

Axiom Zero

A Cryptographically Committed, Exact-Arithmetic Verification Kernel for Auditable Decision Certification

Technical Preprint - Implementation Snapshot v7.9.9

Project: Axiom Zero

Author: [to be completed]

Date: October 2026

Document status: Technical preprint / implementation evidence

Artifact	axiomzero_v7.9.9.zip
Artifact SHA-256	75002dda44e4db784217eddea22744417c86d98fc5ba03815041bf10e5e988b1
Verification profile SHA-256	9e538e9e4cb6e74773ea0cbaa563feefcc40b7e919afd88a8bac4482e042efa7
Paper scope	Documents the implemented v7.9.9 system and its evidence; not an external certification.

Research integrity statement. This document distinguishes implemented mechanisms, regression evidence, independently reproduced observations, design assumptions, and unresolved audit findings. It intentionally does not claim peer review, formal proof of the entire software stack, or platform-level remote attestation.

Abstract

Axiom Zero is a verification-oriented software kernel designed to convert structured project evidence into a deterministic, cryptographically committed certificate whose decision can be independently reproduced. The v7.9.9 implementation combines canonical JSON, a 13-leaf salted RFC 6962-style Merkle commitment, Ed25519 signatures, a signed evidence registry, exact rational arithmetic over $\mathbb{Q}$ for rank and cost-related calculations, 50-digit interval arithmetic for outward numerical bounds, policy witnesses, source-document citations, replay-resistant one-shot authorization, and signed release metadata. The verifier re-computes the committed decision rather than trusting a prover-supplied verdict. The implementation also contains explicit denial-of-service limits, certificate freshness checks, evidence-byte re-hashing, build/profile binding, runtime known-answer tests, and a regression corpus that evolved through repeated adversarial referee rounds.

This paper documents the construction as implemented in the supplied v7.9.9 artifact, provides the mathematical and cryptographic model, summarizes the implementation and test evidence, and states the security boundary precisely. It also records the current limitations identified by adversarial review: service-level trust-root injection remains an open boundary issue in v7.9.9; symlink targets are not fully bound by the filesystem inventory; image-digest enforcement is a platform responsibility; and release-key file permissions require hardening. Accordingly, the paper demonstrates what has been built and what properties are mechanically checked, but does not present v7.9.9 as externally certified or as a complete proof of all deployment assumptions.

Keywords: verifiable computation, audit certificate, exact arithmetic, interval arithmetic, Merkle tree, Ed25519, provenance, reproducibility, policy verification, evidence registry, replay protection, supply-chain integrity.

Ελληνική σύνοψη

Το Axiom Zero v7.9.9 είναι ένας υλοποιημένος verification kernel που παίρνει δομημένα τεχνικά/οικονομικά δεδομένα και αποδεικτικά αρχεία, εκτελεί ντετερμινιστικούς μαθηματικούς και πολιτικούς ελέγχους, δεσμεύει τα αποτελέσματα κρυπτογραφικά και παράγει certificate το οποίο μπορεί να επανελεγχθεί από ανεξάρτητο verifier. Το σύστημα δεν βασίζεται απλώς σε ένα αποτέλεσμα τύπου «PASS»: ο verifier ανακατασκευάζει το Merkle commitment, επαληθεύει υπογραφές και hashes, ξανατρέχει τις κρίσιμες μαθηματικές πράξεις, ελέγχει policy witnesses, freshness, citations και context pins, και αναπαράγει την τελική απόφαση ACCEPT/ABSTAIN/ABORT.

Το παρόν paper λειτουργεί ως τεχνικό αποδεικτικό του τι έχει κατασκευαστεί. Περιλαμβάνει fingerprint του ZIP και των βασικών source files, το verification profile hash, περιγραφή της μαθηματικής μεθόδου, το trust model και το ιστορικό adversarial testing. Δεν ισχυρίζεται ότι η έκδοση v7.9.9 έχει λάβει εξωτερική πιστοποίηση: τα ανοικτά findings του τελευταίου referee round καταγράφονται ρητά.

1. Introduction

Many decision systems produce a result without providing a reproducible chain from source evidence to that result. A human reviewer may see a report, a model, or a software-generated approval, yet be unable to establish that the same bytes, rules, numerical semantics, trust roots, and policy state would reproduce the same conclusion. Axiom Zero addresses this problem by treating a decision as a cryptographically committed object whose mathematical and evidentiary predicates are reconstructed by a verifier.

The project is not a blockchain and does not rely on a distributed consensus protocol. Its core objective is narrower: create an auditable certificate whose internal structure commits the evidence model, mathematical results, policy state, presentation state, provenance metadata, and decision semantics; require a verifier to reproduce those commitments and predicates; and fail closed when required information is missing or inconsistent.

The v7.9.9 artifact contains five principal implementation files - kernel.py, service.py, extractor.py, app.py, and make_release.py - plus a normative specification, a pinned dependency lockfile, release notes, and a regression suite. The implementation snapshot contains 3,958 lines across the five application source files and 2,155 lines in the regression test module. The artifact fingerprint is recorded in Appendix B.

