

Typed Governance Tuples v0.3.0 Candidate Manuscript

Title: Typed Governance Tuples: Portable Interchange and Reference Mediation for Auditable Agentic Systems

Status: Candidate Manuscript, v0.3.0 (Hardened from ARCANA Multi-Agent Peer Review)

Date: 2026-09-21

Author: Reuben Bowlby (reuben@hummb1.io), HUMMBL Research Institute / HUMMBL, LLC

ORCID: 0009-0002-5620-1103

Primary Repository: <https://github.com/hummb1-io/governance-tuple-reference>

Abstract

Autonomous multi-agent systems increasingly rely on complex, multi-tiered delegation: humans delegate high-level intent to orchestrators, orchestrators decompose work across specialized sub-agents, and agents invoke tools, external APIs, and compute substrates. Conventional audit mechanisms capture isolated slices of this lifecycle: event logs record operational timelines, distributed traces capture causality, capability tokens assert authority, and provenance graphs document artifact lineage. However, existing architectures suffer from three critical governance failures: (1) they rely on untrusted executing agents to self-report their own actions, inviting deceptive alignment and unobserved tampering; (2) they artificially scope audit completeness to voluntary “governed paths,” leaving common-pool resources vulnerable to unmonitored sideband bypasses; and (3) they enforce rigid, monocentric tree hierarchies that cannot adapt to distributed operational discovery under uncertainty.

We introduce **Typed Governance Tuples (v0.3.0)**: a formal specification and portable interchange protocol for auditable multi-agent runtime control. A typed governance tuple is an immutable, cryptographically signed envelope pairing a common audit lineage with a strongly typed governance payload. This v0.3.0 specification hardens the framework against classical distributed security and multi-agent alignment vulnerabilities: 1. **Threat-Model Grounding:** Establishes explicit adversary capabilities and trust boundaries separating untrusted agent processes from trusted execution monitors; 2. **Complete Reference Mediation:** Mandates that EVIDENCE records be emitted exclusively by out-of-band execution monitors (hypervisors, kernel eBPF probes, or isolated sidecars) rather than self-reported by executing agents; 3. **Polycentric DAG Topologies:** Generalizes authority delegation from rigid single-parent trees to Directed Acyclic Graphs (DAGs) supporting multi-principal authorization (m -of- n threshold co-signing), low-cost dispute contestation, and lateral agent covenants; 4. **Mandatory Cryptographic Envelopes:** Enforces canonical public-key signatures across all normative records, strictly segregating unsigned structures into non-normative diagnostic schemas; 5. **Semantic Goal Containment:** Pairs syntactic tool whitelisting with verifiable semantic objective containment to prevent goal drift across deep delegation cascades.

Typed governance tuples do not replace underlying primitives such as UCAN, SPIFFE, OpenTelemetry, W3C PROV, or IETF RATS; rather, they provide the cross-system governance envelope required to bind bounded intent, delegated authority, runtime state, and independent evidence into an auditable, non-repudiable proof of custody.

1. Problem Statement

Auditing an autonomous agentic decision requires answering five coupled questions: 1. **Bounded Intent:** What specific objective, operational boundaries, and resource budgets were authorized? 2. **Delegated Authority:** Which principal granted authority, to whom, under what caveats, and via what cryptographic proof? 3. **Runtime Lifecycle State:** Where does this task reside in the collective workflow, and what control-plane decisions (throttles, halts, approvals) were applied? 4. **Mediated Execution:** What tools were actually executed, what artifacts were produced, and how was this execution independently verified

outside the agent’s mutable process memory? 5. **Attestation & Verification:** What independent evaluator reviewed the evidence, and what was the verdict?

