization, no duplicate object keys, no additional or unknown signed fields, strictly validated identifiers, digests and numeric values, and a maximum signed-record size of 65,536 bytes.

A verifier **MUST** reject noncanonical representations rather than interpreting alternative encodings of the same apparent data as equivalent signed records. The canonical record body excludes the signature field.

3.2. Cryptographic signing

The core profile uses Ed25519 signatures and SHA-256 commitments. For claim body C:

```
M = DOMAIN || CanonicalJSON(C)
signature = Ed25519.Sign(issuer_private_key, M)
```

The signature is encoded as canonical Base64. The signing domain is:

OPENAIIS/UNIVERSAL-REGISTRATION/v1\0

The issuer key identifier is:

```
issuer_key_id = "sha256:" || HEX(SHA256(raw_public_key))
```

The verifier **MUST** validate the signature against an independently pinned issuer public key and **MUST** compare the expected issuer identifier with the signed issuer_id whenever issuer identity is required by the decision.

The signing domain is a sequence of ASCII bytes terminated by a single NUL byte. The precise domain, record profile and signed bytes jointly determine verification. A verifier **MUST** select the domain from the authenticated version-specific record profile and **MUST** reject cross-profile substitution. Existing signed records retain their original verification domain and canonical bytes.

3.3. Content commitment

For original bytes B:

```
content_sha256 = "sha256:" || HEX(SHA256(B))
```

When content is supplied, a verifier **MUST** recompute the digest and compare it to the signed commitment. When original content is unavailable, the claim's cryptographic signature and registered history may still be verified, while content integrity remains a separate unevaluated property. This allows large or confidential artifacts to remain outside the common registry.

For a private artifact_registration using profile openaiis-artifact_registration-private-v1, the commitment is a salted, domain-separated digest:

```
content_sha256 = "sha256:" || HEX(SHA256(
    "OPENAIIS/PRIVATE-ARTIFACT-COMMITMENT/v1\0" || salt || B
))
```

In this private profile, salt is exactly 32 bytes and B is the original content. The signed payload identifies visibility as blinded_digest. The salt and original content can remain with the holder until intentional disclosure. A verifier claiming content-integrity verification **MUST** obtain those bytes and recompute the profile-specific commitment; comparison with ordinary SHA-256 of B is not equivalent.

3.4. Typed payload commitment

Every typed claim has a canonical payload P:

```
payload_sha256 = "sha256:" || HEX(SHA256(P))
```

The payload identifies the specific meaning and constraints of the record. Validation requires the declared schema to match the record type, the payload digest to match the supplied bytes, and all required roles, fields and references to satisfy the corresponding profile. Unknown record types are not automatically granted recognized governance semantics.

The declared `payload_schema`, the `record_type`, and the embedded payload profile form a version-specific binding. The validator **MUST** check the exact combination, including canonical payload bytes, required and forbidden references, issuer and subject roles, and any type-specific relationships. The accepted schema generation is determined by signed material, not by a caller-controlled downgrade switch.

3.5. Signature and proof version integrity

A signed-record profile, typed-payload schema, inclusion-receipt profile and portable Bitcoin-bundle schema have separate purposes. Their exact identifiers **MUST** be checked at the relevant verification layer. A record issued under `openaiis-universal-registry-v1` uses receipt profile `openaiis-universal-inclusion-v1`; its portable Bitcoin proof uses `openaiis-universal-bitcoin-bundle-v1`.

Historic signed records remain independently verifiable with their original declared signature and proof encodings. Historic signatures **MUST NOT** be assigned a new profile name, nor can an old checkpoint or Bitcoin proof be reassociated with newly signed bytes. A new registration creates a new commitment and history entry.

4. Universal Claim Types

OpenAIIS defines a common registry history accommodating distinct classes of digital claims. The core vocabulary contains seventeen signed-record types.

Record type	Typed payload schema	Protocol purpose
<code>artifact_registration</code>	<code>schema:artifact-registration-v1</code>	Register an exact artifact commitment
<code>invention_claim</code>	<code>schema:invention-claim-v1</code>	Record an invention or prior-art assertion
<code>research_work</code>	<code>schema:research-work-v1</code>	Commit research output and its declared context
<code>software_release</code>	<code>schema:company-release-v1</code>	Bind a software release to signed authority and evidence
<code>ai_generated_code</code>	<code>schema:ai-generated-code-v1</code>	Record declared AI-generation context
<code>media_work</code>	<code>schema:media-work-v1</code>	Bind media to provenance and rights assertions
<code>identity_attestation</code>	<code>schema:identity-attestation-v1</code>	Issue a consent-governed identity assertion
<code>artifact_attestation</code>	<code>schema:artifact-attestation-v1</code>	Issue an artifact-related attestation
<code>identity_consent</code>	<code>schema:identity-consent-v1</code>	Authenticate the subject's precise consent intent
<code>authority_statement</code>	<code>schema:authority-statement-v1</code>	Register authenticated governance or delegation
<code>evidence_attestation</code>	<code>schema:company-evidence-v1</code>	Bind evidence assertions to committed bytes
<code>issuer_revocation</code>	<code>schema:attestation-lifecycle-v1</code>	Record an authorized issuer revocation
<code>subject_withdrawal</code>	<code>schema:attestation-lifecycle-v1</code>	Record a subject's authenticated withdrawal
<code>dispute_raised</code>	<code>schema:attestation-lifecycle-v1</code>	Register a signed dispute concerning a target
<code>dispute_settled</code>	<code>schema:attestation-lifecycle-v1</code>	Register an authorized resolution of a dispute
<code>artifact_transfer_offer</code>	<code>schema:attestation-lifecycle-v1</code>	Record an attestation-holder's transfer offer
<code>artifact_transfer_accept</code>	<code>schema:attestation-lifecycle-v1</code>	Record the recipient's acceptance of that offer

