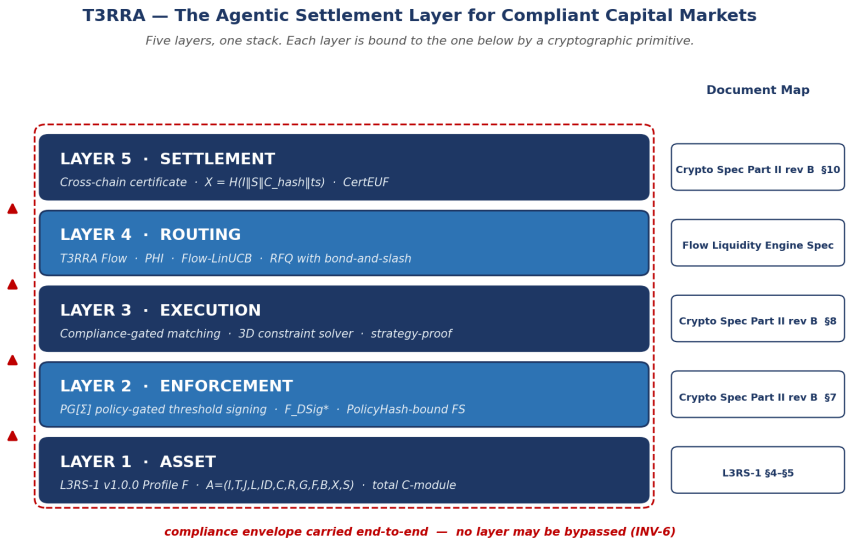
Cryptographic & Protocol Specification



T3RRA Cryptographic & Protocol Specification

Specification v1.0 rev B · April 2026

L3RS-1 v1.0.0 · Profile F (Full) Conformant

Zurab Ashvil · T3RRA Research

Contents

Foreword to rev B

This is the second revision of T3RRA's Cryptographic and Protocol Specification. Rev A described the cryptographic toolkit underlying the T3RRA Wallet — distributed key generation, threshold signing, proactive resharing, identifiable abort, hardware attestation — at the level of a thorough engineering specification. Rev B is a structurally different document. It treats T3RRA's four original constructions — policy-gated threshold signing, compliance-gated matching, the route admissibility predicate, and the cross-chain certificate — as cryptographic objects with formal definitions, security games, theorems, and proofs.

Rev B is written for the cryptographic community. It assumes familiarity with the universal composability framework of Canetti (FOCS 2001), the threshold ECDSA literature (Lindell 2017; GG18; CMP-NI; the DKLs sequence), the Schnorr threshold literature (FROST), and the standard market-microstructure model. Where this document is informal, it is informal on purpose, and the informality is labelled. Where this document gives a paper proof, it labels the proof as a paper proof and points to the mechanization roadmap in §15.

Three things in rev B did not exist in rev A. First, a formal definition of policy-gated threshold signing as an ideal functionality F_DSig*, together with a generic compiler PG[Σ] from any UC-secure threshold signature scheme to a realization of F_DSig*, and three theorems characterizing its security (policy soundness, unforgeability preservation, policy replay resistance). Second, a formal microstructure construction for compliance-gated matching with a strategy-proofness theorem that holds under continuous trading. Third, an unforgeability game for the L3RS-1 cross-chain certificate with a reduction to the EUF-CMA security of the underlying threshold signature, plus a Tamarin model stub for the cross-chain protocol that is being prepared as a public artifact.

Rev B has not been peer reviewed. Theorems are paper proofs. The mechanization in EasyCrypt and Tamarin is in progress and will be released as a public artifact when it is complete. Benchmarks are targets pending the public reproducibility harness scheduled for Q3 2026. We label every gap. We would rather under-promise and reproduce than over-claim and retract.

1. Notation and Preliminaries

> ***Reference convention.*** *Throughout this document,* ***§N*** *refers to* ***Section N of this Specification****; clicking it jumps there. References prefixed with* ***L3RS-1*** *— for example,* L3RS-1 §15 *— refer to the corresponding section of the external L3RS-1 v1.0.0 standard.*

Throughout this document, λ denotes the security parameter; all algorithms run in time polynomial in λ unless explicitly stated otherwise. We write negl(λ) for any function that vanishes faster than the inverse of every polynomial in λ. We write a ←$ S to denote sampling a uniformly at random from a finite set S, and a ← A(x) for the (possibly randomized) output of an algorithm A on input x. PPT abbreviates probabilistic polynomial time.

