

Security

Integrity, Authorization, Audit, and Non-Repudiation on a Structural Coordinate Space

SSCCS Initiative*

ssccs.org

August, 2026

Abstract

Coordination traffic in synTagma moves over structural paths: routing updates name target paths, resolvers exchange evidence, and the audit trail records what happened. tagma-sec is the security primitive layer that answers the accompanying questions: which principal may act on which path, whether the record is intact, whether the origin of an update can be denied, and what evidence remains. The layer provides integrity, authorization, audit, and non-repudiation over Tagma coordinate objects and coordination traffic. It claims no confidentiality from coordinate structure; confidentiality is delegated to the hybrid interface. The initial implementation covers specification milestones 1 to 4 with a legacy pattern as a reverse-verification mirror and the route-update workflow as the scenario contract.

1 Introduction

The initial framing of Tagma security proposed the coordinate space itself as a substitute for encryption¹. Analysis identified four engineering weaknesses in that framing, each recorded as a rejected claim in the specification²:

- Published interpretation rules collapse confidentiality. The mapping from application semantics to Coord slots is public, so a captured CoordPath recovers its meaning without breaking any key.
- A captured CoordPath can be replayed. Nothing in the coordinate binds it to a time, a principal, or a session.
- The valid subspace is enumerable by construction. The 16-bit space holds 11,172 valid values out of 65,536, so structural validity alone cannot serve as a secret.
- Composition, decomposition, and linearization lack any keyed transformation, so those operations cannot satisfy a confidentiality requirement on their own.

The rejection follows from the Kerckhoffs principle: the composition formula, the linearization rule, and the validity bounds are public specifications, and a hardware decoder must remain verifiable against public rules. tagma-sec therefore builds security on keyed primitives over public arithmetic, and Tagma contributes collision-free addressing rather than structural secrecy.

*; Corresponding author: contact@ssccs.org

¹Tagma Whitepaper: docs.ssccs.org/projects/syntagma/tagma/

²Tagma Security Layer Specification: github.com/ssccsorg/syntagma/blob/main/docs/spec/tagma-sec.md

2 Security Model

2.1 Properties

Property	Module	Guarantee
Integrity	integrity	Tamper-evident binding of records to paths, principals, and epochs
Authorization	authority	Decisions over which principal may act on which scope
Audit	audit	Append-only, chained, externally verifiable evidence log
Non-repudiation	channel	Signed evidence exchange; origin and receipt provable to third parties
Confidentiality	hybrid (external)	Hybrid encryption composed with the envelope, not coordinate structure

2.2 Adversary Assumptions

The adversary can observe, capture, and replay any CoordPath and any envelope in transit. The adversary knows the composition formula, the linearization rule, and the validity bounds; these are public by design. The adversary does not hold the keys issued by the authority and cannot forge signatures under the active verification keys. The audit log storage layer is append-only; a log node may be compromised, but chaining makes silent retroactive modification detectable.

2.3 Trust Boundaries

The authority module is the trust root for principal identity and scope authorization. Epoch sources are shared between the integrity and channel modules; the concrete realization is an open question. The audit log and the channel module assume an authenticated time or epoch source for ordering.

3 Module Architecture

The layer is decomposed into four modules, each exposing a small interface. Scope matching follows the milestone 1 contract: Exact and Prefix rules over path sequences, with no dependence on path secrecy.

3.1 Authority

Registration, attestation issuance, scope authorization, and revocation. An attestation carries an issued epoch and a validity window; authorization checks the window and the epoch-scoped revocation record. Authorization reads the window, the scope, and the revocation record from the stored attestation, so a presented copy cannot extend or reshape the grant. Revocation records a revoked (principal, scope) pair effective from the recorded epoch.

3.2 Integrity

Seal, verify, and refresh. The seal is a keyed commitment that makes modification evident without hiding the record. The tagma-sec seal binds record, path, principal, and epoch, so replay across epochs is detected. Refresh re-binds an existing seal to a newer epoch.

3.3 Audit

Append, verify_chain, prove, and export. Every entry chains to its predecessor. Inclusion proofs and evidence bundles pair events with payloads so an external party can verify commitments without access to the log.