In current production agent architectures, these dimensions are siloed or structurally compromised: - **The Self-Reporting Fallacy:** Many agent frameworks rely on the language model itself to append its tool calls and outputs to an audit log. A compromised, prompt-injected, or deceitful agent can simply emit fabricated logs while executing unauthorized actions via ambient host authority. - **The Governed-Path Mirage:** Declaring “100% audit completeness” over a proprietary agent router is meaningless if the host environment permits uninstrumented network egress or out-of-band subprocess execution. - **Central Planning Sclerosis (The Fatal Conceit):** Forcing root orchestrators to pre-calculate all permissible sub-tools and operations ex-ante suppresses local problem-solving, causing agents to fail catastrophically when encountering unmodeled friction or to seek unmonitored shadow workflows. - **Monocentric Fragility (The Unilateralist’s Curse):** Strictly single-parent tree delegation allows a single rogue sub-agent at depth k to unilaterally trigger high-impact actions without multi-principal consensus or contestability channels.

Typed Governance Tuples resolve these failures by defining a standardized, portable envelope and lifecycle grammar that binds external capability tokens, independent reference monitors, and structured attestation into an immutable audit graph.

2. Threat Model, Adversary Capabilities & Trust Boundaries

To establish verifiable auditability, the tuple architecture operates under an explicit threat model.

flowchart TD

```

    subgraph Untrusted Domain
        Agent[Agent Process / LLM Core]
    end

    subgraph Trusted Execution Boundary
        RM[Isolated Reference Monitor / Kernel eBPF / Sandbox Proxy]
        Tools[Tool & API Gateways]
    end

    subgraph Governance & Audit Fabric
        Verifier[Independent Attestation Verifier]
        Ledger[(Append-Only Signed Tuple Store)]
    end

    Agent -->|1. Requests Tool Call| RM
    RM -->|2. Validates DCT & CONTRACT| RM
    RM -->|3. Mediates Execution| Tools
    Tools -->|4. Raw Execution Result| RM
    RM -->|5. Emits Out-of-Band EVIDENCE| Ledger
    RM -->|6. Returns Sanitized Output| Agent
    Ledger -->|7. Evaluates Evidence Chain| Verifier
    Verifier -->|8. Emits ATTEST Tuple| Ledger

```

2.1 Adversary Capabilities

We evaluate the architecture against four adversary classes: 1. **Compromised Delegate (A_{agent}):** An agent process that has been hijacked via prompt injection, model weight misalignment, or malicious tool outputs. A_{agent} attempts to execute unauthorized tools, expand its resource budget, forge execution receipts, or omit audit records. 2. **Colluding Delegator-Delegatee (A_{collude}):** An internal delegator that issues unauthorized capability grants to a sub-agent. The architecture must ensure that monotonic attenuation and

root contract invariants detect and prevent privilege escalation. 3. **Malicious Tool / Egress Point (A_{tool})**: An external service or API that returns corrupted outputs, attempts side-channel exfiltration, or injects adversarial instructions into agent memory. 4. **Eavesdropping / Replaying Adversary (A_{network})**: A network-level attacker that attempts to intercept, replay, or equivocate governance records across distributed agent nodes.

2.2 Trust Boundaries

- **Untrusted:** Agent runtime processes, LLM inference outputs, user-supplied prompt contexts, and external network tool endpoints.
- **Trusted:**
 - **The Reference Monitor:** An isolated supervisor (e.g., hypervisor, Linux container namespace with eBPF enforcement, or local loopback proxy) with complete mediation over network, filesystem, and subprocess execution.
 - **Key Custody Modules:** Hardware-backed (TPM/HSM) or OS-keychain-bound cryptographic identities holding the signing keys of human principals and supervisory monitors.
 - **Append-Only Witness Storage:** A tamper-evident log enforcing backward-hash integrity and public/witness checkpointing.

2.3 Core Security Invariants

- **Invariant I-1 (Complete Mediation):** No tool invocation, external network access, or IPC can occur without passing through the Reference Monitor.
- **Invariant I-2 (Non-Self-Reporting):** An EVIDENCE tuple is cryptographically invalid unless signed by the Reference Monitor’s key. The executing agent cannot emit its own EVIDENCE tuple.
- **Invariant I-3 (Non-Repudiation):** Every normative tuple MUST contain a valid cryptographic signature over its canonical serialized envelope and payload.
- **Invariant I-4 (Monotonic Attenuation):** Authority granted in child capability tokens MUST represent a strict mathematical subset of parent authority, verified syntactically and semantically.