2. Design Goals and Non-Goals

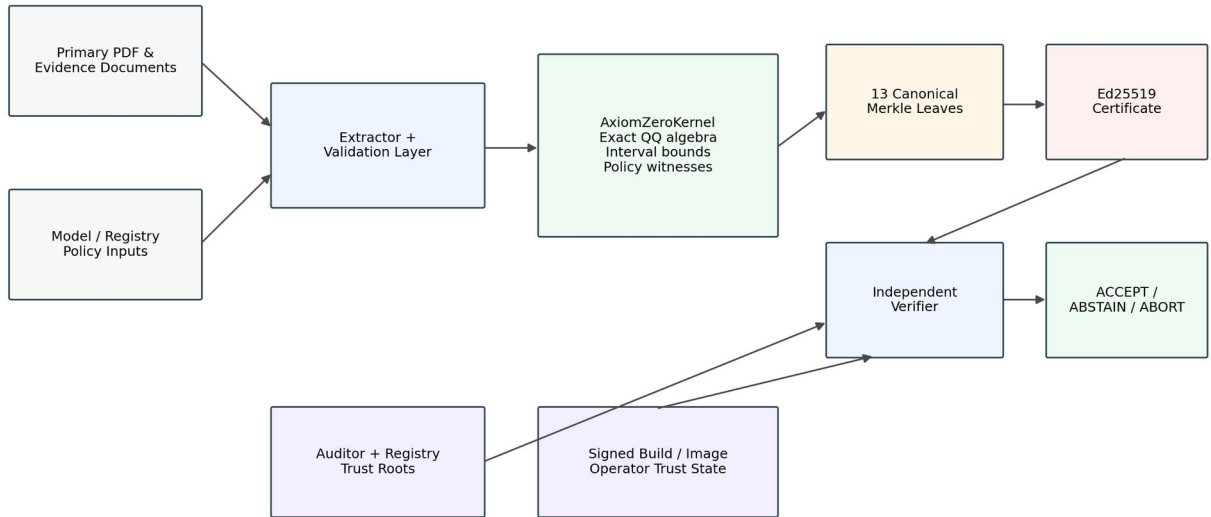
Goal	Implemented approach
------	----------------------

Deterministic representation	Strict JSON parser, canonical serialization, no duplicate keys, no NaN/Infinity, no binary floats in certificates.
Independent decision reproduction	The verifier reconstructs certificate leaves and re-runs canonical_verdict and core mathematical predicates.
Tamper evidence	SHA-256 commitments, RFC 6962-style Merkle tree, Ed25519 signatures, presentation hashes.
Source provenance	Signed evidence registry, per-document SHA-256, verifier-side byte re-hashing, page-and-quote citations.
Numerical reproducibility	Exact rational arithmetic for discrete algebraic predicates; 50-digit interval arithmetic with outward bounds for continuous quantities.
Fail-closed semantics	ABORT and ABSTAIN reasons precede ACCEPT; missing trust, evidence, coverage, freshness, or build conditions prevent certification.
Action safety	Reusable attestations never authorize actions; one-shot authorizations use serials, replay state, leases, and fencing tokens.
Supply-chain awareness	Signed release manifest, dependency/runtime inventory, build id, runtime known-answer tests, optional image digest.

Non-goals are equally important. Axiom Zero does not, by itself, prove that a quoted source sentence semantically entails a model value. It does not provide remote attestation of the Python process. It cannot enforce the immutability of the host OS or container image from inside that same process. It also does not turn the internal labels “Chronos”, “RHUT”, or “QAI” into independently established scientific theories; in this implementation those names denote software predicates and reason-code families.

3. System Architecture

Axiom Zero v7.9.9: data flow and trust boundaries



The kernel proves/reproduces software predicates; platform-enforced image identity remains an external trust condition.

Figure 1. High-level Axiom Zero v7.9.9 data flow and trust boundaries.

Inputs enter through an extraction and validation layer and are converted into a strict model. The kernel evaluates provenance, uncertainty, exact linear-algebraic constraints, interval bounds, policy coverage, retention/information predicates, and the final canonical verdict. The result is serialized into thirteen canonical leaves. Confidential leaves are salted before hashing; the Merkle root is signed by the auditor key. A separate signed registry binds business context and source evidence. The independent verifier receives the certificate plus relying-party expectations and evidence bytes, reproduces the commitment tree and the mathematical predicates, and decides whether the object is merely syntactically valid, attested, or certifiable.

The service layer adds deployment/build checks and execution semantics around the kernel primitive. This separation is deliberate: `kernel.verify_attestation` is a low-level primitive whose trust inputs are asserted by the caller, while the product/service layer is intended to derive production trust state from operator configuration. As discussed in Section 11, v7.9.9 does not yet make that separation complete for every trust root.

4. Formal Data and Mathematical Model

4.1 Node model

For each subsystem or cost node i , the implementation uses exact rational representations of five primary quantities: units u_i , unit rate r_i , uncertainty in units $\sigma_{u,i}$, uncertainty in rate $\sigma_{r,i}$, and a non-negative multiplicative loss coefficient ℓ_i . The cost factor is $f_i = 1 + \ell_i$. Production inputs are rejected if any of these quantities is negative or if the lower 3σ bounds $u_i - 3\sigma_{u,i}$ or $r_i - 3\sigma_{r,i}$ would become negative.

$$f_i = 1 + \ell_i$$

$$g_i = (f_i r_i, f_i u_i)$$

A covariance matrix Σ may be supplied. Its diagonal is required to match the exact squared σ values, and the matrix is tested for positive semidefiniteness using exact rational operations. If no covariance matrix is supplied, the implementation constructs a diagonal independence model.