3.4 Channel

Sign, verify, exchange, and verify_receipt. Exchanged evidence binds the local payload, the remote principal, and the epoch, so replay of signed messages is prevented. Receipts prove delivery and origin.

4 Composition and Workflow

The proxy composes the four modules behind trait objects and switches between two implementations: the legacy pattern and the tagma-sec pattern. The route-update workflow exercises all four modules in sequence: authorization of the path and action at the current epoch, seal and verify of the record, audit commit of the record, channel receipt exchange binding the record to its path, epoch, and origin, and audit commit of the signed evidence.

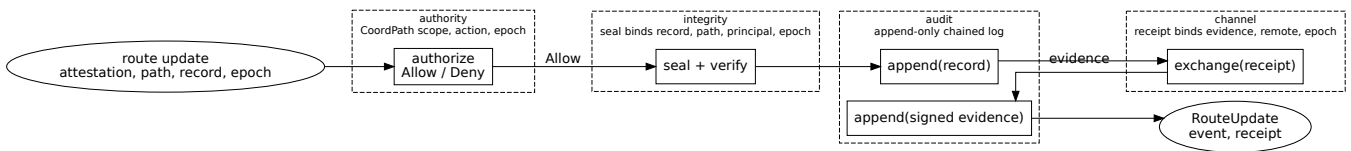


Figure 1: Route update workflow across the four tagma-sec modules.

5 Reverse Verification

The legacy pattern serves as the reverse-verification mirror: the same workflow suite runs identically against both stacks, so a security property holds for the tagma-sec pattern only if the same contract passes on the legacy implementation. Distinguishing tests pin the deltas: epoch and principal bound seals, refresh epoch binding, prefix scope coverage, revocation effective from the recorded epoch, scope-specific revocation keys, and the attestation lifetime boundary.

6 Benchmark Insights

Every security scenario in this document is pinned by a unit test and quantified by a benchmark. The measured values come from the sec criterion group in `sw/rust/benches/bench.rs` (ARMv8.4-A Firestorm 3.2 GHz). The benchmarks answer the scenario questions in cost terms: replay detection costs 33 ns per seal, fail-closed rejection costs 3.1 ns, silent log tampering is detectable by a ~1 ns per entry chain walk, and denial of origin is settled by a receipt that verifies in ~87 ns.

Scenario (guarantee)	Benchmark	Measured	Insight
Path authorization	authorize/allow	36.8 ns	Decision cost is O(scope depth), independent of store size
Fail-closed rejection	authorize/deny	3.1 ns	Scope miss short-circuits at the first coord, ~12x cheaper than Allow
Revoked scope enforcement	authorize/revoked	40.5 ns	Revocation is one map lookup on top of the scope match
Epoch replay detection	seal/pattern	171.4 ns	Epoch and principal binding costs ~33 ns over the 2-way seal (138.8 ns)

Scenario (guarantee)	Benchmark	Measured	Insight
Seal verification	verify/pattern	129.2 ns	Recompute and compare; refresh (296 ns) is verify plus seal
Audit chaining	audit/verify_chain (10k)	110.3 ns	~1 ns per entry prev-link walk, no hashing
Offline investigation	audit/prove, export (10k)	~200 μ s	Memory-bound clones, ~20 ns per entry, scales with evidence size
Non-repudiation signing	channel/sign, exchange	118 / 133 ns	Evidence composition adds ~15 ns over plain signing
Receipt verification	channel/verify_receipt	86.8 ns	Parsing plus recompute, same order as verify
Route update workflow	route_update/pattern	696.1 ns	Exact sum of module costs (691 ns), cost-transparent composition
Baseline contrast	route_update/legacy	699.5 ns	The tagma-sec pattern adds ~87 ns per update, the price of replay detection

6.1 Workflow Cost Transparency

The route-update workflow is a cost-transparent composition of the four modules. The measured total, 696.1 ns, equals the sum of its module calls: authorize 36.8, seal 171.4, verify 129.2, append 110.3, append 110.3, and exchange 132.7, totaling 690.7 ns. The proxy and the trait-object dispatch add no measurable overhead; the four-module composition costs exactly what its parts cost.