Each type **MUST** declare permitted issuer and subject roles, exact payload fields, required references, prohibited ref-

erences and applicable lifecycle rules. The core type-catalogue identifier is `openaiis-universal-type-lock-v1`. Type admission is distinct from the decision that a record's real-world assertion is true.

An `invention_claim` records an authenticated invention assertion, while independent legal and evidentiary processes determine patentability or legal priority. Similarly, `ai_generated_code` records declared generation provenance; verification of the claimed generator may require additional independent evidence.

4.1. Extensibility

New claim types are introduced through explicit versioned definitions. A type definition **MUST** identify its unique record-type identifier and payload schema; accepted issuer and subject roles; required evidence; authorization and lifecycle rules; precise verification semantics; and resource and privacy constraints.

A generic signed JSON payload **MUST NOT** acquire trusted governance or safety semantics without an explicit registered type definition. This ensures that extensibility does not weaken the meaning of existing claims.

4.2. Extension-type profiles

The explicit extension catalogue identified by `openaiis-universal-type-extensions-v2` adds the following typed declarations:

Record type	Typed payload schema	Governance meaning
<code>artifact_transfer_cancel</code>	<code>schema:attestation-lifecycle-v1</code>	Authenticated cancellation of a specifically referenced transfer offer
<code>company_release_authorization</code>	<code>schema:company-signed-release-approval-v1</code>	Signed responsible-principal approval bound to an exact software release

An explicitly selected transfer-offer extension also admits the `expires_at_tree_size` field under its dedicated versioned validation profile. Core and extension admission retain separate authorization rules. No extension implicitly enlarges an existing claim's authority.

5. Trust, Authority and Delegation

OpenAIIS expresses authority through independently signed and verifiable relationships. An authority statement identifies the issuing actor, its subject, permitted operation, scope and applicable conditions. A verifier resolves these relationships against authenticated history.

For delegated operation A, the verifier must establish:

`Authorized(A) = TrustedPrincipal AND ValidDelegation AND ScopeMatch AND CurrentAuthority AND AuthenticatedRequest`

Where responsibility is required, authorization additionally depends on an authenticated acceptance of responsibility. No element can be inferred solely from possession of an agent signing key.

5.1. Scoped authorization

Delegations bind specific actors to specific operations. Scope may include an authority domain, action, resource, environment and validity interval. A request is authorized only when the signed delegation covers its precise operation at the evaluation time.

An authorization for `PUBLISH_SOFTWARE` in one project environment cannot be interpreted as authority to publish unrelated software or to perform an arbitrary external operation.

5.2. Responsibility acceptance

An operator binding establishes which operator controls a given agent. A separate responsibility statement establishes which principal accepts responsibility for that agent within the declared relationship. A delegation statement establishes the actions the principal permits.

A verifier **MUST** evaluate the authenticity and current validity of all required relationships. An agent cannot grant itself the organizational authority needed to validate its own consequential actions.

5.3. Revocation and recovery

Authority must be evaluated against the applicable lifecycle history. A compromised or retired key can be associated with authenticated revocation or recovery actions under a configured trust policy.

Recovery authorities are explicitly trusted principals or keys; a new, self-declared key cannot automatically replace a revoked principal. Revocation history remains preserved so that prior signatures and their original contexts can still be examined.

5.4. Signed approval and authenticated ordering

A company-release approval identifies the responsible principal, agent, release record, code commitment, authority scope, delegated-authority reference and intended action. Its signature and inclusion are evaluated independently of the agent's software-release signature. Where the approval-before-release policy applies, the approval's authenticated registry position **MUST** precede that of the release it authorizes.

Delegation validity is resolved from authenticated authority statements and the governing history. A signed `claimed_at` timestamp is an issuer assertion; log position and independently supported historical time are separate evidence. The universal `authority_statement` payload can carry an independently signed authority statement whose original inner signature format is verified without reinterpretation.

6. Consent-Governed Identity Attestations

Identity assertions involve the issuer and the identity subject. OpenAIIS therefore requires independent consent when issuing a consent-governed identity attestation.

The consent signer **MUST** be the authenticated controller of the relevant subject under the relying party's trust configuration.

6.1. Consent-intent binding

The subject signs an exact attestation intent. The intent is derived from the intended issuer registration, including its issuer, subject, content commitment, payload and other signed fields, with consent references excluded to avoid a circular signature dependency.