4.2 Expected cost and Wick-Isserlis variance

For authenticated covariance data, the expected bilinear cost of node i includes the covariance between units and rate. The nominal aggregate cost is computed in exact rational arithmetic:

$$E[C] = \sum_i f_i \cdot (u_i r_i + \text{Cov}(U_i, R_i)).$$

The first-order covariance contribution uses the gradient vector g and the full covariance matrix:

$$V_{\text{linear}} = g^T \Sigma g.$$

Because the cost contains products $U_i R_i$, the kernel adds an exact fourth-moment correction derived from the Gaussian Wick-Isserlis identity. For node pairs i, j :

$$V_{\text{Isserlis}} = \sum_i \sum_j f_i f_j [\Sigma(u_i, u_j) \Sigma(r_i, r_j) + \Sigma(u_i, r_j) \Sigma(r_i, u_j)].$$

When the covariance registry is cryptographically authenticated, total variance is $V_{\text{linear}} + V_{\text{Isserlis}}$. Otherwise the implementation uses conservative maxima involving correlated, diagonal, and Cauchy-style bounds rather than allowing an unauthenticated covariance model to reduce uncertainty.

$$U_C = 3 \cdot \text{sqrt}(V_{\text{total}}).$$

4.3 Cost hull and interval bounds

The implementation combines two uncertainty enclosures. The first is a direct 3σ rectangular box computed from the lower and upper parameter endpoints. The second is a probabilistic ellipsoid-style interval around nominal cost using U_C . A non-negative residual evidence bound R enlarges the final hull. All conversions to continuous bounds use 50-digit interval arithmetic and outward rounding.

$$C_{\text{box}}^- = \sum_i f_i (u_i - 3\sigma_{u,i})(r_i - 3\sigma_{r,i})$$

$$C_{\text{box}}^+ = \sum_i f_i (u_i + 3\sigma_{u,i})(r_i + 3\sigma_{r,i})$$

$$C_{\text{hull}} = \text{hull}(C_{\text{box}}, [E[C] - U_C, E[C] + U_C]) \text{ expanded by residual } R.$$

The “spectral guard” compares the normalized nominal budget margin with a normalized uncertainty width:

$$\gamma_{\text{hat}} = (B - E[C] - R)/B, \quad 2\delta = 2U_C/B, \quad \text{spectral_pass} \Leftrightarrow \inf(\gamma_{\text{hat}}) > \sup(2\delta).$$

4.4 Proper-cone / Chebyshev interval predicate

The cost hull induces a normalized budget margin interval. The kernel validates a configured Chebyshev polynomial approximation and its derivative over subdivisions of the admissible domain, then constructs interval lower bounds for

both the direct margin and the Chebyshev transfer. The predicate called `PROPER_CONE` succeeds only when the domain is valid, the polynomial is admissible, and both lower bounds remain strictly above the configured approximation error. In this paper, “proper cone” refers to this implemented software predicate; it should not be read as an external proof of a physical theory.

4.5 Exact rank leakage predicate

Axiom Zero uses exact fraction RREF over $\mathbb{Q}$ to compute a forbidden-rank increment:

$$r_{\text{forb}} = \text{rank}([A; B; V]) - \text{rank}([A; B]).$$

The predicate internally labelled “Chronos” passes exactly when $r_{\text{forb}} = 0$. The RREF implementation is guarded by deterministic algebraic operation and bit-growth budgets, so pathological rational inputs fail closed based on an implementation-independent work count rather than a wall-clock timeout. This is important for prover/verifier agreement across machines of different speeds.

4.6 Policy coverage and witnesses

Policy items can be `VERIFIED`, `ATTESTED`, `BREACH`, `EXCLUDED`, or `UNRESOLVED`. A `VERIFIED` item must identify a kernel-supported witness predicate. Supported witnesses include zero leakage, proper cone, spectral guard, no information loss, no orphan sources, and all nodes backed by signed evidence. `ATTESTED` items are recorded but do not count as mechanically verified. A failed witness is an `ABORT` condition.

5. Canonical Decision Semantics

The `canonical_verdict` routine is the single source of truth for prover and verifier decision semantics. Its ordering is intentional and deterministic.

Priority	Condition class	Representative reasons
1 - ABORT	A hard invariant is violated.	Leakage/ r_{forb} , cone/Chebyshev breach, information gain, policy breach, failed policy witness.
2 - ABSTAIN	The system lacks sufficient verified basis to accept.	Orphan sources, information loss, spectral guard failure, unresolved policy items, empty production coverage universe.
3 - ACCEPT	No ABORT or ABSTAIN condition remains.	<code>ALL_PREDICATES_VERIFIED</code> .

This ordering prevents a soft uncertainty from masking a hard violation. It also makes `ACCEPT` a derived state rather than an input claim. The verifier compares the signed leaf-13 decision against the decision it reconstructs from the committed inputs and predicates.

6. Cryptographic Certificate Construction

6.1 Canonical JSON

Certificates and unredacted Merkle leaves must be canonical JSON. The implementation uses UTF-8, sorted object keys, compact separators, no duplicate keys, no NaN or Infinity, and prohibits binary floating-point values in certificates. Each unredacted leaf must round-trip through `strict_parse` and `canonical_json` to the identical byte representation. This closes parser-differential and alternative-spelling classes of attacks that were discovered in earlier audit rounds.