1.1 Groups and Curves

We work with two cyclic groups of prime order:

- G_secp = secp256k1 of order q_secp ≈ 2^256, with generator g_secp, used for ECDSA signatures on Ethereum, Bitcoin, and EVM L2s.
- G_ed = edwards25519 of order q_ed ≈ 2^252 + δ, with base point B_ed, used for EdDSA signatures on Solana and other ed25519-native chains.

Where the group is unambiguous, we write G with order q and generator g.

1.2 Hash Functions and Domain Separation

We model the following functions as random oracles, each with a distinct domain-separation string:

```
H_com    : {0,1}* → {0,1}^256      (commitment hash, dom "T3RRA-COM-v1")
H_FS     : {0,1}* → Z_q             (Fiat–Shamir, dom "T3RRA-FS-v1")
H_msg    : {0,1}* → Z_q             (message hash, chain-specific)
H_id     : {0,1}* → {0,1}^256      (asset identifier, dom "L3RS-1-AID-v1")
H_pol    : {0,1}* → {0,1}^256      (PolicyHash, dom "T3RRA-POLICY-HASH-v1")
H_cert   : {0,1}* → {0,1}^256      (cross-chain cert, dom "L3RS-1-CERT-v1")
```

Concrete instantiations are SHA-256 (NIST), SHA-3-256 (NIST), and Keccak-256 (Ethereum), selected per chain. The domain-separation strings are part of the specification and are immutable across the lifetime of an asset; changing a domain string is a hard fork.

1.3 Signature Schemes

We use Σ to denote a digital signature scheme (KeyGen, Sign, Verify). For ECDSA on secp256k1 we write Σ_ECDSA; for EdDSA on edwards25519 we write Σ_EdDSA. A (t,n)-threshold version of Σ is denoted Σ^{(t,n)} and is required to UC-realize the threshold ideal functionality F_DSig defined in §2.

1.4 Default Parameters

Symbol	Meaning	Default
(t, n)	Threshold and number of MPC parties for user wallets	(2, 3)
(t_b, n_b)	Threshold and number of bridge committee members	(5, 9)
τ_resh	Proactive resharing interval	90 days
dm(I)	Per-asset de minimis threshold for Travel Rule	$1,000 USD-equivalent
q_H	Random-oracle query bound for adversary	2^60
λ	Computational security parameter	128 bits

1.5 L3RS-1 Object Recap

We assume familiarity with L3RS-1 v1.0.0. We recall only the objects directly used in this specification:

- Asset object A = (I, T, J, L, ID, C, R, G, F, B, X, S) per L3RS-1 §4.
- Asset_ID I = H_id(pk_issuer ‖ ts ‖ nonce) per L3RS-1 §4.2.
- ComplianceModule C : ctx → {accept, reject} as a deterministic, total decision function realized by version-pinned bytecode whose hash is C_hash. We use C interchangeably to denote the function, the bytecode, and the version-pinned artifact, when context is clear.
- Cross-chain certificate X with the structure of L3RS-1 §10.2, formalized in §10 below.

2. Threat Model and the Ideal-Functionality Setting

2.1 Adversary Model

We consider a polynomial-time adversary A with the following capabilities. A may statically corrupt up to t−1 of the n MPC parties at protocol setup (for the user wallet, up to 1 of 3; for the bridge committee, up to 4 of 9). A may adaptively compromise non-share processes such as front-end interfaces, RPC endpoints, RFQ adapters, and browser extensions, with the constraint that compromise of these processes does not give A access to MPC keyshares. A controls the network including the relay between MPC parties and may delay, reorder, drop, or duplicate messages, but cannot break authenticated channels established between honest parties. A may issue arbitrary L3RS-1-conformant or non-conformant transfer requests through the public APIs. A is computationally bounded and cannot break the cryptographic primitives we instantiate (ECDSA on secp256k1, EdDSA on edwards25519, SHA-256, SHA-3-256, Keccak-256, ML-KEM-768), cannot extract keys from FIPS 140-3 Level 3 HSMs, and cannot forge hardware attestations rooted in vendor manufacturer keys (Apple, Google, AWS Nitro).

This is the standard dishonest-majority threshold model with static corruption and a network adversary, augmented by the L3RS-1-specific constraint that A may submit arbitrary transfer contexts.