The consent commitment uses a separate domain:

OPENAIIS/IDENTITY-CONSENT/v1\0

The consent record commits to the resulting intent digest. The identity-attestation record then references the subject's signed consent record. The intent binds the proposed issuer-signed attestation fields while excluding its `consent_refs` to avoid circular commitment. The consent digest is computed using the signing domain associated with the attestation's signed schema version; historic consent records retain their exact original commitment semantics.

A verifier **MUST** establish that the consent record was signed by the trusted subject controller, its subject matches the attestation subject, its committed content matches the attestation, its intent digest matches the exact intended issuer statement, its inclusion precedes or coincides with the attestation under the accepted history rules, and the attestation identifies the corresponding consent record.

An arbitrary consent reference, even if well formed, does not satisfy these conditions.

6.2. Identity lifecycle

The issuer may revoke an attestation through an authorized issuer or recovery action. The subject may withdraw through a separate authenticated subject action. A dispute can be raised and resolved through the configured dispute-governance rules. The passage of time does not automatically revoke an identity attestation; lifecycle changes are governed by authenticated events and applicable policy.

These events do not erase earlier historical records. They determine the status returned for a specified attestation at a particular evaluation point. Identity attestations are bound to their subject and are not transferable as property.

6.3. Authenticated lifecycle resolution

The lifecycle resolver evaluates an attestation under a specified trusted checkpoint, issuer and subject-controller mappings, authenticated event records, and applicable policy. An affirmative current-state decision requires an independently identified latest checkpoint and a complete relevant history. The resolver distinguishes ALLOW, DENY and REVIEW based on authenticated state and the sufficiency of evidence.

Lifecycle version 3 resolves transfers without granting validity through unrelated growth of the registry. Revocation, withdrawal, dispute and transfer are type-specific changes bound to their target and authorized signers. The same events can be evaluated with authenticated role-scoped lifecycle-index proofs described in Section 11.4.

7. Artifact Attestations and Transfer

An artifact attestation may designate a holder whose state is governed by a signed lifecycle. The current holder is resolved from the original attestation and subsequent valid transfer events.

A transfer requires two distinct authenticated declarations.

Transfer offer: The current holder signs an offer identifying the target attestation, current holder, intended recipient and relevant predecessor state.

Transfer acceptance: The recipient signs acceptance of that specific offer.

The resolver MUST validate the present holder, offer sequence, distinct recipient, referenced offer, replay protection and applicable revocation state.

An unsigned offer, unmatched acceptance or unilateral issuer declaration cannot complete the transfer. A transfer changes the attestation's resolved holder according to the applicable protocol rules. Its existence does not independently transfer legal ownership of an underlying physical or intellectual asset.

7.1. Offer cancellation and replay resistance

An authorized holder can cancel a referenced transfer offer through a signed `artifact_transfer_cancel` event admitted under its extension profile. Acceptance is evaluated against the actual offer, its intended recipient, predecessor state, cancellation history and authenticated order. A cancelled, replayed, conflicting or incorrectly referenced offer cannot establish a completed transfer.

Identity attestations remain non-transferable. The artifact-transfer lifecycle applies to the attestation's resolved holder and does not independently establish legal conveyance of the underlying intellectual property.

8. Disputes and Governance State

Disputes are first-class authenticated lifecycle records. A dispute identifies a target and dispute identifier and is signed by an actor authorized under the configured governance relationship.

An authorized settlement references the original dispute and records a recognized resolution. The trust policy determines which actors may raise disputes and which independent arbiters may resolve them.

For the defined lifecycle profile, the arbiter must be independent of the attestation issuer, current holder and configured disputant set. Active disputes can cause a verifier to return REVIEW until the relevant conditions are resolved.

A settled dispute remains in the historical record. An issuer revocation, subject withdrawal, open dispute and completed transfer produce distinct states. They MUST NOT be collapsed into a generic boolean such as `verified = true`.

9. Accountable Software and Autonomous-Agent Actions

OpenAIIS connects software provenance with organizational authority. A conforming accountable-release decision verifies a chain of independently authenticated records:

Organization -> Operator -> Responsible Principal -> Agent -> Signed Release -> Signed Evidence

Each link has a separate cryptographic or governance meaning.

9.1. Signed release

A software-release claim binds organization and agent identifiers; responsible principal; project and release identifiers; authority domain and requested action; target environment; exact code commitment; and evidence-record reference and evidence commitment.

The implemented release profile uses the action `PUBLISH_SOFTWARE` and validates the permitted project/environment scope. The release signer must be the authenticated agent key expected by the relying party.

9.2. Governance chain

The verifier obtains the linked operator binding, responsibility acceptance and authority delegation. It verifies their signatures, identities, current states and relationships under the configured authority resolver. The authority records must correspond to the references carried by the release.

A valid agent signature cannot substitute for any missing authority relationship.

9.3. Evidence chain

Supporting evidence is represented by a separately signed evidence attestation. The verifier checks the expected evidence issuer, release identifier, code commitment, evidence commitment and applicable validity window.

Where the policy requires examination of the evidence bytes, the original evidence must be supplied and matched to its committed digest.

9.4. Request authentication

An executable request is separately authenticated. The verifier checks the request signer, intended audience, request body, replay protection and current delegation scope. The authorization decision is made for the exact requested operation.