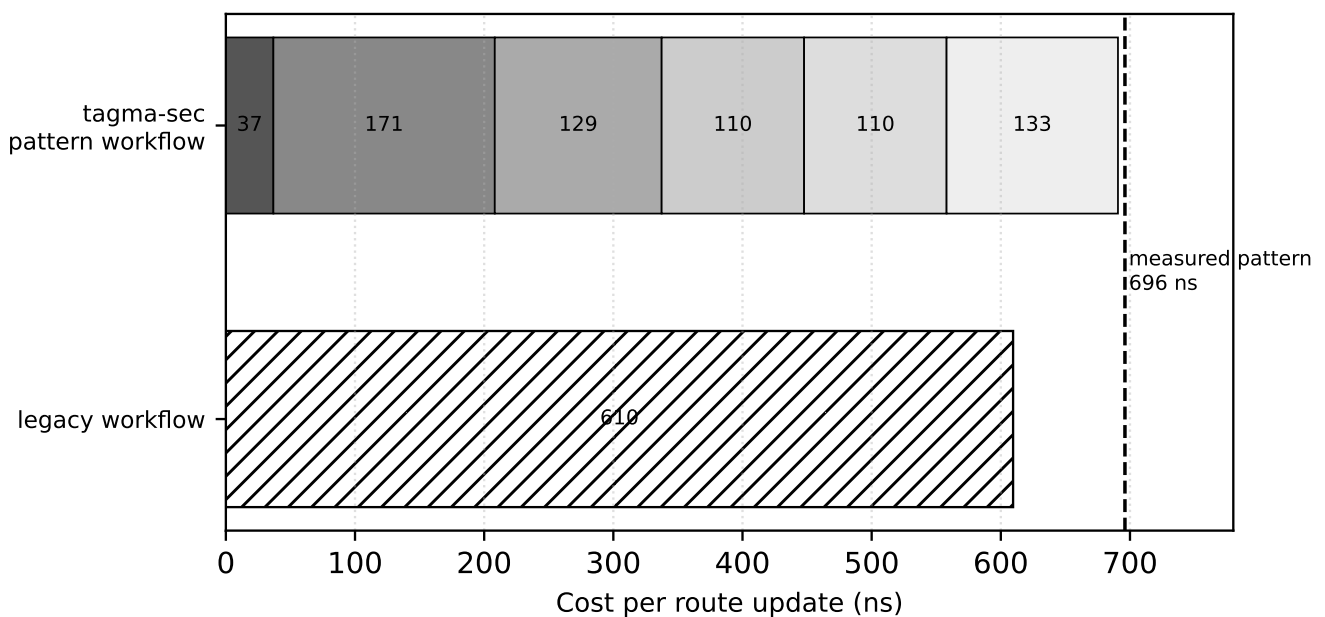


Figure 2: Route update cost: the tagma-sec pattern workflow equals the sum of its module costs

6.2 Authorization Decision Cost

Authorization is a structural predicate: a scope match compares coordinate sequences, so the decision cost is bounded by scope depth, not by store size. A 19-coord scope costs 62.7 ns against 36.0 ns for a 1-coord scope. The deny path short-circuits at the first mismatching coord and costs 3.1 ns, about 12x cheaper than acceptance, so fail-closed rejection is cheap. A revocation hit adds one map lookup on top of the scope match, 40.5 ns.