6.2 Thirteen-leaf Merkle commitment

The certificate commits thirteen ordered leaves. Leaf hashes use the RFC 6962 domain-separated construction $H(0x00 || \text{data})$; internal nodes use $H(0x01 || \text{left} || \text{right})$. Eight confidential leaves - indices 4 through 11 in zero-based indexing - require a 32-byte salt with the Axiom Zero leaf-salt domain prefix. Tier-1 presentation redacts exactly these leaves while retaining their commitments.

The implementation uses SHA-256. The domain-separated Merkle construction follows the hashing convention defined by RFC 6962 [1], while Axiom Zero adds mandatory per-leaf salting for selected confidential leaves.

6.3 Auditor signature

The Merkle root is signed with Ed25519 over a mandatory auditor-domain prefix concatenated with the root. The verifier pins allowed auditor public keys and reproduces the signature check. Ed25519 follows the algorithm standardized in RFC 8032 [2]. SHA-256 is specified by FIPS 180-4 [3].

6.4 Signed evidence registry

A separate Ed25519 registry signature binds project identity, buyer identity, primary-source hash, maximum budget, node order and node parameters, covariance matrix Σ , policy hash, and verification-profile hash. The registry payload is reconstructed from committed leaf data rather than trusted as a free-form registry object. This prevents a valid registry signature from being transplanted to a different project, policy, or mathematical profile without detection.

6.5 Presentation binding

Fields outside the Merkle carrier are bound through presentation hashes committed in leaf 13. Carrier fields themselves must equal normative values. This design prevents a wrapper from displaying materially different metadata around an otherwise valid Merkle root.

7. Verification, Evidence and Citation Semantics

Certification requires more than a valid auditor signature. The verifier independently checks the certificate structure, canonical leaf bytes, leaf hashes, Merkle root, auditor signature, registry signature, pinned keys, source-document hashes, evidence-document hashes, policy/profile bindings, project and buyer context, freshness, build approval, citations, and mathematical predicates. It then re-runs `canonical_verdict`.

7.1 Source-byte verification

When expected primary-source bytes are supplied, the verifier computes SHA-256 over the actual bytes and compares the digest to the certificate commitment. Evidence documents associated with nodes are likewise re-hashed. This establishes byte identity. It does not establish that a human-readable document has been semantically interpreted correctly.

7.2 Page-and-quote citations

Every node may carry a citation {document, page, quote}. The verifier extracts the cited PDF page with `pdfminer.six` (or decodes page 1 for non-PDF evidence), whitespace-normalizes the text, and searches for the committed quote. The certificate can be certified only when all required citations are independently found. The design deliberately states the limit: citation existence proves that the quoted text is present, not that a particular numerical model value is a faithful interpretation of that text.

7.3 Freshness and relying-party context

The certificate records issue and expiry times. The verifier enforces a maximum clock skew of 300 seconds and a maximum certificate lifetime of 90 days. Relying parties may additionally pin project id, buyer id, policy hash, verification profile hash, source hash/bytes, and action id.

8. Attestation and One-Shot Execution

Axiom Zero distinguishes two certificate semantics. `REUSABLE_ATTESTATION` describes a state of the world during a validity window and is explicitly incapable of authorizing an action. `ONE_SHOT_EXECUTION_AUTHORIZATION` commits an action id and a unique certificate serial. The service checks the expected action id and uses an execution ledger before invoking the external executor.

The v7.9.9 execution ledger uses `RESERVED`, `EXECUTED`, and `FAILED` state transitions, lease expiry, a stable serial-based idempotency key, and a fencing token/reservation id. The fencing token closes the stale-worker race identified in an earlier adversarial round: a worker that lost its lease cannot finalize a newer worker's reservation. Exact "once" semantics still rely on the external executor honoring the same durable idempotency key; the local ledger cannot make an arbitrary remote system transactional by itself.

