4TS Technical Specification

v1.0.2

Complete vendor-neutral specification (§§0-11) with PCD schema, replay modes, decision boundaries, and implementation guidance.

Published: November 2025
Steward: FERZ LLC
License: CC BY-NC-ND 4.0 (Spec Text), MIT (Schemas/Vectors)
Document Type: Technical Standard

Table of Contents

Glossary

4TS: Four Tests Standard - vendor-neutral specification for verifiable AI governance.

Attestation: Cryptographic proof of authority, timing, and integrity using digital signatures.

Canonical Hash: SHA-256 hash computed over the canonicalized (deterministic, order-independent) JSON representation of a PCD.

CAS (Content-Addressed Storage): Storage system where content is retrieved by its cryptographic hash rather than location, ensuring immutability.

Decision Boundary: Temporal and hierarchical scope for governance decisions: deploy | policy | inference.

Effect-Token: Authorization token issued only after approval decision, enabling external side-effects. Must not exist on denial paths.

Fail-Closed: Security design where system denies actions by default if approval cannot be proven, preventing unauthorized operations.

Gate: Deterministic acceptance rule in protocol-replay mode. Format: metric(dataset[,group_by]) operator value.

Lineage: Execution trace showing all steps, inputs, outputs, and policy references that led to a decision.

Merkle Tree: Hierarchical hash structure enabling efficient verification of large artifacts through chunk-level hashing.

PCD (Proof-Carrying Decision): Canonical JSON object encoding all information necessary to verify the Four Tests at a decision boundary.

Protocol-Replay: Verification mode using deterministic gate re-evaluation on frozen datasets, suitable for stochastic workflows.

Replay Mode: Method for deterministically verifying governance decisions: State-Replay (byte-exact) or Protocol-Replay (gate-based).

State-Replay: Verification mode requiring byte-exact artifact reproduction and identical PCD hashes.

Verifier: System that validates PCD conformance to 4TS requirements through deterministic checks.

§0 Introduction and Metadata

System Identifier: 4TS-Standard (Four Tests Standard)

Version: v1.0.2 (specification with traceability matrix, glossary, and examples)

Steward: FERZ LLC (vendor-neutral publication)

License: CC BY-NC-ND 4.0 (specification text), MIT (schemas and test vectors)

Year: 2025

Authors: Edward Meyman (FERZ), FERZ Deterministic Governance Team

Knowledge Cutoff: 2025-11-09

0.1 Design Principles

- Evidence before effects (fail-closed by design)
- Deterministic acceptance at decision boundary
- Replayable by state or protocol
- Provable human custody and escalation
- Vendor neutrality and interoperability

0.2 Compatibility Matrix

The 4TS standard follows semantic versioning (SemVer). PCD major version 1.x maintains backward compatibility within minor version updates. Verifiers must ignore unknown fields under the must-ignore policy. Breaking changes occur only on major version increments.

§1 Purpose and Scope

The Four Tests Standard (4TS) ensures consequential AI decisions are stop-capable, owned, replayable, and escalatable—by design—via a proof-carrying decision object and deterministic acceptance at the decision boundary.

1.1 The Four Tests

STOP: System can be halted before side-effects. Effect-token issuance is gated by approval decision.

OWNERSHIP: Identified authority signs policy prior to execution window. Clear responsibility chains established.

REPLAY: Decision can be reproduced at boundary using either state-based or protocol-based verification.

ESCALATION: Mandatory custody transfer routes on denial or threshold crossings. Human-in-loop routes explicit.

1.2 Traceability Matrix

The following matrix maps each of the Four Tests to specific PCD fields, verification logic, and error codes, enabling auditors to systematically verify conformance.