3. The Normative Envelope

Every governance tuple conforms to a universal, tamper-evident envelope. The envelope guarantees uniform indexing, causal lineage tracking, and cryptographic non-repudiation.

Table 1: Universal Tuple Envelope Specification

Field	Type	Required	Normative Meaning
<code>tuple_type</code>	enum	MUST	Registered class (CONTRACT, DCT, DCTX, SYSTEM, EVIDENCE, ATTEST, REVOCATION, DISPUTE).
<code>schema_version</code>	semver string	MUST	Fixed semver string ("0.3.0").
<code>entry_id</code>	UUIDv4 / URI	MUST	Universally unique, immutable record identifier.

Field	Type	Required	Normative Meaning
<code>timestamp</code>	ISO 8601 UTC	MUST	Normalized UTC timestamp with microsecond precision.
<code>intent_id</code>	UUIDv4 / URI	MUST	Root human or organizational intent lineage identifier.
<code>task_id</code>	UUIDv4 / URI	MUST	Specific unit-of-work identifier.
<code>parent_task_ids</code>	array[UUIDv4]	MUST	Array of direct causal parent task IDs (supports polycentric DAGs; empty for root).
<code>profile_id</code>	URI / string	MUST	Normative profile defining validation constraints (e.g., "humbl.idp.v0.3.0").
<code>tuple_data</code>	object	MUST	Type-specific structured payload conforming to class schema.
<code>previous_hash</code>	SHA-256 string	MUST	Cryptographic digest of preceding tuple in local partition log (genesis = 64 zeros).
<code>witness_checkpoint</code>	string / null	<i>OPTIONAL</i>	Merkle root or external witness log signature (e.g., RFC 9162 / Trillian tree head).
<code>signature</code>	object	MUST	Cryptographic signature object (<code>algorithm</code> , <code>key_id</code> , <code>value</code>).

3.1 Cryptographic Signing and Canonicalization

In v0.3.0, **signature** is **normatively required** for all governance interchange. Unsigned structures are forbidden from circulating as valid governance records and are quarantined to a segregated, non-normative schema (`draft_tuple_fixture.json`).

The signature is computed over the canonical RFC 8785 (JSON Canonicalization Scheme - JCS) representation of the tuple with the **signature** field excluded:

$$\Sigma = \text{Sign}_{K_{\text{issuer}}}(\text{JCS}(\text{Tuple} \setminus \{\text{"signature"}\}))$$

Supported signature algorithms: Ed25519 (preferred), ES256 (ECDSA over P-256 with SHA-256), and RSA-PSS-SHA256.

4. Core Tuple Taxonomy

The v0.3.0 core taxonomy defines six fundamental tuple classes, each corresponding to an irreducible governance role:

CLASS	ROLE	PRIMARY EMITTER
CONTRACT	Bounded task specification	Authorizing Principal / Orchestrator
DCT	Delegated Capability Token	Authorizing Principal / Delegator
DCTX	Lifecycle & DAG Context State	Orchestrator / Runtime Scheduler
SYSTEM	Control-plane enforcement gate	Reference Monitor / Policy Engine
EVIDENCE	Execution artifact receipt	Independent Execution Monitor
ATTEST	Formal evaluation & audit	Independent Verifier / Arbiter

4.1 CONTRACT (Bounded Work Definition)

The CONTRACT tuple establishes the task mandate, resource constraints, permitted tool boundaries, and dispute protocols before delegation or execution begins.

Table 2: CONTRACT Payload Specification (tuple_data)

Field	Type	Required	Definition
contract_id	UUIDv4 / URI	MUST	Canonical identifier for this contract.
objective	string	MUST	Explicit task mandate in controlled natural language.
semantic_bounds	object	MUST	Formal constraints defining permissible goal scope.
allowed_tools	array[string]	MUST	Whitelist of permitted tool identifiers.
denied_tools	array[string]	MUST	Blacklist of strictly forbidden tool identifiers.
allowed_operations	array[string]	MUST	Specific sub-operations permitted within tools.
resource_bounds	object	MUST	Quantitative caps: max_tokens , max_usd , max_wallclock_seconds .
evidence_policy	array[string]	MUST	Mandatory evidence artifacts required before completion.