9. Verification Profile, Build Identity and Trust Stack

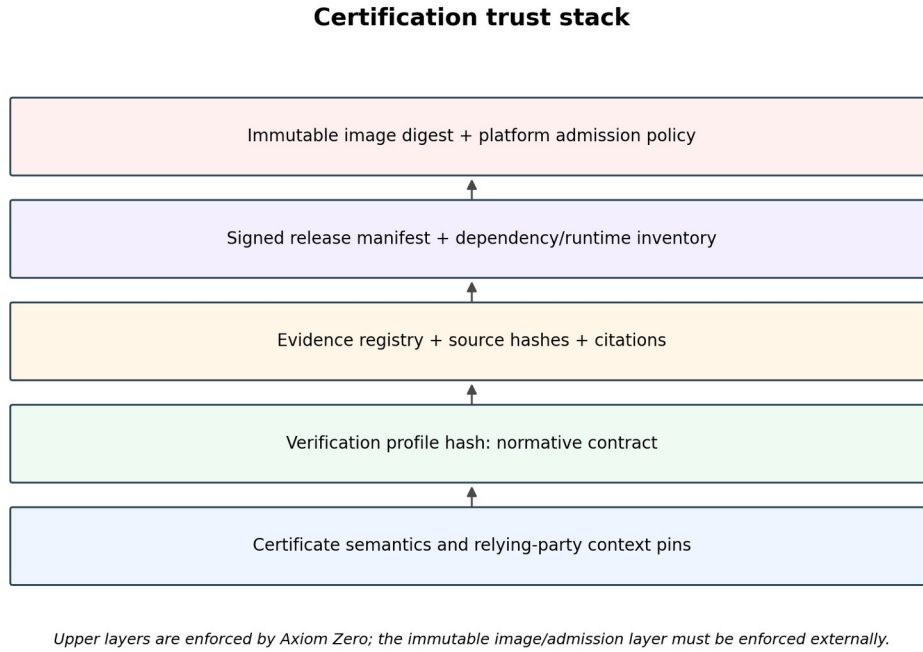


Figure 2. Separation between normative contract identity, application build identity, and platform image identity.

The verification profile hash is intentionally the fingerprint of the verification contract, not of executable code. For the supplied v7.9.9 snapshot, calling `AxiomZeroKernel.verification_profile_hash()` produced:

9e538e9e4cb6e74773ea0cbaa563feefcc40b7e919afd88a8bac4482e042efa7

The profile includes normative constants, schemas, Chebyshev parameters, exact-algebra limits, freshness limits, uncertainty semantics, and the normative specification text. A change in those normative constants changes the profile hash.

Executable identity is treated separately. v7.9.9 introduces a signed release manifest covering application files, requirements.lock, security-relevant dependencies including bytecode and unknown extra files, site hooks, interpreter identity, a kernel runtime digest, and an optional image digest. Runtime known-answer tests cover SHA-256, Ed25519, exact rank, interval arithmetic, canonical JSON, and PDF extraction. These mechanisms are defence in depth and drift detection; the code itself explicitly acknowledges that an in-process self-check cannot prove its own integrity against an attacker who controls that same process.

The authoritative deployment boundary is therefore an immutable signed image whose admission is enforced by the platform. This is a central design correction learned through adversarial rounds 27-29. The Python layer may bind an image digest into a release manifest, but it cannot enforce Kubernetes admission, cosign policy, TPM attestation, or an equivalent external control against itself.

10. Resource and Abuse Resistance

The implementation contains deterministic resource limits intended to convert malformed or adversarial inputs into controlled rejection rather than process failure. Principal limits include:

Limit	v7.9.9 value / rule
Certificate size	8 MiB maximum
Merkle leaf size	2 MiB maximum per leaf
Evidence document size	50 MiB maximum
JSON nesting	32 levels maximum
Citation quote	500 characters maximum

Citation page	1..10000
Exact-algebra bit length	8192-bit element growth limit
Exact-algebra work	3,000,000 deterministic element-update budget
Certificate lifetime	90 days maximum
Clock skew	300 seconds maximum

The explicit JSON depth check was introduced after a v7.9.6 negative control demonstrated that extreme nesting could raise an uncaught RecursionError. The current design performs size/depth validation before semantic parsing, making this a concrete example of adversarial testing changing the implementation rather than merely adding documentation.

11. Adversarial Audit Evolution

A central part of the project evidence is not that no defects were found, but that repeatedly discovered counterexamples were converted into explicit security properties and regression tests. The following table summarizes the most important recent transitions.

Round	Counterexample / issue	Resulting engineering change
R27	Verification-profile hash was incorrectly described as executable identity; source byte match did not prove semantic interpretation; execution/recovery and pre-parse limits needed hardening.	Separated contract hash from implementation identity; added citations, execution state machine, and pre-parse limits.
R28	No operator pin could silently self-approve; runtime digest missed imported crypto/PDF/service semantics; stale execution worker could finalize another reservation.	Signed build design, KATs, fail-closed deployment checks, fencing tokens.
R29	Caller could inject deployment/approved-build state; .pyc could escape dependency digest; stale digest cache could hide tampering; release could sign an environment that failed its own gate.	Removed deployment/approved-build API injection, full file inventory including bytecode, ctime-aware cache invalidation, release preflight.
R30	Caller can still supply auditor/registry trust roots at service boundary; symlink targets can change without changing inventory; platform image identity may be absent; release key permissions need hardening.	Open findings for the next patch; documented here to prevent overclaiming v7.9.9.

This history matters for publication because it demonstrates a falsifiable development process. Claims that failed an explicit mutation or adversarial test were corrected in the specification rather than retained as aspirational statements.

12. Regression and Reproducibility Evidence

The supplied test module contains 145 top-level pytest test functions. Parametrization expands these into 198 test cases in the pinned v7.9.9 environment according to the release notes. Twelve new Round-29 test functions specifically cover the mandatory regressions from the previous referee round. The suite spans parser canonicity, numerical exactness, provenance, signatures, source-byte checks, policy coverage, certificate semantics, replay safety, citations, build identity, KAT mutation tests, deployment gates, bytecode inventory, cache tampering, release preflight, and fencing.

The pinned dependencies are cryptography 50.0.1, mpmath 1.3.0, sympy 1.14.0, pdfminer.six 20260107, numpy 2.5.3, pydantic_core 2.46.5, pydantic 2.13.5, reportlab 5.0.1, and pytest 9.1.1. The release notes require PYTHONDONTWRITEBYTECODE=1 and recommend a wheelhouse for offline reproducibility.