A successfully signed release record therefore provides necessary provenance, while a complete authenticated authorization and evidence chain provides the basis for operational permission.

9.5. Authorization of consequential effects

An application using OpenAIIS evidence to authorize an external effect separates the policy decision from execution. The policy-enforcement point (PEP) obtains an independently authenticated operation intent binding the agent, tenant, executor, action, resource scope, method and target. The requested effect MUST correspond to the authenticated intent and applicable current delegation; client-supplied operation parameters cannot substitute for operator-authorized execution facts.

The execution authorization is scoped to the exact operation and may be issued as a durable, single-consumption grant. Replay, changed effects, cross-tenant substitution, expired authorization and reuse of a consumed grant MUST be rejected. Where delegation imposes a cumulative resource or expenditure limit, reservation and grant issuance are evaluated together as one atomic budget decision.

A signed executor outcome records the operation and its reported completion, failure or uncertainty against the authorized intent. An uncertain outcome remains distinguishable from a proven completed effect and does not confer

permission to repeat it. Decision-signing authority is evaluated against authenticated activation, rotation, retirement and revocation history. These safeguards bind operational consequences to governed responsibility without treating every transparency claim as a request for execution.

10. Verifiable Safety Evidence

Safety evaluation is a consequence of governed, authenticated information. OpenAIIS uses its signed lifecycle and governance history to establish whether an authority or attestation has been revoked, withdrawn, disputed or otherwise affected by trusted evidence.

A relying party can then re-evaluate an affected artifact or actor.

10.1. Security-relevant relationships

Safety decisions may depend on revocation of an issuer or delegated authority; withdrawal of consent; an authenticated dispute affecting an artifact; a change in responsible principal; release evidence that is no longer valid for the evaluation time; or conflict between signed claims or governance histories.

Each event is validated according to its own authorized actor and claim type.

10.2. Incident-advisory extension

A security-incident assertion follows the registered-type mechanism. Its type definition establishes the affected subject, report issuer, incident classification, supporting evidence, observation window, scope, and associated recovery or resolution records.

A verifier only recognizes the incident semantics after validating the corresponding registered profile and its issuer-authority policy. The incident model distinguishes reporting from enforcement.

A report is an attributable signed assertion. Its consequences depend on authenticated authority, supporting evidence and relying-party policy.

10.3. Trust-state reassessment

When a related lifecycle or security event appears, a relying party retrieves the authenticated event and recomputes the relevant trust state.

For a compromised release signer, the decision may change from ALLOW to DENY or REVIEW based on the recognized revocation, evaluation time and applicable policy.

New authenticated evidence can change a current trust decision without changing or destroying the historical record from which it was derived.

11. Authenticated Registry History

The registry maintains an ordered, append-only sequence of signed records. Each record is bound to its position through an authenticated Merkle construction.

The registry uses the log kind:

registry_record

A receipt includes:

Field	Meaning
profile	Receipt-format identifier
kind	Registry log kind
record_sha256	Hash of the exact signed record

Field	Meaning
index	Zero-based record position
tree_size	Authenticated tree size
root	Merkle root commitment
prev_entry_digest	Commitment to preceding entry relation
inclusion_path	Merkle authentication path

The Merkle construction binds the record hash, entry position and predecessor commitment through the log's defined leaf encoding. The verifier **MUST** use the exact registered leaf-hashing algorithm. It **MUST NOT** assume that a hash of raw record bytes alone is the Merkle leaf.

11.1. Inclusion verification

For a signed record R and receipt I, the verifier verifies the issuer signature on R, recomputes the exact record hash, validates receipt position and tree size, reconstructs the expected leaf commitment, validates the inclusion path and compares the resulting root with an independently authenticated checkpoint.

A receipt tied to one checkpoint size cannot simply be relabeled as a receipt for a later root. The inclusion path must be valid for the authenticated target tree state.

11.2. Signed checkpoints

A signed checkpoint binds origin, tree_size and root. The checkpoint follows the signed-note structure used by C2SP transparency checkpoints. Its signature is validated using an independently pinned checkpoint public key.

The checkpoint signer is separate from the issuer of the claim. A claim issuer authenticates a statement; a checkpoint signer authenticates a registry state. These signatures perform distinct functions.

11.3. Witnessing and consistency

Independent witnesses retain and compare authenticated checkpoint states. A network verification policy **SHOULD** require consistency between successive states and identify conflicting signed roots or inconsistent histories.

A witness **MUST** refuse to endorse a checkpoint that cannot be reconciled with the history previously accepted under its witness policy. Equivocation evidence remains independently auditable.

11.4. Authenticated lifecycle index

A lifecycle index enables selective verification of governance state at an authenticated registry checkpoint. The index uses a sparse Merkle structure keyed by a record reference or a subject identifier, together with an issuer signing-key identifier. For a lifecycle record, the operator derives filing keys from the logged record's related_record_refs, subject_id and issuer_key_id. The relying party independently authenticates the record signatures and roles. Both release and historical lifecycle schema versions receive their corresponding subject-index entries.

The index root is authenticated by a separately signed index checkpoint that binds the registry's origin, tree size and log root. The signed index note uses the identifier openaiis-lifecycle-index-v1; disclosed membership or non-membership evidence uses openaiis-lifecycle-index-proof-v1.