Test	PCD Fields	Verification Logic	Error Codes
STOP	decision.outcome controls.fail_closed effect-token presence	Deny by default if approval unproven No effect-token on denial path Explicit approval required	E_SIDE_EFFECT_ON_DENIAL E_MISSING_EFFECT_TOKEN
OWNERSHIP	attestations.timestamps attestations.key_roles attestations.signatures	policy_signed ≤ exec_start ≤ exec_end Key role separation enforced Signature validity verified	E_PREEXEC_SIGNING E_KEY_SEPARATION E_SIG_INVALID E_SIG_MISSING
REPLAY	replay.strategy artifacts (SHA-256) controls.gates lineage steps	State: byte-equal artifacts + hash match Protocol: re-evaluate gates on frozen data Decision outcome must match	E_HASH_MISMATCH E_REPLAY_NONDETERMINISTIC E_STEP_REPRO_FAIL E_PROTOCOL_GATE_FAIL
ESCALATION	decision.routed decision.outcome lineage.policy_refs custody routes	Mandatory custody on denial Escalation routes explicit Human-in-loop paths encoded	E_MISSING_CUSTODY E_UNTYPED_LINEAGE E_POLICY_REF_MISSING

1.3 Core Components

- PCD (Proof-Carrying Decision) canonical JSON object
- Replay modes: State-Replay and Protocol-Replay
- Decision boundaries: deploy | policy | inference
- Conformance bundle: schemas + test vectors + quickstart validator
- Reference verifier (deterministic verification engine)

§2 PCD Schema Specification

The Proof-Carrying Decision (PCD) is a canonical JSON object that encodes all information necessary to verify the Four Tests at a decision boundary. The PCD schema is formally defined using JSON Schema draft 2020-12.

2.1 Mandatory Fields

Field	Type	Purpose
pcd_spec_version	string	Schema version (e.g., '1.0.1')
decision	object	Decision outcome and boundary info
artifacts	object	Models, policies, prompts, data
lineage	array	Execution steps with I/O
controls	object	Policy DSL, gates, monitors
attestations	object	Signatures, timestamps, hashes

2.2 Decision Object Structure

```
"decision": { "id": "unique-decision-id", "boundary": "deploy|policy|inference",
"outcome": "approved|denied|escalated", "effective_window": { "start":
"ISO-8601-timestamp", "end": "ISO-8601-timestamp-optional" }, "routed": false,
"replayable": true, "counterfactual": { "flip_var": "policy-variable-id",
"explanation": "minimal-flip-explanation" } }
```

2.3 Artifacts Structure

All artifacts must include SHA-256 hashes for byte-level custody verification. Large artifacts (models) may use Merkle trees with chunk-level hashing to enable streaming verification without full materialization.

```
"artifacts": { "models": [{ "id": "model-identifier", "sha256": "hash-required",
"merkle_root": "optional-for-large-models", "chunks": [{"i": 0, "size": 1048576,
"sha256": "chunk-hash"}] }], "policies": [{"id": "policy-id", "sha256": "hash"}],
"prompts": [{"id": "prompt-id", "sha256": "hash"}], "config": [{"id": "config-id",
"sha256": "hash"}], "data": [{"id": "dataset-id", "sha256": "hash"}] }
```

§3 Decision Boundaries

Decision boundaries define temporal and hierarchical scoping for governance decisions. The 4TS standard defines three boundary types with explicit parent-child relationships and effective time windows.

3.1 Boundary Types

Boundary	Scope	Parent	Typical Duration
deploy	Model/system deployment	None	Months to years
policy	Policy change or update	deploy	Weeks to months
inference	Individual inference action	policy	Milliseconds to minutes

3.2 Effective Windows

Each decision boundary must specify an effective window with a start timestamp. The end timestamp is optional and defaults to "until superseded." Child decisions must have effective windows that fall within their parent's window. This enables temporal reasoning about policy validity and ensures decisions can be audited with correct historical context.

3.3 Boundary Hierarchy

The hierarchy (deploy > policy > inference) enables layered governance. A deployment PCD establishes the foundational constraints. Policy PCDs inherit and may further restrict those constraints. Inference PCDs operate within the policy boundaries. This structure aligns with regulatory change control requirements and enables efficient audit trails.