Independent audit in a non-pinned sandbox did not reproduce the complete 198-case run because the deployment preflight correctly rejected mismatched dependency versions. This is recorded as an environment mismatch rather than silently counted as a software failure or as a successful independent reproduction. Selected Round-29 regression tests were separately exercised in a diagnostic compatible environment. A publication-quality external validation should run the exact signed image or an exact wheelhouse matching requirements.lock.

13. Evidence That the System Has Been Implemented

The claim “Axiom Zero has been built” is supported here by an inspectable artifact rather than by a conceptual architecture alone. The supplied ZIP contains implementation source, a normative specification, release tooling, a dependency lockfile, release notes, and a regression suite. The following evidence can be independently checked without trusting this paper.

Evidence item	Concrete artifact
Kernel implementation	kernel.py - 2,541 lines; exact algebra, interval bounds, policy evaluation, certificate creation and verification.
Service / deployment / execution	service.py - 681 lines; signed release state, trust loading, execution ledger, production wrapper.
Extraction and model validation	extractor.py - 458 lines; Pydantic schema, strict rational normalization, matrix construction, PDF extraction path.
User-facing application	app.py - 225 lines.
Release tooling	make_release.py - 53 lines.
Normative contract	SPEC_v7.9.9.txt - 11 normative clauses.
Regression corpus	tests/test_regression_v799.py - 2,155 lines; 145 top-level test functions, 198 cases claimed in pinned runtime.
Pinned runtime	requirements.lock with exact package versions.
Release history	CHANGES_v7.9.9.md with negative controls and migration instructions.

Appendix B lists SHA-256 hashes for the ZIP and each principal file. A third party can therefore establish that the paper refers to a specific immutable source snapshot rather than an unspecified moving codebase.

14. Security Properties Demonstrated by the Current Construction

Property	Status in v7.9.9	Evidence / qualification
Canonical certificate representation	Implemented	Strict parser + canonical JSON + binary-float prohibition + regression tests.
Merkle commitment and auditor signature	Implemented	13 leaves, domain-separated SHA-256, Ed25519 root signature.
Registry context binding	Implemented	Project/buyer/source/budget/nodes/ Σ /policy/profile included in reconstructed signed payload.
Exact rank reproduction	Implemented	Fraction RREF over $\mathbb{Q}$ with deterministic work/bit limits.
Uncertainty/cost enclosure	Implemented	Exact inputs + Wick-Isserlis variance + 50-digit interval hull and outward rounding.
Policy witness enforcement	Implemented	VERIFIED requires supported witness; failed witness aborts.
Evidence byte identity	Implemented	Verifier re-hashes primary and evidence bytes.
Citation existence	Implemented	Verifier independently searches quote on cited page.
Replay-resistant one-shot flow	Implemented	Serial/idempotency key + ledger + lease + fencing token; downstream idempotency remains required.
Application build drift detection	Implemented with known limits	Signed manifest, inventory, KATs; symlink target issue remains open in v7.9.9.
Immutable platform image enforcement	External condition	Must be provided by deployment platform; cannot be proved by the Python process itself.
External peer-reviewed certification	Not yet obtained	Requires independent audit in exact pinned/signed deployment environment.

15. Known Limitations and Open Findings

A publication intended to establish engineering credibility should state unresolved findings, not omit them. At the time of this paper, the following points remain open for v7.9.9:

1. Production service functions still accept auditor/registry trust roots as caller-supplied arguments. A malicious integration that can select these roots may cause attacker-chosen keys to be treated as trusted. The intended fix is

to load operator trust roots internally in public production APIs and reserve injectable trust roots for private test/primitives only.

2. The filesystem inventory records symlinks as link targets but does not hash the bytes of an external resolved target. A trusted executable tree should therefore reject symlinks or bind only targets provably inside the same immutable root.
3. The `image_digest` field is optional. Platform-level certification should either require a non-empty externally enforced image digest or explicitly downgrade the result to application-layer certification without platform identity.
4. `make_release.py` should create release private keys with mode 0600 independent of process umask.
5. Citation checking establishes textual existence only. Semantic mapping from text to model values remains an attestation of the registry signer unless a separate semantic extraction proof is introduced.
6. The exact mathematical predicates prove only the conditions encoded in the software model. They do not, by themselves, prove correctness of upstream business assumptions, physical models, or external data.

These limitations do not negate the implemented mechanisms. They define the boundary of the current evidence and the next engineering work required before a stronger certification claim would be justified.

16. Recommended External Validation Protocol

To turn this implementation paper into stronger third-party evidence, an external laboratory or security reviewer should receive the exact source ZIP, a signed immutable container image, the wheelhouse used to construct it, the operator release public key, and a corpus of known prior attacks. The reviewer should be instructed to seek a concrete win condition rather than merely run static analysis.

1. Attempt to obtain `is_certified=True` while violating at least one normative certification requirement.
2. Attempt to cause `execute_once` to invoke the executor without a legitimate one-shot authorization bound to the expected action id and approved trust state.
3. Mutate certificate JSON, Merkle leaves, salts, signatures, registry data, context fields, policy/profile hashes, and presentation wrapper fields.
4. Exercise parser differentials, duplicate keys, numeric respellings, depth/size exhaustion, pathological rational matrices, and concurrency races.
5. Mutate evidence documents, PDF page text, citations, dependency files, bytecode, site hooks, symlinks, release manifests, and image metadata.
6. Re-run the complete regression suite in the exact signed image and publish the image digest, test log, and auditor report.
7. Keep all known R27-R30 attacks in the corpus and require the reviewer to discover at least one previously unknown class of failure or explicitly report none found within the agreed scope.

