Canonicalization and Determinism: The Byte-Level Basis of Reproducible Proof

Intentrax Labs Research Paper | Evidence Portability
Published 2026-07-09 - Intentrax Labs - www.intentrax.com/labs

The serialization, numeric, and metadata rules that make two independent implementations compute the same hash for the same execution.

Why canonicalization is the foundation

Every downstream property of an Autonomous Execution Proof, tamper-evidence, replayability, and independent verification, reduces to a single requirement: two parties serializing the same protocol data must produce identical bytes, and therefore identical hashes. If serialization varied by implementation, language, or environment, hashes would diverge and no external verifier could confirm a proof. Canonicalization is the discipline that removes that variance. AEVN's canonical serialization must enforce a deterministic byte representation and must produce identical output across systems; an implementation that emits different serialization or different hashes for the same inputs is, by the conformance rules, non-compliant.

The serialization rules

The canonical strategy is ordered keys plus a canonical encoding, hashed with SHA-256. Key ordering removes the most common source of cross-implementation drift, since object member order is otherwise unspecified in most JSON libraries. The encoding is fixed so that whitespace, escaping, and structural representation cannot vary. All artifacts participating in verification, the Execution Event Object, the Policy Object Definition, the decision object, and the proof itself, pass through the same rules. The engineering cost is a custom canonical encoder rather than a stock serializer, accepted deliberately in exchange for cross-system consistency. The consequence, stated plainly in the architecture decisions, is that canonical rules must match across Go, Python, and TypeScript.

Deterministic numeric representation

Numbers are a primary hazard for determinism because floating-point representation varies across languages and platforms. AEVN therefore forbids floating-point values wherever deterministic replay is required. The preferred representations are integer minor units and canonical decimal strings; a monetary amount is represented as an integer minor-unit value with an explicit currency, not as a decimal such as 85000.00. Where decimal values are genuinely unavoidable, canonical string formatting rules must be enforced so that the byte representation is fixed. This rule exists so that a payment, a threshold, or a quantity hashes identically on every conformant implementation, which is a precondition for a proof to survive replay on a machine that is not the one that produced it.

Canonical data versus operational metadata

AEVN draws a hard line between data that participates in proof and data that merely accompanies it. Canonical protocol data, execution events, policy snapshots, decision objects, and proof artifacts, participates in canonical serialization, deterministic hashing, and proof verification. Operational metadata, request identifiers, observability traces, user-agent strings, and network metadata, may be stored or displayed but must never affect protocol hashing. It may be promoted into canonical inputs only by an explicit schema definition, never implicitly. This separation is what lets a proof remain stable while surrounding systems evolve: changing a trace format or a request-id scheme cannot alter a proof hash, so operational churn never invalidates evidence.

Algorithm agility and versioning

Determinism must not freeze the system in one cryptographic era. AEVN builds in agility through explicit identifiers. Every hash carries a hash_algorithm field (SHA-256 in the MVP), and every signature carries signature_algorithm and key_id fields, enabling key rotation, hardware-security-module integration, and algorithm upgrades without breaking existing proofs. Versioning operates at two levels: a protocol version, carried as aevn_protocol_version on every canonical artifact, governs verification semantics, while per-artifact schema versions, event_version, policy_version, and proof_version, govern structural format. Verification engines must reject artifacts whose protocol version is unsupported and reject hashes whose algorithm is unsupported, so agility never becomes ambiguity.

Conformance and immutability

Determinism is enforced, not assumed. All SDK implementations must pass the Intentrax Protocol Conformance Test Suite, which includes canonical artifact examples, canonical serialized output, expected hashes, and expected proof structures; producing different hashes or serialization is non-compliance. The proof registry compounds these guarantees by operating as an append-only ledger: artifacts are not modified after creation, updates create new records rather than mutating existing ones, and deletion is strongly discouraged, with audit logs preserved where compliance forces it. Cross-language consistency, conformance testing, and append-only storage together mean a proof hash is not merely computed once but remains reproducible and unrewritten for the life of the evidence.