A relying party evaluating current governance state **MUST** independently establish the appropriate latest registry checkpoint and its corresponding signed index note. It **MUST** verify supplied sparse-Merkle proofs and the signatures and roles of the disclosed entries. Missing or unauthenticated proof for a required role cannot be interpreted as an absence of revocation or dispute.

The index is deterministically reconstructible from the committed registry history. An independent auditor may rebuild the index and compare its root with the signed note, making omissions and conflicting index assertions auditable. The index does not change the underlying signed claims, Merkle log or checkpoint encodings.

11.5. Durable claim registration and checkpoint consistency

An admitted signed record is durably stored as canonical bytes and appended to the registry's authenticated log. An identical record submission is idempotent; its digest identifies a single registered object. A signed checkpoint authenticates a specific tree size and root; portable inclusion receipts are derived against that exact checkpoint. Successive signed checkpoints can be related by an RFC 9162 consistency proof. A valid proof of inclusion at one tree size cannot replace a consistency proof across distinct roots.

12. Portable Verification

OpenAIIS evidence is designed to remain verifiable outside the original application or platform.

A portable proof may contain a signed universal claim, canonical typed payload, original content or authenticated reference, transparency inclusion receipt, signed checkpoint, referenced authority/consent/evidence records, relevant lifecycle history and optional historical timestamp evidence.

The verifier supplies its own issuer key bindings, checkpoint trust roots, accepted history and policy. It **MUST NOT** import authority merely because an untrusted evidence bundle names a public key or includes a favorable conclusion.

12.1. Verification procedure

A complete policy evaluation follows this sequence:

```
VERIFY(record, payload, receipt, checkpoint, context):  
verify canonical encoding  
verify issuer identity and signature  
verify typed payload and schema  
verify content commitment when required  
verify checkpoint signer and origin  
verify Merkle inclusion  
retrieve referenced records  
verify consent and authority relationships  
resolve applicable lifecycle history  
verify required evidence and evaluation windows  
apply relying-party policy  
return decision, reasons, evidence references
```

Each result reports which properties were actually evaluated. A relying party **MUST** require an adequately complete and current authenticated history before issuing a decision that depends upon the absence of later revocation, withdrawal or other governing events. Where a signed lifecycle index is used, the relying party **MUST** authenticate its independent root, log-root binding and necessary membership/non-membership proofs.

12.2. Decision semantics

ALLOW: The requested operation satisfies all applicable authenticated requirements.

DENY: A recognized rule fails or an authenticated prohibition applies.

REVIEW: Additional or conflicting information prevents an affirmative automated decision.

The result carries reasons and the evidence required to reproduce the outcome. A valid Merkle receipt alone cannot be promoted into **ALLOW**.

12.3. Registry HTTP interface

OpenAIIS defines a JSON HTTP reference interface for submitting already-signed typed claims, retrieving authenticated history, and verifying portable evidence. The transport exposes the following operations:

Method	Path	Operation
GET	/health	Service liveness
GET	/v1/capabilities	Supported profile and assurance manifest
GET	/v1/claim-types	Registered typed-claim catalogue
GET	/v1/registry/status	Registry size and authenticated root status
POST	/v1/claims	Validate and register an externally signed typed claim
GET	/v1/claims/{digest}	Retrieve the exact stored signed record by its SHA-256 digest
GET	/v1/claims/{digest}/receipt	Retrieve its inclusion receipt and signed checkpoint
POST	/v1/checkpoints	Create and persist an operator-authenticated checkpoint
GET	/v1/checkpoints/latest	Retrieve the latest signed checkpoint
GET	/v1/checkpoints/{size}	Retrieve a signed checkpoint at a specified tree size
GET	/v1/proofs/consistency?old=N&new=N	Verify the consistency relationship between two signed tree states
POST	/v1/verify/record	Verify an issuer-signed canonical record
POST	/v1/verify/typed	Verify a canonical typed payload and its signed record
POST	/v1/verify/inclusion	Independently verify a signed inclusion proof against supplied trust pins

The signature-validation API does not generate an issuer's signature. A claim is signed by its issuer before submission. For POST /v1/claims, the JSON request has exactly the fields `record_base64`, `payload_base64` and `issuer_id`. The operator verifies the pre-provisioned Ed25519 issuer key, issuer identifier, allowed claim type, signed bytes, payload commitment and type-specific admission rules before appending. The canonical signed record is persistent; disclosure of the typed payload and original content remains under the client's control.

POST /v1/checkpoints is an operator-authorized action using separately provisioned checkpoint-signing material. Its empty JSON request is authenticated with an operator bearer token. Public read and verification operations do not inherit this signing authority. Checkpoint signatures and claim signatures use distinct keys and verification procedures.

The source-distributed reference transport uses IPv4 loopback at 127.0.0.1:8082; the defined operations and evidence structures are independent of that transport address. The machine-readable API contract is OpenAPI 3.1.

12.4. Verification response semantics

A signature or inclusion-verification endpoint reports cryptographic validity **relative to its supplied or configured keys**. Caller-supplied public keys are not, by themselves, proof of issuer authority. Typed-payload verification establishes schema and commitment integrity; consent, rights, delegated authority, content truth and lifecycle state are evaluated through their respective governed proof processes.