17. Discussion

The strongest engineering idea in Axiom Zero is not any single cryptographic primitive. SHA-256, Ed25519, Merkle trees, exact rational algebra, interval arithmetic, and replay ledgers are established techniques. The contribution is their composition around a decision certificate in which evidence identity, numerical semantics, policy semantics, source provenance, presentation state, and execution meaning are explicitly separated and then rebound into a reproducible verification process.

A second notable property is the evolution of the trust model. Early versions attempted to make a verification-profile or runtime digest carry more security meaning than it could defensibly support. Adversarial mutation tests falsified those claims. The current specification explicitly separates the normative contract from the executable build and, in turn, the executable build from the externally enforced immutable image. This separation is a stronger scientific position because it identifies what can be checked inside the program and what requires an external trust anchor.

The remaining work is correspondingly narrower. Rather than adding another introspection layer, the next release should close the public service trust-root boundary, reject or fully constrain symlinks, enforce safe release-key permissions, and define a strict platform-certified profile that requires an externally verified image identity.

18. Conclusion

The supplied v7.9.9 artifact demonstrates that Axiom Zero is a concrete software system rather than a conceptual proposal. It contains a deterministic verification kernel, exact and interval mathematical machinery, cryptographic commitments and signatures, an evidence registry, source-byte and citation verification, deterministic decision semantics, replay-resistant execution support, signed build metadata, and a large regression/adversarial corpus. The artifact is fingerprinted and its normative verification profile is reproducible.

The correct conclusion is deliberately narrower than “proved secure”. What has been demonstrated is an implemented, testable verification architecture with explicit formalized predicates and a documented chain from inputs to certificate to verifier. External certification still requires closing the current Round-30 findings and running an independent audit in the exact signed deployment image. This distinction is essential: the paper is evidence of construction and verification discipline, not a substitute for third-party certification.

Appendix A - Normative Verification Specification v7.9.9

AXIOM ZERO VERIFICATION SPEC v7.9.9

- 1 Encoding: certificates and Merkle leaves are canonical JSON (CANONICAL_JSON_RULES); every unredacted leaf must equal canonical_json(strict_parse(leaf)); certificates contain no binary floats.
- 2 Merkle: RFC6962 over 13 leaves; leaf = SHA256(0x00 || [LEAF_SALT_DOMAIN_PREFIX || salt32] || bytes) with salts mandatory on CONFIDENTIAL_LEAF_INDICES; node = SHA256(0x01 || L || R); root signed with Ed25519 over AUDITOR_DOMAIN_PREFIX || root. Tier 1 redacts exactly CONFIDENTIAL_LEAF_INDICES with REDACTION_PLACEHOLDER.
- 3 Registry: Ed25519 over REGISTRY_DOMAIN_PREFIX || canonical_registry_payload rebuilt from leaves 01/04/05/06/11, binding project, buyer, primary source, budget, nodes, Sigma, policy hash and verification profile hash.
- 4 Mathematics: exact QQ for costs, variance (Wick-Isserlis) and ranks (fraction RREF, MAX_ALGEBRA_OPS, MAX_ALGEBRA_BIT_LENGTH); 50-digit outward interval arithmetic for the cost hull, cone margin and Chebyshev transfer; $r_{\text{forb}} = \text{rank}(A;B;V) - \text{rank}(A;B)$.
- 5 Decision: canonical_verdict(); ABORT reasons before ABSTAIN reasons; codes from REASON_CODES; policy names sorted. VERIFIED coverage items carry a witness from POLICY_WITNESSES; a failed witness is POLICY_WITNESS_FAILED (ABORT).
PRODUCTION with no VERIFIED item is POLICY_COVERAGE_UNIVERSE_EMPTY (ABSTAIN). ATTESTED is recorded, not counted.
- 6 Certification (verifier): ACCEPT reproduced; all nodes CERTIFIED_NODE_STATUS; registry signature reproduced and key in verifier allowlist; both keys pinned; req_reg; PRODUCTION without overrides; registry bound to policy hash and verification profile hash; fresh (MAX_CLOCK_SKEW_SECONDS, MAX_CERTIFICATE_LIFETIME_SECONDS); all evidence bytes and
the primary source re-hashed by the verifier; relying party pins project, buyer, policy hash and profile hash; ONE_SHOT_EXECUTION_AUTHORIZATION additionally needs the expected action id and an unconsumed serial in a replay ledger.
- 7 Presentation: every wrapper field outside MERKLE_CARRIER_FIELDS is bound by presentation hashes in leaf 13; carrier fields must equal their normative values.
- 8 Semantics: REUSABLE_ATTESTATION certifies a state of the world for its validity window and never authorises an action.
- 9 Identity: the verification profile hash fingerprints this CONTRACT (constants and this text), not an executable.
The executable is identified by a BUILD: a release manifest signed by an operator-trusted release key, binding the source files, requirements.lock, the Python runtime, the installed bytes of every security-relevant dependency and the kernel runtime digest. The certificate commits build_id and implementation_sha256; certification requires both the prover's (build_id, digest) and the verifier's own (build_id, digest) to be in the relying party's approved builds, and the runtime known-answer self-tests (SHA-256, Ed25519, interval arithmetic, exact rank, canonical JSON, PDF extraction) to pass. No pin => never certified. In-process integrity cannot be self-attested: a process that can rewrite itself can rewrite its self-checks; the trust boundary is the immutable signed build/image.
The release manifest binds every file of the application directory and of each security dependency (including bytecode caches and unknown extra files), site hooks (.pth, sitecustomize), the interpreter binary and an optional image_digest; bytecode writes must be disabled. It is a filesystem description: the interpreter's standard library and the kernel/OS are covered only by the image digest, whose enforcement (signed-image admission) is the platform's. Production trust state (deployment, approved builds) is derived inside the service and never accepted from callers; kernel.verify_attestation is a primitive whose trust inputs are caller-asserted (certification_scope says so).