2.2 The Ideal Functionality F_DSig

> **Definition 2.1 (F_DSig — Threshold Signature Ideal Functionality).** F_DSig is parameterized by a signature scheme Σ. On (KeyGen, sid) from at least t parties, F_DSig runs Σ.KeyGen and stores (sk, pk); it returns pk to all parties. On (Sign, sid, m) from at least t parties, F_DSig computes σ ← Σ.Sign(sk, m) and returns σ to the requesting parties. On (Verify, sid, m, σ, pk') from any party, F_DSig returns Σ.Verify(pk', m, σ). The functionality leaks no information about sk beyond what σ leaks.

F_DSig is the standard threshold-signature ideal functionality of Lindell (2017) and Canetti et al. (2020). A (t,n)-threshold protocol Π UC-realizes F_DSig if for every PPT adversary A there exists a PPT simulator S such that no PPT environment Z can distinguish the real-world execution of Π under A from the ideal-world execution with F_DSig under S, except with negligible probability in λ.

2.3 The Augmented Setting

T3RRA's wallet does not realize F_DSig. It realizes a strengthening F_DSig* defined in §7. The strengthening adds a deterministic compliance predicate to the signing query and an additional output condition (signature exists ⇒ predicate accepts). The construction in §7 is generic and applies to any Π that realizes F_DSig.

Figure 1. *Where DKG sits in the T3RRA cryptographic stack.*

3. Distributed Key Generation

3.1 DKG Goal

Distributed Key Generation produces, in a single ceremony, a public key pk known to all parties and a (t, n) Shamir secret-sharing of the corresponding secret key sk such that no proper subset of t−1 parties learns any information about sk. The DKG protocol must be a UC-secure realization of the F_KeyGen ideal functionality.

3.2 Protocol GG-DKG

We use the Gennaro–Goldfeder DKG protocol (CCS 2018, refined in subsequent work and integrated into CMP-NI) with Feldman Verifiable Secret Sharing. The protocol is six rounds in the standard model and three rounds in the random oracle model with non-interactive zero-knowledge proofs. The protocol is summarized informally below; the precise specification follows the Canetti–Makriyannis–Peled formulation.

```
GG-DKG(t, n) — informal:
  Round 1: Each P_i samples a_i,0 ←$ Z_q and additional
           coefficients a_i,1, ..., a_i,t-1 ←$ Z_q.
           P_i broadcasts commit_i = H_com(a_i,0 || ... || a_i,t-1 || r_i).
  Round 2: Each P_i broadcasts (a_i,0, ..., a_i,t-1, r_i)
           and Feldman commitments {g^{a_i,k}}_{k=0..t-1}.
           Each P_j verifies commit_i and the Feldman commitments.
  Round 3: P_i sends to P_j the share f_i(j) = sum_{k} a_i,k * j^k
           over an authenticated channel.
  Round 4: Each P_j verifies its share against the Feldman commitments
           via g^{f_i(j)} = prod_k (g^{a_i,k})^{j^k}.
  Round 5: P_i broadcasts ZK proof of knowledge of a_i,0.
  Round 6: All parties compute pk = prod_i g^{a_i,0} and
           sk_j = sum_i f_i(j). Output: pk to all, sk_j to P_j.
```

3.3 Properties

GG-DKG satisfies four properties: correctness — all honest parties output the same pk and consistent shares; secrecy — no PPT adversary corrupting up to t−1 parties learns any information about sk; robustness — the protocol terminates with output even in the presence of t−1 misbehaving parties; identifiability — any deviation by P_j is attributable to P_j by the honest parties (see §6).

3.4 Hardware-Attested DKG

In T3RRA's deployment, each P_i runs inside an attested hardware environment. P_i's first-round broadcast is augmented with a hardware attestation chain that links the running process to the manufacturer root key (Apple Secure Enclave, Android StrongBox, AWS Nitro hypervisor, or independent HSM). The attestation is recorded as part of the DKG transcript and is anchored as a LegalMirror artifact under L3RS-1 §13. Subsequent signing ceremonies re-verify each party's attestation chain before round 1 and abort if any chain fails to validate.

4. Threshold Signing — CMP-NI, DKLs23, FROST

4.1 CMP-NI for ECDSA

Our default ECDSA threshold signing protocol is CMP-NI, the non-interactive proactive UC-secure protocol of Canetti, Makriyannis, and Peled (CCS 2020). CMP-NI is selected because it is non-interactive in the preprocessing phase, supports identifiable abort, and is one of the few threshold ECDSA protocols with a complete UC security proof in the dishonest-majority model. The protocol uses Paillier encryption for blinded multiplication and a suite of zero-knowledge proofs to verify each party's contribution. We refer to the original paper for the full protocol and security proof.