Field	Type	Required	Definition
<code>risk_tier</code>	enum	MUST	Risk classification: TIER_0_TRIVIAL, TIER_1_STANDARD, TIER_2_ELEVATED, TIER_3_CRITICAL.
<code>required_signers</code>	integer	MUST	m -of- n threshold requirement for activation ($m \geq 2$ for Tier 2/3).
<code>dispute_protocol</code>	string	MUST	URI to applicable arbitration and contestation policy.
<code>expires_at</code>	ISO 8601 UTC	MUST	Hard wall-clock expiration timestamp.

4.2 DCT (Delegated Capability Token)

The DCT tuple encapsulates delegated authority. Rather than reinventing authorization cryptography, the DCT acts as a governance container that wraps standards-compliant capability tokens (UCAN, Biscuit, Macaroons, or W3C Verifiable Credentials) while binding them strictly to a governing **CONTRACT**.

Table 3: DCT Payload Specification (`tuple_data`)

Field	Type	Required	Definition
<code>dct_id</code>	UUIDv4 / URI	MUST	Canonical identifier for this authority grant.
<code>contract_id</code>	UUIDv4 / URI	MUST	Linked CONTRACT being delegated.
<code>issuers</code>	array[string]	MUST	SPIFFE IDs / public keys of authorizing principals.
<code>subject</code>	string	MUST	SPIFFE ID / public key of designated delegatee agent.
<code>audience</code>	string	MUST	Intended execution environment / reference monitor ID.
<code>nonce</code>	string (hex)	MUST	Cryptographic 128-bit replay-prevention challenge.
<code>allowed_tools</code>	array[string]	MUST	Strictly attenuated tool whitelist ($\subseteq$ CONTRACT).
<code>allowed_operations</code>	array[string]	MUST	Strictly attenuated operation whitelist ($\subseteq$ CONTRACT).

Field	Type	Required	Definition
caveats	array[object]	MUST	Contextual execution predicates (e.g., rate limits, IP bounds).
issued_at	ISO 8601 UTC	MUST	Issuance timestamp.
expires_at	ISO 8601 UTC	MUST	Expiration timestamp ($\leq$ CONTRACT.expires_at).
token_envelope	object	MUST	Encapsulated raw token: <code>format</code> ("UCAN", "BISCUIT", "MACAROON"), <code>token_string</code> .

4.3 DCTX (Delegation Context & Polycentric Lifecycle)

The DCTX tuple records state transitions within the delegation graph. To eliminate the vulnerabilities of monocentric trees and unilateral halts, v0.3.0 introduces a polycentric DAG lifecycle including contestation and renegotiation arenas.

```
stateDiagram-v2
    [*] --> PROPOSED
    PROPOSED --> ISSUED: Multi-Signature Approval
    ISSUED --> RUNNING: Reference Monitor Verification
    RUNNING --> EVIDENCE_READY: Execution Completed
    EVIDENCE_READY --> VERIFIED: ATTEST Pass

    RUNNING --> DISPUTED: Anomaly / Boundary Challenge
    DISPUTED --> RENEGOTIATING: Variance Requested
    RENEGOTIATING --> ISSUED: Amended Contract
    RENEGOTIATING --> CONTESTED_HALT: Consensus Failure

    RUNNING --> FAILED: Execution Error / Timeout
    RUNNING --> CONTESTED_HALT: Emergency Kill-Switch
```

Table 4: DCTX Payload Specification (tuple_data)

Field	Type	Required	Definition
dctx_id	UUIDv4 / URI	MUST	Canonical lifecycle context identifier.
task_id	UUIDv4 / URI	MUST	Linked unit of work.
parent_task_ids	array[UUIDv4]	MUST	Inbound edges in the polycentric delegation DAG.
delegator_ids	array[string]	MUST	Principals holding delegating authority.
delegatee_id	string	MUST	Agent assigned to task execution.
dag_depth	integer	MUST	Maximum directed topological distance from root intent.