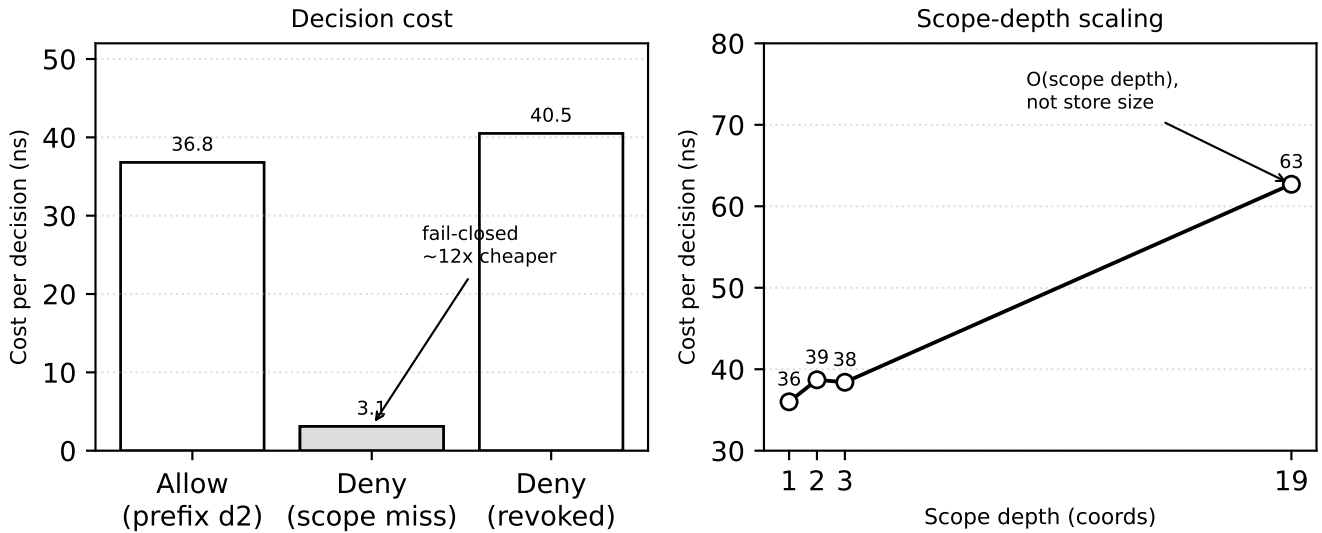


Figure 3: Authorization decision cost and scope-depth scaling

6.3 Verification Primitives

Coordinate arithmetic is negligible; the dominant cost everywhere is keyed hashing. Seal, append, sign, and exchange all sit in the 110 to 175 ns band, matching the specification's expectation that keyed hashing dominates end-to-end cost. The tagma-sec seal binds record, path, principal, and epoch, which costs 171.4 ns against 138.8 ns for the record-and-path legacy seal; the 33 ns difference is the price of epoch replay detection. Verification recomputes the seal at 129.2 ns, and refresh is verify plus seal at 296.0 ns. Channel operations stay in the same band: sign 117.6, exchange 132.7, verify and verify_receipt about 86 ns.