4.2 DKLs23 for ECDSA

For deployments where bandwidth is the binding constraint we instantiate DKLs23 (Doerner, Kondi, Lee, shelat, IEEE S&P 2024). DKLs23 builds on the DKLs18 / DKLs19 line and achieves three-round signing with security against an active adversary in the dishonest-majority model. The protocol uses oblivious transfer extension rather than Paillier encryption, which significantly reduces bandwidth at the cost of a longer one-time setup.

4.3 FROST for EdDSA

For ed25519 chains we instantiate FROST (Komlo and Goldberg, SAC 2020). FROST is a two-round Schnorr-based threshold signing protocol that produces standard Schnorr signatures verifiable by any RFC 8032 EdDSA verifier. FROST is the natural choice for ed25519 because Schnorr signatures compose cleanly under threshold, unlike ECDSA where the multiplicative inversion structure forces more complex constructions.

4.4 Round and Bandwidth Summary

Protocol	Curve	Online Rounds	Preprocessing	Bandwidth (per signer, per signature)
CMP-NI	secp256k1	3	Non-interactive	12 KB
DKLs23	secp256k1	2 (after preproc)	Reusable OT extension	3 KB online
FROST	edwards25519	2	None	1.2 KB

4.5 Selection Criterion

T3RRA selects the signing protocol per asset and per chain. The default is CMP-NI for EVM and Bitcoin, FROST for ed25519, and DKLs23 for high-throughput EVM operations where bandwidth dominates. The selection is part of the platform configuration and is auditable.

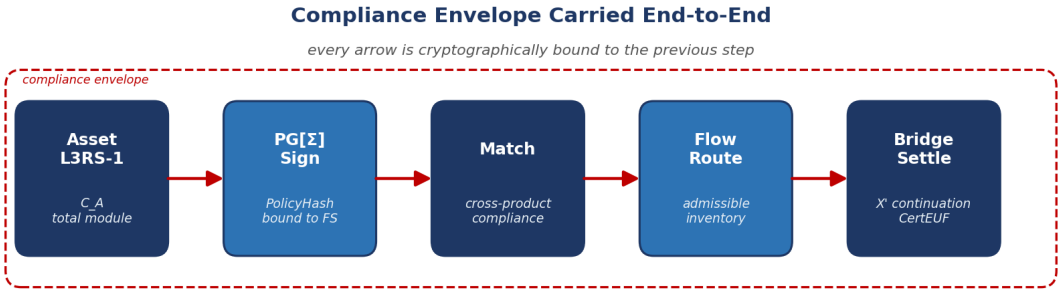


Figure 2. *Proactive resharing — share continuity without key change.*

5. Proactive Resharing

5.1 The Mobile Adversary

In the mobile-adversary model of Ostrovsky and Yung (PODC 1991), an adversary may corrupt different sets of parties in different epochs, with the constraint that the total number of corrupted parties at any single epoch never exceeds t−1. Without proactive resharing, an adversary that corrupts P_1 in epoch 1 and then moves to P_2 in epoch 2 eventually accumulates t shares and reconstructs sk. Proactive resharing defeats this by re-randomizing the share polynomial at the start of each epoch — the public key pk remains the same, but old shares become useless.

5.2 Refresh Protocol

T3RRA's refresh is a five-round protocol that generates a fresh sharing of zero and adds it to the existing sharing of sk. Informal sketch:

```
Refresh — informal:
  Round 1: Each P_i samples a fresh degree-(t-1) polynomial
           h_i(x) with h_i(0) = 0 and broadcasts H_com(h_i || r_i).
  Round 2: Each P_i broadcasts h_i and Feldman commitments
           to its coefficients (constant term must be the identity).
  Round 3: Each P_i sends h_i(j) to P_j over an authenticated channel.
  Round 4: Each P_j verifies the shares received against the
           Feldman commitments. If any verification fails,
           ABORT(IdentifiableRefresh, P_i).
  Round 5: Each P_j updates sk_j ← sk_j + sum_i h_i(j) and
           securely erases prior shares and refresh polynomials.
```

5.3 Forward Security

> **Theorem 5.1 (Forward Security under Refresh).** *Under the assumption that the underlying threshold scheme Σ^{(t,n)} is UC-secure under static corruption, the protocol Π = (DKG, Sign, Refresh) above is secure against a mobile adversary that corrupts at most t−1 parties in any single epoch, where epochs are delimited by Refresh executions.*