§4 Replay Modes

The 4TS standard defines two replay modes that enable deterministic verification of governance decisions: State-Replay for byte-exact artifact reproduction, and Protocol-Replay for gate-based acceptance verification. Systems must declare their replay strategy in the PCD.

4.1 State-Replay Mode

Definition: Verification through byte-exact artifact reproduction.

Requirements:
• All artifacts must have identical SHA-256 hashes
• PCD canonical_hash must match exactly
• Decision outcome must be identical
• Suitable for deterministic pipelines with stable artifacts

Use Cases: Model deployment, policy updates, ETL pipelines, any workflow where artifacts can be frozen and reproduced byte-for-byte.

4.2 Protocol-Replay Mode

Definition: Verification through deterministic gate re-evaluation.

Requirements:
• Explicit gates define acceptance surface
• Gates must use deterministic metrics on frozen datasets
• Gate outcomes must match (decision must match)
• PCD hash may differ (implementation-dependent)
• Suitable for stochastic steps with deterministic validation

Use Cases: RAG systems with frozen indices, LLM tool chains with accuracy gates, model evaluation with fixed test sets, any workflow with sampling or non-deterministic components that still require deterministic approval.

4.3 Replay Strategy Selection

Characteristic	State-Replay	Protocol-Replay
Deterministic artifacts	Required	Optional
Frozen datasets	Implicit	Explicit required
Acceptance gates	Optional	Mandatory
PCD hash equality	Required	Not required
Stochastic steps	Not allowed	Allowed if gated
Verification cost	Low (hashing)	Higher (gate execution)

§5 Attestation and Verification

Attestations provide cryptographic proof of authority, timing, and integrity. The 4TS standard requires key role separation between policy signing and runtime execution to prevent post-hoc authorization.

5.1 Attestation Structure

```
"attestations": { "canonical_hash": "SHA-256-of-canonicalized-PCD", "signatures": [ {
"key_id": "policy-key-identifier", "algorithm": "EdDSA|ECDSA|RSA", "signature":
"base64-encoded-signature", "role": "policy|runtime|approver" } ], "timestamps": {
"policy_signed": "ISO-8601-timestamp", "exec_start": "ISO-8601-timestamp", "exec_end":
"ISO-8601-timestamp" }, "key_roles": { "policy": ["key-id-1", "key-id-2"], "runtime":
["key-id-3"] } }
```

5.2 Key Role Separation

Policy Keys: Sign policy content before execution window. Must not be used for runtime emission. Typically held by compliance/legal teams.

Runtime Keys: Sign PCD at execution time. Must not be used to sign policies. Typically held by automated systems with HSM backing.

Approver Keys: Sign escalation decisions or manual approval routes. Typically held by designated human authorities.

Enforcement: Verifiers MUST reject PCDs where policy keys and runtime keys overlap, unless explicitly allowed by policy configuration (not recommended).

5.3 Timestamp Ordering

The 4TS standard enforces strict timestamp ordering to prevent post-hoc policy signing. Required ordering: **policy_signed** ≤ **exec_start** ≤ **exec_end** Violation of this ordering triggers E_PREEXEC_SIGNING error and automatic denial.

5.4 Canonicalization

PCD canonicalization must be deterministic and order-independent:
• UTF-8 encoding
• Sorted keys (recursive for nested objects)
• Normalized numbers (no trailing zeros, scientific notation normalized)
• Normalized whitespace (consistent spacing)
• No undefined or null values (omit keys with null values)

The canonical_hash is computed over the canonicalized JSON string and provides tamper evidence for the entire PCD structure.

§6 Lineage and Custody

Lineage captures the execution flow of a decision, including all steps, inputs, outputs, and policy references. Custody tracking ensures artifact integrity through content-addressed storage and cryptographic hashing.