Field	Type	Required	Definition
<code>status</code>	enum	MUST	Current lifecycle state (see diagram above).
<code>contract_id</code>	UUIDv4 / URI	MUST	Governing CONTRACT.
<code>dct_id</code>	UUIDv4 / URI	MUST	Linked DCT exercising authority.
<code>dispute_ref</code>	UUIDv4 / null	<i>CONDITIONAL</i>	Linked DISPUTE tuple if status is DISPUTED or CONTESTED_HALT.

4.4 SYSTEM (Synchronous Runtime Control-Plane Barrier)

SYSTEM tuples are not passive audit logs; they record synchronous policy enforcement decisions made by the Reference Monitor at runtime barriers (e.g., pre-execution gates, circuit breakers, or rate limiters).

Table 5: SYSTEM Payload Specification (tuple_data)

Field	Type	Required	Definition
<code>system_id</code>	UUIDv4 / URI	MUST	Canonical event identifier.
<code>control_surface</code>	string	MUST	Active enforcement surface ("SANDBOX_EGRESS", "KERNEL_EBPF", "API_GATEWAY").
<code>enforcement_mode</code>	enum	MUST	Mode: "FAIL_CLOSED_BLOCK", "DYNAMIC_THROTTLE", "AUDIT_ONLY".
<code>barrier_phase</code>	enum	MUST	Phase: "PRE_EXECUTION_GATE", "MID_EXECUTION_PROBE", "POST_EXECUTION_CHECK".
<code>decision</code>	enum	MUST	Enforcement result: "PERMITTED", "DENIED", "THROTTLED", "EMERGENCY_HALT".
<code>trigger_rule</code>	string	MUST	Verifiable reference to the specific policy or invariant evaluated.
<code>linked_task_id</code>	UUIDv4 / URI	MUST	Task evaluated at the barrier.
<code>monitor_attestation</code>	object	MUST	Cryptographic proof of monitor health and policy hash.

4.5 EVIDENCE (Out-of-Band Execution Receipt)

The EVIDENCE tuple captures the raw execution telemetry, tool invocations, and output artifacts. **In accordance with Invariant I-2, the emitter MUST be the Reference Monitor, completely eliminating agent self-reporting.**

Table 6: EVIDENCE Payload Specification (tuple_data)

Field	Type	Required	Definition
evidence_id	UUIDv4 / URI	MUST	Canonical evidence identifier.
contract_id	UUIDv4 / URI	MUST	Governing CONTRACT being discharged.
dct_id	UUIDv4 / URI	MUST	Authority token presented for execution.
monitor_id	string	MUST	SPIFFE ID / public key of the observing Reference Monitor.
execution_status	enum	MUST	Status: "SUCCESS", "EXECUTION_ERROR", "RESOURCE_EXHAUSTED", "TIMEOUT".
tools_invoked	array[string]	MUST	Tools physically mediated by the monitor ($\subseteq$ DCT.allowed_tools).
operations_run	array[string]	MUST	Operations executed ($\subseteq$ DCT.allowed_operations).
artifacts	array[object]	MUST	Content-addressed artifact hashes (sha256, uri, mime_type).
consumed_resources	object	MUST	Physical accounting: tokens_consumed, wallclock_ms, egress_bytes.
data_classification	enum	MUST	Handling tag: "PUBLIC", "INTERNAL", "RESTRICTED_DUAL_USE", "CONFIDENTIAL".
redaction_manifest	object / null	<i>OPTIONAL</i>	Salted commitment hashes for redacted sensitive payloads.

4.6 ATTEST (Independent Adjudication & Verification)

The ATTEST tuple records the findings of an independent verifier (human auditor, consensus committee, or specialized verification model) that evaluates EVIDENCE against CONTRACT requirements.