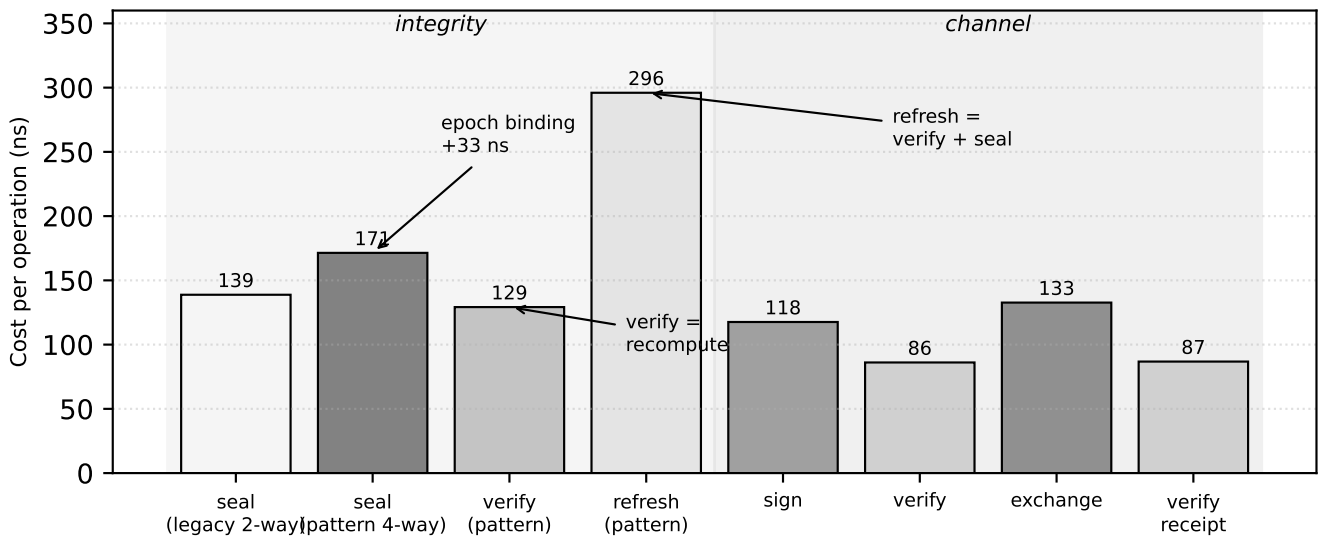


Figure 4: Verification primitives: integrity and channel costs

6.4 Audit Scaling

The audit log separates chaining from evidence. Appending one event costs 110.3 ns, a keyed payload hash plus two amortized pushes. Chain verification over 10,000 entries costs 9.9 μ s, about 1 ns per entry, because it is a prev-link pointer walk with no hashing. Offline investigation inverts the cost: prove and export clone the evidence range at about 20 ns per entry and are memory-bound, so investigation scales with evidence size rather than log size.

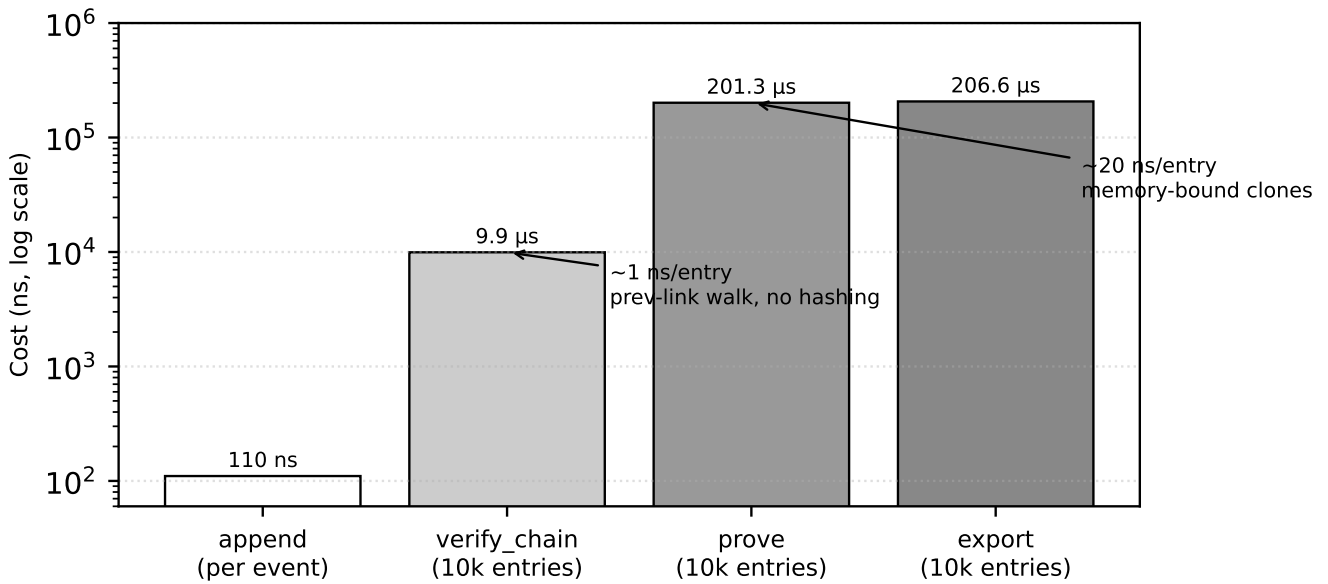


Figure 5: Audit operations: append cost and 10k-entry verification scale

7 Implementation Status

Milestones 1 to 4 (authority, integrity, audit, channel) are implemented in the synTagma repository³ under `sw/rust/sec`, with 32 integration tests (17 common workflow, 10 tagma-sec-specific, 5 real scenario) and a compiled README example. The `sec` criterion group in `sw/rust/benches/bench.rs` carries 21 security-layer benchmarks. Delos, the security-layer implementation at the same layer as the Chton IO layer, is in planning stage; it will compose these primitives with the neXus envelope and hybrid confidentiality. The initial implementation is a proof of concept: the channel is MAC-based and authenticates origin between holders of the shared key, PoC keys are hardcoded, paths are heap-backed vectors, and the concrete epoch source remains an open question.

References

³synTagma repository: Github (Pre-release, Apache 2.0)