6.1 Lineage Structure

```
"lineage": [ { "id": "step-identifier", "op": "operation-name", "types": { "inputs":
["type1", "type2"], "outputs": ["type3"] }, "inputs": ["artifact-id-1",
"artifact-id-2"], "outputs": ["artifact-id-3"], "policy_refs": ["policy-id-1"],
"stochastic": false } ]
```

6.2 Typed Steps

Every step in the lineage must declare input and output types. This enables:
• Type checking for protocol replay
• Verification that steps connect correctly
• Clear interface contracts for audit
• Detection of type mismatches that could indicate tampering

Types should follow domain-specific ontologies (e.g., FHIR for healthcare, MISMO for mortgage).

6.3 Content-Addressed Storage

All artifacts referenced in lineage must be stored in content-addressed storage (CAS) using SHA-256 hashing. This ensures:
• Immutable artifact identity
• Tamper detection through hash verification
• Efficient deduplication across PCDs
• Reliable artifact retrieval for replay

Large artifacts may use Merkle trees (merkle_root + chunk hashes) to enable streaming verification without full materialization.

6.4 Policy References

Each step may reference one or more policies that governed its execution. Policy references must include:
• Policy artifact ID (with SHA-256 hash)
• Effective window during which policy applied
• Specific policy rules or gates evaluated

This enables auditors to trace which policies influenced each decision step and verify compliance with regulatory requirements.

§7 Conformance Requirements

Conformance to the 4TS standard requires passing all test vectors in the official conformance bundle and satisfying the requirements for PCD emission and verification. This section defines the normative requirements for 4TS-conformant systems.

7.1 Conformance Bundle v1.0.2

The conformance bundle includes:
- **pcd.schema.json:** JSON Schema (draft 2020-12) for PCD structure
- **verifier.config.schema.json:** Configuration schema for reference verifier
- **Test Vectors:** 3 positive cases (PCD-A1, A2, A3) and 5 negative cases (NC-1 through NC-5)
- **Quickstart Validator:** Python reference implementation
- **Documentation:** Implementation guide and test methodology

7.2 Required Test Vectors

Vector	Type	Expected Result	Tests
PCD-A1	Positive	PASS	State-replay, auto-approve
PCD-A2	Positive	PASS	Protocol-replay with gates
PCD-A3	Positive	PASS	Fail-closed denial path
NC-1	Negative	E_PREEXEC_SIGNING	Post-hoc signature
NC-2	Negative	E_MISSING_CUSTODY	Missing artifact hash
NC-3	Negative	E_KEY_SEPARATION	Key role violation
NC-4	Negative	E_UNTYPED_LINEAGE	Missing step types
NC-5	Negative	E_SIDE_EFFECT_ON_DENIAL	Effect-token on denial

7.3 Conformance Claims

A conformant system must publish a conformance claim with the following structure:

Tool@Version • PCD Major • Bundle Version • Pass Count • Manifest Hash • Logs Link

Example: *ACME-Verifier@2.1.0 • PCD-1 • Bundle-1.0.2 • 8/8 • sha256:abc123... • https://acme.com/4ts/logs*

Conformance claims enable users to verify that a system implements 4TS correctly and provides the required governance guarantees.

§8: Implementation Guidance

This section provides practical guidance for implementing 4TS-conformant systems across common AI/ML deployment patterns. Implementation profiles define how to apply 4TS principles to specific use cases.

8.1 Implementation Profiles

Profile	PCD Emission	Replay Mode	Key Considerations
LLM Tools	Per tool action with external effects	State (default) or Protocol	Use typed lineage for tool I/O
RAG Systems	Per response triggering workflows	Protocol (frozen index)	Gates on answerability/attribution
Model Deployment	At deployment and policy changes	State or Protocol (eval gates)	Require pre-exec policy signature
BPMN/ETL	Per job with external writes	State or Protocol	Compensating actions for rollbacks
Agentic Systems	Per plan execution	Protocol with explicit gates	Sub-PCDs for high-risk steps

8.2 Gate Definition Best Practices

When using Protocol-Replay, gates must be:
• **Deterministic:** Same inputs always produce same outputs
• **Frozen:** Use fixed datasets and metric implementations
• **Explicit:** Define metric, dataset, operator, value, and optional group_by
• **Comprehensive:** Cover all approval criteria in policy