A relying party **MUST** keep these properties distinguishable in the returned result. In particular, `CRYPTOGRAPHIC_INCLUSION_VERIFIED` is an inclusion verdict rather than an operational `ALLOW`. A checkpoint marked `SIGNED` is an authenticated registry state, not by that fact a Bitcoin confirmation.

13. Bitcoin Historical Commitments

OpenAIIS binds verifiable registry history to Bitcoin through a signed registry checkpoint. The committed object is the exact signed checkpoint note covering an authenticated set of universal claims. A single checkpoint can establish a shared historical commitment for many independently verifiable records.

The proof chain is:

```

Issuer-signed claim
-> RFC 9162 inclusion receipt
-> operator-signed checkpoint note

```

```
-> detached OpenTimestamps proof of that exact note
-> Bitcoin transaction commitment
-> mainnet block-header and confirmation evidence
```

The portable universal bundle is identified by `openaiis-universal-bitcoin-bundle-v1`. It contains the exact signed-record bytes, inclusion receipt, signed checkpoint note and detached OpenTimestamps proof. The record profile, receipt profile and bundle schema **MUST** be mutually consistent.

13.1. Historical-proof verification

A verifier **MUST** authenticate the issuer signature and expected issuer identifier, the independent checkpoint-signing key and origin, the authenticated inclusion path, the signed checkpoint note, and the OpenTimestamps commitment to that precise note's SHA-256 digest. Bitcoin historical assurance is evaluated only after the registry evidence is cryptographically valid.

Where an OpenTimestamps proof contains a Bitcoin attestation, the verifier checks the timestamp operations and transaction/merkle commitment against Bitcoin block-header evidence. It evaluates the block's proof of work, links to confirming headers, applicable difficulty information and the relying party's required confirmation depth. The reference Bitcoin profile uses six confirmations as the minimum default. Pending observations, invalid proofs and externally unverifiable observations have different meanings and **MUST** be reported distinctly.

13.2. Bitcoin assurance classes

The same historical commitment can be evaluated at distinct assurance levels without changing the underlying signed record or checkpoint:

Assurance	Cryptographic and chain evidence
Public-source header verification	Locally checked checkpoint and OpenTimestamps proof, Bitcoin merkle commitment, header proof of work, linked confirming headers, packaged mainnet minimum-work reference, with active-chain selection reported by the chosen public source
Independently pinned anchor	Public-source verification plus an independently established Bitcoin anchor block identity
Independently pinned confirming chain	Independently established anchor and confirming-header identity together with verified linkage and applicable local proof checks

Verification output **MUST** identify the assurance class actually applied. A public provider's assertion of the active chain is distinct from independent full-chain consensus validation. The work-reference freshness and consistency policy is validated as part of the corresponding public-source assurance class. A verifier **MUST NOT** silently promote one assurance class into another.

13.3. Historical time and trust state

The protocol distinguishes issuer-declared `claimed_at`, registry ordering, checkpoint signing and externally authenticated Bitcoin history. Bitcoin confirmation supports a historical bound on the committed checkpoint and every record correctly proven beneath that checkpoint. It does not replace the actor authorization, evidence evaluation, consent or lifecycle state applicable to a claim.

A record's later revocation or dispute may change its present governance status without changing the cryptographic validity of the historical Bitcoin-anchored assertion.

14. Decentralized Network and Consensus

The OpenAIIS network model supports replication and independent verification of trust history across independently operated nodes without making any one infrastructure operator the authority over the meaning of its claims. The decentralized network must preserve the integrity of the same signed claims and governance relationships defined by this protocol.

14.1. Node responsibilities

A conforming network node must validate canonical records and signatures; apply defined admission and resource-control rules; validate deterministic record ordering and block structure; maintain or verify authenticated historical commitments; reject invalid consensus transitions; support synchronization and independent history validation; and preserve verifiable evidence of conflicting histories and recovery decisions.

Node acceptance of a record establishes its place in the network history, not the truth of its contents.

14.2. Proof-of-work ordering

The no-native-token proof-of-work architecture provides a mechanism for permissionless participation and independently verifiable historical ordering.

An independently verifiable consensus profile **MUST** deterministically define block serialization, parent commitments, valid work targets, difficulty adjustment, fork choice, reorganization limits, timestamps and synchronization.

All independently validating nodes must reach compatible conclusions when presented with the same valid history and consensus context. Nodes **MUST** evaluate the exact serialization, target, transition and history rules of the declared consensus profile; a verification result under one profile is not transferable to an incompatible profile.

14.3. Admission and abuse prevention

Admission controls must account for hostile submission volume, expensive parsing, invalid signatures, replayed claims and storage exhaustion. Open registration must be balanced with bounded validation costs and rate-resistant admission mechanisms.

Client-side computational admission proofs and network resource budgets may be used to reduce abuse without assigning a native token to the protocol. Permission to submit a claim remains distinct from permission to issue a trusted attestation.

14.4. Economic model

OpenAIIS separates registry participation from premium attestation and service relationships. The network architecture accommodates unverified public claims and differentiated attestations supported by recognized issuers.