Table 7: ATTEST Payload Specification (tuple_data)

Field	Type	Required	Definition
attest_id	UUIDv4 / URI	MUST	Canonical attestation identifier.
evidence_id	UUIDv4 / URI	MUST	Linked EVIDENCE record evaluated.
verifier_id	string	MUST	SPIFFE ID / public key of the independent verifier.
verdict	enum	MUST	Adjudication: "PASS", "FAIL", "CONTESTED", "WAIVED".
verification_checks	array[object]	MUST	Detailed rubric results: check_name, status, eval_metric, evidence_ref.
confidence_score	float	MUST	Verifier confidence metric ([0.0, 1.0]).
dissenting_opinions	array[object]	<i>OPTIONAL</i>	Minority opinions in multi-party verification panels.

5. Lifecycle Invariants & Governance Rules

The v0.3.0 specification formalizes seven mandatory lifecycle invariants enforced at runtime barriers and validated during forensic reconstruction:

Invariant 1: Contract Precedence & Threshold Activation

A DCT cannot be activated without referencing an active, unexpired **CONTRACT**. For elevated risk tiers (**TIER_2** or **TIER_3**), activation requires cryptographic co-signatures from m -of- n authorized principals, mitigating the unilateralist's curse.

Invariant 2: Complete Out-of-Band Mediation (Non-Self-Reporting)

An **EVIDENCE** tuple is structurally invalid if its cryptographic signing key matches the executing agent's identity. All tool invocations and artifact commitments **MUST** be observed and signed by the Reference Monitor.

Invariant 3: Semantic and Syntactic Monotonic Attenuation

Let $A(T)$ represent the authority granted by token T , comprising tool sets $\mathcal{T}$, operation sets $\mathcal{O}$, and semantic objective scope $\mathcal{S}$. For any child delegation T_{child} derived from T_{parent} :

$$\mathcal{T}_{\text{child}} \subseteq \mathcal{T}_{\text{parent}} \quad \wedge \quad \mathcal{O}_{\text{child}} \subseteq \mathcal{O}_{\text{parent}} \quad \wedge \quad \mathcal{S}_{\text{child}} \models \mathcal{S}_{\text{parent}}$$

Authority expansion at subordinate depths is cryptographically rejected. If an operational agent discovers that local task success requires capabilities outside $A(T_{\text{child}})$, it **MUST** initiate an explicit **DISPUTE** / **RENEGOTIATING** petition rather than attempting out-of-band escalation.

Invariant 4: Freshness Challenge & Replay Defense

Every DCT presented to a Reference Monitor MUST contain a unique cryptographic **nonce** and bind the monitor's public **audience** identifier. Reference Monitors maintain a local replay cache; re-submission of an existing (dct_id, nonce) pair triggers an immediate **SYSTEM** halt.

Invariant 5: Synchronous Pre-Execution Interception Barrier

State transition from **ISSUED** to **RUNNING** in a DCTX record requires an accompanying **SYSTEM** tuple with **decision**: "PERMITTED" and **barrier_phase**: "PRE_EXECUTION_GATE". Execution cannot commence asynchronously prior to policy enforcement.

Invariant 6: Evidentiary Closure Before Attestation

An **ATTEST** tuple cannot be emitted without referencing a valid, signed **EVIDENCE** tuple whose **contract_id** matches the verifier's inspection mandate.

Invariant 7: Polycentric Contestation & Graduated Sanctions

When a policy violation or evidence anomaly is detected, the Reference Monitor does not default to catastrophic uncoordinated system termination. Instead: 1. The monitor transitions the task to **DISPUTED** and applies graduated throttling; 2. A formal **DISPUTE** tuple is emitted, identifying the conflicting invariants; 3. The delegator and delegatee enter a bounded **RENEGOTIATING** window; 4. If agreement is reached, an amended contract is signed; if irreconcilable, a coordinated **CONTESTED_HALT** is safely executed.