Proof sketch. By the standard reduction to Ostrovsky–Yung. A mobile adversary that accumulates shares across epochs is reduced to a static adversary at a single epoch by the property that shares from prior epochs are statistically independent of shares in the current epoch (since the refresh polynomial is uniform with constant term zero). The verifiable nature of Feldman commitments ensures that any deviation in step 2 or step 4 is detected and identified, preventing an adversary from forcing the protocol into an inconsistent state. □

5.4 Refresh Schedule

T3RRA's default refresh schedule is τ_resh = 90 days for production user wallets and 30 days for the bridge committee. Refresh is automatic and does not require user action. Refresh failures abort the refresh, leave the prior sharing in place, and surface an alert to the operations team and to the user.

6. Identifiable Abort and the AbortReport Object

6.1 The Need for Identifiability

In many threshold protocols a deviating party can cause the protocol to abort without being identified, leading to denial-of-service attacks where a single malicious party silently breaks every signing ceremony. Identifiable abort is the property that any deviation by a party P_j is detected and attributed to P_j by every honest party, enabling P_j to be removed from the quorum and added to a reputation registry.

6.2 The AbortReport Object

Definition 6.1 (AbortReport). An AbortReport is a structured artifact emitted by an honest party upon detection of a protocol deviation. It has the form: AbortReport = (sid, P_accused, reason_code, evidence, signatures), where sid identifies the protocol session, P_accused is the identifier of the deviating party, reason_code is drawn from a fixed enumeration {InvalidCommit, InvalidShare, InvalidZK, MalformedMessage, ProtocolTimeout, InvalidPolicy, ComplianceReject, TravelRuleAbort, AttestationFailure}, evidence is a self-contained transcript that allows any third party to verify the accusation, and signatures is a set of attestations from honest parties confirming the accusation.

6.3 ComplianceAbort and TravelRuleAbort

Two reason codes are specific to T3RRA's policy-gated layer. ComplianceAbort is emitted by an honest party that locally evaluates b ← C(ctx) = reject in the policy-gated signing prelude (§7.3); the reason field contains a structured citation of the L3RS-1 clause violated. TravelRuleAbort is emitted when the Travel Rule pre-sign check (§13) fails; the reason field contains the failing condition (no VASP discovery, invalid receipt, or invalid attestation chain).

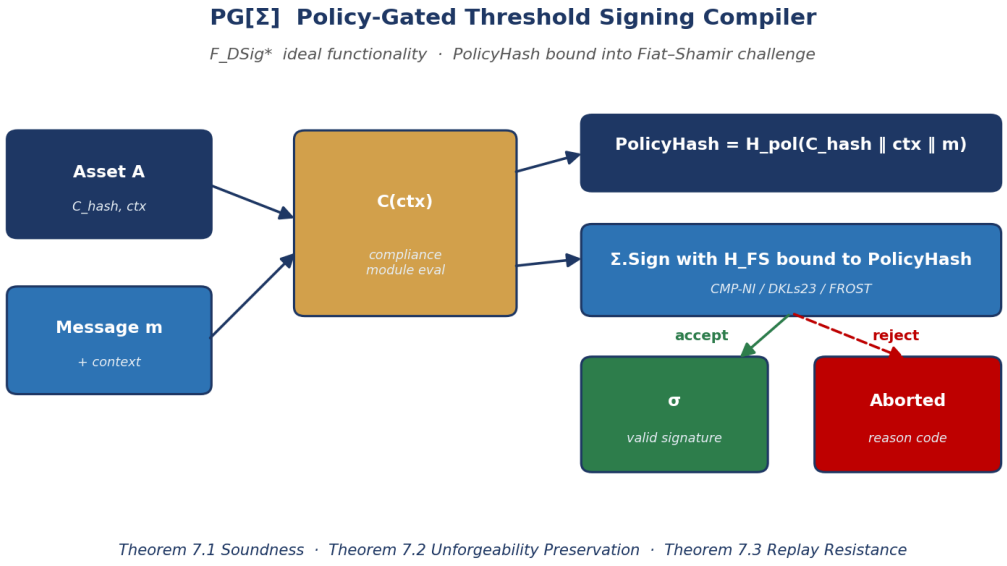


Figure 3. *Policy-Gated Threshold Signing PG[Σ] — PolicyHash bound into the Fiat–Shamir challenge.*