Gate format: *metric(dataset[,group_by]) operator value*
Example: *accuracy(eval_set_v2.1) >= 0.95*

8.3 Fail-Closed Design Pattern

4TS systems must fail closed by design:
1. **Pre-execution validation:** Verify policy signatures before exec_start
2. **No effects on denial:** Effect-tokens MUST NOT exist on denial paths
3. **Explicit approval:** Require positive approval; absence of denial ≠ approval
4. **Escalation on uncertainty:** Route ambiguous cases to human custody

This pattern ensures that failures in governance infrastructure cannot lead to unauthorized actions.

§9 Error Handling and Recovery

The 4TS standard defines explicit error codes for all conformance violations. Systems must detect, report, and handle these errors deterministically to maintain governance integrity.

9.1 Error Codes

Code	Trigger	Recovery
E_HASH_MISMATCH	Artifact hash verification fails	Reject PCD, audit log
E_MISSING_CUSTODY	Required artifact hash missing	Reject PCD
E_PREEXEC_SIGNING	policy_signed > exec_start	Reject PCD, investigate tampering
E_KEY_SEPARATION	Policy and runtime keys overlap	Reject PCD, fix key config
E_UNTYPED_LINEAGE	Step missing input/output types	Reject PCD, fix lineage
E_STEP_REPRO_FAIL	Protocol replay step fails	Reject PCD, investigate
E_SIDE_EFFECT_ON_DENIAL	Effect-token present on denial	Reject PCD, critical audit
E_MISSING_EFFECT_TOKEN	No effect-token on approval	Reject PCD, fix emission
E_REPLAY_NONDETERMINISTIC	Replay produces different result	Reject PCD, audit
E_PROTOCOL_GATE_FAIL	Gate evaluation fails or non-deterministic	Reject PCD
E_SIG_INVALID	Signature verification fails	Reject PCD, investigate
E_SIG_MISSING	Required signature missing	Reject PCD

9.2 Recovery Mechanisms

When errors occur, systems should:
- **Immediate rejection:** Do not process PCD further; emit error code
- **Immutable logging:** Record error with full context for audit
- **Escalation:** Route to human custody for critical errors (tampering, key violations)
- **Remediation guidance:** Provide actionable steps to fix underlying issues
- **Monitoring:** Track error rates and patterns for systemic issues

9.3 Edge Cases

Clock Skew: Systems with distributed components may experience timestamp ordering issues due to clock skew. Implementations should use trusted timestamping services or accept small tolerances (e.g., ±5 seconds) with audit logging.

Mixed Canonicalization: If different services use different canonicalization implementations, hash mismatches may occur. Use a single canonical implementation across all components.

Streaming Effects: Some systems may begin side effects before full approval (streaming responses). This violates 4TS fail-closed requirements. Buffer outputs until approval is confirmed.

§10 Extensions and Interoperability

The 4TS standard supports vendor-specific extensions while maintaining core interoperability through the must-ignore policy. Extensions enable innovation without fragmenting the standard.

10.1 Extension Mechanism

Extensions use namespaced fields: **extensions.vendor.feature**

Reserved namespaces (must not be used by vendors):
- **extensions.std.*** - Reserved for future standardization
- **extensions.spec.* -** Reserved for 4TS specification team
- **extensions.conformance.* -** Reserved for conformance testing
- **extensions.test.*** - Reserved for test vectors

Vendor namespaces should follow reverse-DNS convention:
extensions.com.example.feature_name

10.2 Must-Ignore Policy

Verifiers and consumers MUST ignore unknown fields, including:
- Unrecognized extension namespaces
- New optional fields in future minor versions
- Vendor-specific metadata

This policy ensures forward compatibility: Systems implementing 4TS v1.0 can process PCDs from v1.1, v1.2, etc., even if they don't understand new optional features. Breaking changes occur only on major version increments (e.g., 2.0).