6. Composition With Related Standards

Typed Governance Tuples do not replace existing standards in identity, authorization, tracing, or attestation. Rather, the tuple envelope functions as the overarching **governance binding plane** that cross-links these heterogeneous primitives into a unified audit graph.

STANDARD	DOMAIN	TUPLE COMPOSITION MECHANISM
UCAN / Biscuit	Capability Tokens	Wrapped directly inside DCT payload.
SPIFFE / SPIRE	Workload Identity	Supplies canonical issuer/subject IDs.
OpenTelemetry	Distributed Tracing	Span IDs linked in EVIDENCE artifacts.
W3C PROV	Lineage Modeling	DCTX DAG maps directly to prov:wasDerivedFrom.
IETF RATS (RFC 9334)	Remote Attestation	RATS Evidence -> EVIDENCE; Results -> ATTEST.
in-toto / SLSA	Supply-Chain Provenance	In-toto predicates wrapped in ATTEST.

7. Implementation & Verification Suite

A reference implementation and verification harness for v0.3.0 is maintained in `packages/python/humbl-governance` and `governance-tuple-reference`:

1. **Normative JSON Schema Suite (schema/v0.3.0/)**: Strict JSON Schema specifications for the universal envelope, core classes (**CONTRACT**, **DCT**, **DCTX**, **SYSTEM**, **EVIDENCE**, **ATTEST**), and extension classes (**DISPUTE**, **REVOCATION**).
2. **Semantic DAG Validator (verify_chain.py / semantic_validator.py)**:
 - Validates RFC 8785 JCS canonicalization and Ed25519 signatures.
 - Reconstructs polycentric DAG delegation topologies from arbitrary leaf tuples.
 - Proves monotonic attenuation ($\mathcal{T}_{\text{child}} \subseteq \mathcal{T}_{\text{parent}}$).

- Verifies out-of-band monitor signatures on EVIDENCE records.
 - Enforces replay protection and nonce freshness.
3. **Reference Monitor Adapter (monitors/ebpf_proxy/):**
- Lightweight loopback enforcement proxy demonstrating complete mediation over tool HTTP/gRPC requests.
 - Emits signed EVIDENCE tuples upon execution completion without agent intervention.
-

8. Revision History & Audit Traceability

Changes in v0.3.0 (Hardened from ARCANA 5-Lens Multi-Agent Review):

1. **Threat Model Formalization (Schneier P1.4, Bostrom P2):** Introduced Section 2, defining explicit adversary classes, trust boundaries, and physical execution assumptions.
2. **Elimination of Self-Reporting Fallacy (Schneier P1.2, Foucault P2, Bostrom P2):** Mandated Invariant I-2; EVIDENCE tuples must be generated and signed exclusively by an out-of-band Reference Monitor.
3. **Resolution of Governed-Path Circumvention (Schneier P1.1, Foucault P1, Ostrom P2, Bostrom P2, Hayek P2):** Re-scoped claims and established Invariant I-1 (Complete Mediation) at the sandbox/proxy boundary.
4. **Mandatory Signatures in Envelope (Schneier P2.3, Foucault P2, Ostrom P3, Bostrom P3):** Made `signature` mandatory across all normative records; relegated unsigned fixtures to a segregated testing schema.
5. **Polycentric DAG Delegation (Ostrom P2.1, Hayek P2.4, Bostrom P2.3):** Replaced single-parent tree restriction with `parent_task_ids` array, supporting DAG topologies and *m-of-n* threshold signatures.
6. **Contestability & Graduated Sanctions (Ostrom P2.3/P2.5, Foucault P3):** Introduced DISPUTED and RENEGOTIATING states into the DCTX lifecycle to prevent catastrophic uncoordinated halts.
7. **Semantic Objective Containment (Bostrom P1.1, Hayek P1.1):** Added formal semantic bounds to CONTRACT payloads to prevent goal drift across deep delegation cascades.
8. **Replay & Freshness Defense (Schneier P2.7):** Added mandatory cryptographic `nonce` and `audience` fields to DCT grants.