Fees for services or accepted network capacity can be denominated in existing assets rather than a protocol-native currency. Payment or service fees do not themselves establish trusted identity, ownership or authority.

Consensus participation, claim admission and service economics must remain defined by explicit rules rather than implicit privileges of an infrastructure provider.

15. Interoperability

OpenAIIS uses openly specified cryptographic structures and permits compatibility with established transparency and signed-statement ecosystems. Its universal signed-claim semantics remain distinct from the encoding of an external format.

A COSE-signed statement may carry a corresponding assertion under a separately defined interoperability profile. A SCITT receipt may prove transparency inclusion in a compatible external service. A C2PA media assertion may be referenced as evidence. A Sigstore software record may provide supplementary supply-chain evidence.

Conversion or cross-registration requires explicit preservation and verification of issuer identity, committed content, statement semantics and historical receipts. Changing a profile identifier cannot upgrade the assurance of an earlier record.

Independent verifiers must identify the exact supported encoding and receipt format before evaluating it.

15.1. Format-specific interoperability

The universal registry uses its explicitly identified canonical JSON/Ed25519 signed-claim format, RFC 9162 Merkle proofs and C2SP-compatible signed checkpoint notes. A separately identified COSE/SCITT interoperability profile uses its own signature, statement and receipt verification rules, including format-specific trust roots. SCITT verifiable data structures and receipt semantics are defined through their selected profiles rather than inferred from the OpenAIIS JSON record.

A verifier **MUST** identify the actual profile, encoding, cryptographic algorithm, receipt structure, origin and keys before granting any interoperability-derived assurance. Interoperability preserves the signed claim's evidence relationships; it cannot manufacture authority, consent or historical time by converting representations.

16. Privacy, Data Protection and Evidence Handling

The registry primarily stores commitments and authenticated statements. Original evidence may remain under the control of the issuer, subject or an independently selected evidence custodian.

A verifier must be able to establish whether it has inspected the original committed bytes or only the public commitment. Sensitive identifiers and predictable-content hashes require appropriate privacy handling.

Applications may use salted content commitments or controlled disclosure where such mechanisms are explicitly supported by their content-verification profile. Security and governance history must remain auditable while minimizing unnecessary exposure of confidential source material.

16.1. Deliberate content disclosure

The signed registry record, including its public metadata and cryptographic commitments, is retrievable by its record digest. The registry HTTP reference validates the typed payload at admission and preserves the exact signed record, while the client retains the typed payload and original content for later selective disclosure. A relying party checks an undisclosed payload or artifact only to the extent supported by its signed commitment and available evidence.

Private artifact profiles use independently generated salts and domain-separated commitments to reduce straightforward discovery of predictable confidential content. Disclosing a salt enables verification against the exact committed bytes; a disclosure policy determines whether and to whom the underlying content is made available. Consent-related records and lifecycle history remain authenticated without requiring blanket publication of private evidence.

17. Conformance and Independent Acceptance

Protocol conformance is defined by deterministic agreement on cryptographic encodings, signed statements, governance semantics and verified trust inputs. Conformance to one profile **MUST NOT** be interpreted as conformance to another profile merely because their statements describe the same apparent subject.

A conforming OpenAIIS verifier **MUST** apply the requirements of the declared profile:

1. **Canonical record validity.** Reject additional or duplicated signed fields, noncanonical bytes, malformed identifiers and invalid signature encodings.
2. **Versioned signatures.** Verify the Ed25519 signature against the exact version-specific signing domain and an independently established issuer key; reject profile/domain substitution.
3. **Typed-claim integrity.** Bind the signed `record_type`, payload schema, canonical payload bytes, issuer and subject roles, and required references. Reject unregistered or semantically mismatched types.
4. **Content commitment.** Apply the correct ordinary or private domain-separated content algorithm to disclosed bytes; distinguish a verified commitment from evidence that has not been inspected.

5. **Registry history.** Authenticate the signed checkpoint origin, operator key, tree size and Merkle root; validate the precise inclusion and any applicable consistency proof.
6. **Consent and governance.** Validate consent signers and exact consent intents, and independently resolve delegated authority, responsibility, revocation and recovery from authenticated history.
7. **Lifecycle and disputes.** Evaluate revocation, withdrawal, disputes, transfer offers, cancellations and acceptance in authenticated order, with the correct authorized actors and independent latest-state evidence.
8. **Lifecycle-index integrity.** Authenticate the signed index note and its log-root binding; validate required membership/non-membership proofs and the signatures of relevant indexed records.
9. **Accountable operations.** Require the exact signed release, applicable principal approval, delegation, evidence and request-authentication chain before returning an affirmative operational authorization decision.
10. **Historical anchoring.** Bind a Bitcoin timestamp proof to the exact signed checkpoint bytes, verify the Bitcoin evidence required by the selected assurance class, and report that class without substitution.
11. **Independent trust.** Keep verification of a cryptographic assertion separate from independent trust in the issuer, its authority, the truth of evidence and the relying party's final policy decision.

Given identical authenticated evidence, trust roots, profile definitions and evaluation policy, independent verifiers MUST reach the same protocol-level result and reasons. Invalid signatures, incomplete required histories and contradictory governed evidence MUST NOT yield automatic authorization.