10.3 Interoperability Scenarios

Cross-Vendor Verification: Vendor A emits PCD, Vendor B verifies. Both must implement core 4TS; extensions are optional and ignored by non-supporting parties.

Regulatory Integration: 4TS PCDs can be submitted to regulators regardless of implementation vendor. Standard format enables consistent audit processes.

Multi-System Workflows: PCDs from different systems can be chained (parent decision → child decision) across organizational boundaries. Standard lineage format enables end-to-end audit trails.

Tool Ecosystem: Third-party tools (analyzers, visualizers, compliance dashboards) can process any 4TS-conformant PCD without vendor-specific integration.

10.4 Migration Paths

When migrating between major versions:

1. **Dual emission:** Systems should emit both old and new format PCDs during grace period
2. **Migration scripts:** Official tooling converts PCD-1.x → PCD-2.x
3. **Deprecation timeline:** N-1 version supported for 12 months after N release
4. **Verification support:** Verifiers should support both versions during migration period

§11 References and Changelog

11.1 Normative References

- **JSON Schema draft 2020-12:** https://json-schema.org/specification-links.html#2020-12
- **RFC 3339 (ISO 8601):** Date and time format for timestamps
- **RFC 6749 (OAuth 2.0):** Referenced for optional authentication patterns
- **SHA-256:** FIPS 180-4, Secure Hash Standard
- **EdDSA/ECDSA:** Digital signature algorithms for attestations

11.2 Related Documents

- **Deterministic AI Governance - Executive Guide:** Business rationale and minimum governance bar. 4TS is the technical standard that operationalizes these principles.
- **FERZ Implementation Guide:** Detailed implementation guidance for FERZ systems (LASO(f), DAGS-CVCA, DELIA) that implement 4TS.

11.3 Version History

Version	Date	Changes
1.0.0	2025-09	Initial release with core standard
1.0.1	2025-10	Added adoption profiles, clarified gate format, expanded error codes
1.0.2	2025-11	Added traceability matrix, glossary, and concrete PCD examples (Appendix A)

11.4 Contact and Feedback

Steward: FERZ LLC
Website: https://ferz.ai
Standards Email: contact@ferzconsulting.com
GitHub: https://github.com/ferz-ai/4ts-standard

Feedback and proposed changes should be submitted via GitHub issues or by email to the standards team. The 4TS standard is maintained through an open process with community input.

Appendix A: Concrete PCD Examples

This appendix provides two minimal but complete PCD examples demonstrating state-replay and protocol-replay modes. These examples are illustrative and simplified for clarity.

A.1 Example: Model Deployment (State-Replay)

A healthcare organization deploys a diagnostic model. This uses state-replay because all artifacts are deterministic and frozen at deployment time.

```
{ "pcd_spec_version": "1.0.2", "decision": { "id": "deploy-diag-model-v2.1",
"boundary": "deploy", "outcome": "approved", "effective_window": { "start":
"2025-11-01T00:00:00Z", "end": "2026-11-01T00:00:00Z" }, "routed": false, "replayable":
true }, "artifacts": { "models": [{ "id": "diagnostic-model-v2.1", "sha256":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" }], "policies": [{
"id": "fda-510k-compliance-policy", "sha256":
"a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e" }] }, "lineage": [ {
"id": "step-1-validation", "op": "model_validation", "types": { "inputs": ["model",
"test_dataset"], "outputs": ["validation_report"] }, "inputs":
["diagnostic-model-v2.1", "fda-test-set-2025"], "outputs": ["validation-report-001"],
"policy_refs": ["fda-510k-compliance-policy"], "stochastic": false } ], "controls": {
"fail_closed": true, "policy_dsl": "internal-v1", "replay": { "strategy": "state" } },
"attestations": { "canonical_hash": "c157a79031e1c40f85931829bc5fc552...",
"signatures": [{ "key_id": "policy-key-001", "algorithm": "EdDSA", "signature":
"5a8c9d2e...", "role": "policy" }], "timestamps": { "policy_signed":
"2025-10-30T14:00:00Z", "exec_start": "2025-10-30T14:30:00Z", "exec_end":
"2025-10-30T14:35:00Z" }, "key_roles": { "policy": ["policy-key-001"], "runtime":
["runtime-key-005"] } } }
```