- 10 Citations: every node cites {document, page, quote}; certification requires the verifier to find each whitespace-normalised quote in the cited page of the cited document bytes. A citation proves that the quoted text exists in the source; whether the model value is a faithful reading of it remains the registry signer's attestation.
- 11 Limits: certificates $\leq$ MAX_CERTIFICATE_BYTES, JSON depth $\leq$ 32, leaves $\leq$ MAX_LEAF_BYTES, documents $\leq$ MAX_DOCUMENT_BYTES.

Appendix B - Artifact Fingerprints

Artifact	SHA-256
axiomzero_v7.9.9.zip	75002dda44e4db784217eddea22744417c86d98fc5ba03815041bf10e5e988b1
kernel.py	3a316bd8fe0d83b0fcd73b096ff6b05593ae46510201d9844980659222ac79b8
service.py	61390b8330d71bc5b7713ed1115808885073c9bb4e94d7177ef48a0d8132adb4
extractor.py	3b9be1164ea0d49237954648a9cda6994c23318594a4040f1185b79f3c6ab75
app.py	411b4ccd4075248177b4ffce633387776e240ddf7105344c0ce1d77e25be2122
make_release.py	f00c1dc699e047699a9ab08916979d595772a831eece60c4b8b7840bafdc4e2
requirements.lock	47f6064ca88da0145fb480f970d425dcfc73c5f8951dae595a10c6eb41f4b9bb
SPEC_v7.9.9.txt	922e267b111226cb725f9610157e6be99cdad5fb90c34cb9bfc4a23a056207e7
CHANGES_v7.9.9.md	e95185bdc2b18189a1108c3d02fb5ba82a12f02360521f33d1d6230e0e1c12d7
tests/test_regression_v799.py	cabf62e67cac26559d427c8820025fd4cc4965eaf09416fcf61a87c298976f27

Verification profile SHA-256 for the default v7.9.9 profile:

9e538e9e4cb6e74773ea0cbaa563feefcc40b7e919afd88a8bac4482e042efa7

A runtime implementation digest observed in the authoring sandbox was

73df7381063c36522a1898585f4a8bcab3bd9b5354147ed97406b09b76f9d5ee. Unlike the verification profile hash, this value is environment-sensitive and is not presented as a universal artifact identifier.

Appendix C - Claim/Evidence Matrix

Claim	Evidence class	What a third party can verify
The v7.9.9 code artifact exists.	Artifact hash	Hash the ZIP and compare with Appendix B.
The normative contract is fixed.	Profile hash + SPEC	Run verification_profile_hash and compare with the value in this paper.
The verifier reproduces key mathematical predicates.	Source + tests	Inspect verify_attestation and run the regression suite in the pinned environment.
The certificate commits evidence and presentation.	Source + negative tests	Mutate leaves/wrapper fields/signatures and confirm rejection.
Citation text is independently checked.	Source + PDF tests	Change page or quote and confirm CITATION_NOT_FOUND_IN_SOURCE.
One-shot authorization is replay guarded.	Ledger tests	Replay a consumed serial or complete with a stale fencing token.
Build drift is detected.	Signed release tests	Modify dependency/source bytes and re-run check_deployment.
The complete deployed process is immutable.	External platform evidence	Cannot be established by this Python artifact alone; requires signed image/admission evidence.
The system is externally certified secure.	Independent audit report	Not established by v7.9.9 or by this paper.

References

- [1] B. Laurie, A. Langley, E. Kasper, "Certificate Transparency," RFC 6962, IETF, June 2013. (Axiom Zero uses its domain-separated Merkle hashing convention; RFC 6962 has since been obsoleted by RFC 9162.)
- [2] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, January 2017.
- [3] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," FIPS PUB 180-4, 2015, DOI: 10.6028/NIST.FIPS.180-4.
- [4] IEEE, "IEEE Standard for Interval Arithmetic," IEEE Std 1788-2015.
- [5] L. Isserlis, "On a Formula for the Product-Moment Coefficient of any Order of a Normal Frequency Distribution in any Number of Variables," Biometrika, vol. 12, no. 1/2, pp. 134-139, 1918, DOI: 10.1093/biomet/12.1-2.134.
- [6] Axiom Zero v7.9.9 source artifact, SPEC_v7.9.9.txt, CHANGES_v7.9.9.md, kernel.py, service.py, extractor.py, and tests/test_regression_v799.py; artifact SHA-256 listed in Appendix B.