18. Protocol Conclusion

OpenAIIS establishes a common cryptographic foundation for digital claims whose interpretation depends on authenticated authority, consent, responsibility and changing evidence.

Its architecture unifies several operations that have traditionally been handled independently. Claims establish provenance. Governance determines valid authority. Responsibility binds authorized actors to consequential activities. Lifecycle and safety evidence update trust state. Authenticated registry history makes those relationships independently inspectable.

The protocol's central invariant is:

A claim's history remains verifiable, while its current authority and trustworthiness are determined from authenticated evidence and governing rules.

This separation makes it possible for a universal registry to serve creators, research institutions, software publishers, enterprises, autonomous systems and security infrastructure while retaining meaningful distinctions among their trust requirements.

OpenAIIS is designed to make digital trust independently examinable, accountable and responsive to evidence across the Internet.

Appendix A – Exact Core Wire Identifiers

The following identifiers denote distinct versioned signed or authenticated protocol artifacts:

Object	Identifier
Signed universal record	openaiis-universal-registry-v1
Universal record signing domain (ASCII + NUL)	OPENAIIS/UNIVERSAL-REGISTRATION/v1\0
Inclusion receipt	openaiis-universal-inclusion-v1
Registry log kind	registry_record
Core 17-type catalogue	openaiis-universal-type-lock-v1
Explicit extension type catalogue	openaiis-universal-type-extensions-v2
Identity consent-intent signing domain (ASCII + NUL)	OPENAIIS/IDENTITY-CONSENT/v1\0

Object	Identifier
Private artifact registration payload profile	openaiis-artifact_registration-private-v1
Private artifact commitment domain (ASCII + NUL)	OPENAIIS/PRIVATE-ARTIFACT-COMMITMENT/v1\0
Authenticated lifecycle index	openaiis-lifecycle-index-v1
Lifecycle-index proof	openaiis-lifecycle-index-proof-v1
Universal Bitcoin proof bundle	openaiis-universal-bitcoin-bundle-v1

The displayed \0 notation denotes **one zero byte**, not two literal printable characters. Signed and hashed representations use the exact protocol bytes.

Record-type schemas are enumerated in Section 4. Identity, artifact, consent, lifecycle and accountable-release semantics depend on these exact schema bindings. Historical profiles remain verifiable by their original immutable encodings and signing domains; profile renaming or re-serialization does not transfer historical authentication to different bytes.

Appendix B — Core Verifier Invariants

A verifier must preserve the following properties:

1. An issuer public key is pinned independently of the claim being evaluated.
2. A checkpoint key and expected origin are pinned independently of the receipt.
3. Exact canonical bytes determine cryptographic signature validity.
4. The type schema determines the permitted interpretation of a claim.
5. Required consent is independently signed and bound to exact intent.
6. Authority is resolved from authenticated delegation and lifecycle history.
7. Responsibility is not inferred from agent key possession.
8. Revocation and withdrawal are different authorized events.
9. An artifact transfer requires a valid holder offer and recipient acceptance.
10. An identity attestation cannot become transferable through artifact-transfer rules.
11. A dispute does not erase its target record.
12. Missing or contradictory required evidence cannot result in automatic authorization.
13. A valid receipt is evaluated against the exact authenticated checkpoint it references.
14. Historical inclusion cannot establish absence of later governing events.
15. Bitcoin historical evidence is distinct from issuer-declared and registry-observed time.
16. Consensus ordering and real-world trust assessment remain distinct operations.
17. Safety decisions must be reproducible from authenticated evidence and configured policy.
18. A signature is evaluated only under the domain selected by its exact authenticated record profile.
19. A typed-payload schema and its embedded profile must agree with the record type and signed generation.
20. A lifecycle index note is independently authenticated and linked to the registry checkpoint it summarizes.
21. Role-scoped absence proofs cannot replace independently established current registry and index heads.
22. Historical Bitcoin assurance applies to the exact signed checkpoint bytes authenticated in its detached timestamp proof, not to a merely similar or later checkpoint.
23. Verifier conclusions preserve the distinction among issuer signature integrity, content integrity, governance authority, historic registry inclusion and Bitcoin-chain assurance.

These invariants define the separation between record authenticity, governed authority and operational trust that gives OpenAIIS its universal applicability.

OpenAIIIS

CREATIVE COMMONS LICENSING NOTICE

OpenAIIIS Protocol - Version 5.0

Copyright (c) 2026 Tarun Sethi

This publication is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0).

You may share and adapt it, including for commercial purposes, provided you follow the license conditions.

Attribution must identify the work and its version, credit the author, link to the license, and indicate whether changes were made.

License: <https://creativecommons.org/licenses/by/4.0/>

Legal terms: <https://creativecommons.org/licenses/by/4.0/legalcode.en>

The license covers copyright permissions in this document.

It does not grant patent rights or trademark rights.

Software is separately licensed under the Apache License 2.0.

Independent compatible implementations are welcomed; relevant patent, trademark and other rights remain separate from CC BY.

Official project: <https://openaiis.org>

This notice was added as a separate final page in October 2026.

The original publication pages remain unchanged.