A.2 Example: RAG System (Protocol-Replay)

A financial services RAG system generates an investment recommendation. This uses protocol-replay because retrieval involves ranking/scoring, but approval gates are deterministic on a frozen index snapshot.

```
{ "pcd_spec_version": "1.0.2", "decision": { "id": "inference-rag-rec-87234",
"boundary": "inference", "outcome": "approved", "effective_window": { "start":
"2025-11-08T10:15:00Z" }, "routed": false, "replayable": true, "counterfactual": {
"flip_var": "gate-attribution-threshold", "explanation": "If attribution score < 0.85,
decision would escalate" } }, "artifacts": { "models": [{ "id": "llm-finance-v3",
"sha256": "7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069" }],
"data": [{ "id": "corpus-snapshot-2025-11-08", "sha256":
"9c3f9e8eb8d14e1c7f9f9e8eb8d14e1c9c3f9e8eb8d14e1c7f9f9e8eb8d14e1c" }], "prompts": [{
"id": "investment-recommendation-template", "sha256":
"1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2b" }] }, "lineage": [ {
"id": "step-1-retrieval", "op": "vector_search", "types": { "inputs": ["query",
"corpus"], "outputs": ["retrieved_docs"] }, "inputs": ["user-query-87234",
"corpus-snapshot-2025-11-08"], "outputs": ["retrieved-docs-set-001"], "policy_refs":
["sec-compliance-rag-policy"], "stochastic": true }, { "id": "step-2-generation", "op":
```

```
"llm_generate", "types": { "inputs": ["retrieved_docs", "prompt_template"], "outputs":
["recommendation"] }, "inputs": ["retrieved-docs-set-001",
"investment-recommendation-template"], "outputs": ["recommendation-87234"],
"policy_refs": ["sec-compliance-rag-policy"], "stochastic": true }, { "id":
"step-3-validation", "op": "validate_attribution", "types": { "inputs":
["recommendation", "retrieved_docs"], "outputs": ["attribution_score"] }, "inputs":
["recommendation-87234", "retrieved-docs-set-001"], "outputs":
["attribution-score-0.91"], "policy_refs": ["sec-compliance-rag-policy"], "stochastic":
false } ], "controls": { "fail_closed": true, "policy_dsl": "internal-v1", "replay": {
"strategy": "protocol" }, "policy_invariants": { "gates": [ { "id":
"gate-1-attribution", "metric": "attribution_score", "dataset":
"step-3-validation-output", "op": ">=", "value": 0.85 }, { "id":
"gate-2-corpus-frozen", "metric": "corpus_hash", "dataset":
"corpus-snapshot-2025-11-08", "op": "==", "value":
"9c3f9e8eb8d14e1c7f9f9e8eb8d14e1c9c3f9e8eb8d14e1c7f9f9e8eb8d14e1c" } ] } },
"attestations": { "canonical_hash": "8d7e6f5a4b3c2d1e0f9a8b7c6d5e4f3a...",
"signatures": [{ "key_id": "runtime-key-012", "algorithm": "EdDSA", "signature":
"9e8f7d6c...", "role": "runtime" }], "timestamps": { "policy_signed":
"2025-11-01T00:00:00Z", "exec_start": "2025-11-08T10:15:02Z", "exec_end":
"2025-11-08T10:15:08Z" }, "key_roles": { "policy": ["policy-key-003"], "runtime":
["runtime-key-012"] } } }
```

Key Differences:

• State-replay example has deterministic steps (stochastic: false), requiring byte-exact reproduction.

• Protocol-replay example has stochastic steps (retrieval, generation) but deterministic gates that validate the outcome (attribution score, corpus hash).

• Both enforce fail-closed design and proper timestamp ordering.

• Both include complete lineage showing step types, inputs, outputs, and policy references.