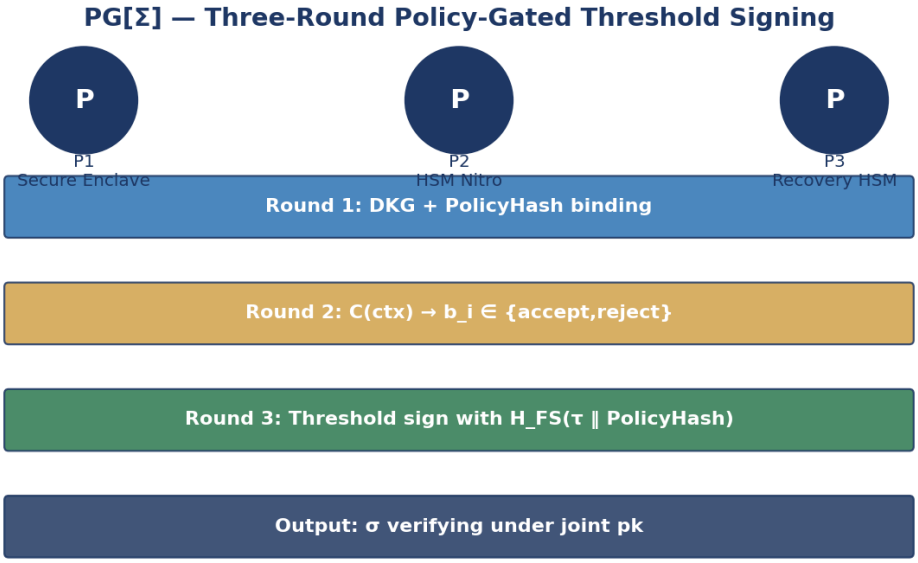


Figure 4. *PG[Σ] three-round message flow across the (2,3) MPC committee.*

7. Policy-Gated Threshold Signing

This is the central original construction of rev B. We define a strengthening of F_DSig that binds compliance into the signing functionality, give a generic compiler from any UC-secure threshold scheme to a realization of the strengthening, and prove three theorems.

7.1 The Policy-Gated Ideal Functionality F_DSig*

Definition 7.1 (F_DSig*). F_DSig* is parameterized by a signature scheme Σ and a deterministic, total predicate C : {0,1}* → {accept, reject, reason}, identified by the hash C_hash = H_com(bytecode(C)). KeyGen: identical to F_DSig. Sign: on input (Sign, sid, m, ctx, C_hash) from at least t parties, F_DSig* first verifies that H_com(bytecode(C)) = C_hash; if not, it ignores the request. F_DSig* then evaluates b ← C(ctx); if b = reject, F_DSig* outputs (Aborted, sid, reason) to all parties and halts the session. Only if b = accept does F_DSig* compute σ ← Σ.Sign(sk, m) and return σ to the requesting parties. Furthermore, F_DSig* binds the signing query to the policy: it computes PolicyHash = H_pol(C_hash ‖ ctx ‖ m) and parameterizes the signature transcript by PolicyHash as a domain separator in every random-oracle query made during signing. Verify: identical to F_DSig. Leakage: F_DSig* leaks no information about sk beyond what σ leaks. F_DSig* leaks (C_hash, ctx) to the adversary at the moment of the Sign query — this is unavoidable, since in any realization the parties must learn ctx in order to evaluate C.

7.2 The Generic Compiler PG[·]

We give a generic compiler that turns any UC-secure threshold signature scheme Σ^{(t,n)} realizing F_DSig into a scheme PG[Σ]^{(t,n)} realizing F_DSig*. The compiler is intentionally minimal: it does not modify the signing protocol's communication structure; it adds a deterministic prelude that every party runs locally before contributing any round-1 message, and it modifies the Fiat–Shamir challenge of every zero-knowledge proof inside the signing protocol to bind PolicyHash.

7.3 The PG Compiler

```
Compiler PG[Σ](C, C_hash) — formal specification
================================================

On (KeyGen, sid) from at least t parties:
  Run Σ.KeyGen unmodified. Output pk to all, sk_j to P_j.

On (Sign, sid, m, ctx) from party P_i:
  // Step 1 — policy authenticity
  P_i checks H_com(bytecode(C)) = C_hash.
  If not, P_i emits AbortReport(sid, P_i, InvalidPolicy, evidence).
  P_i refuses to participate. Halt for P_i.

  // Step 2 — local compliance evaluation
  b_i ← C(ctx).
  If b_i = reject:
    P_i emits AbortReport(sid, P_i, ComplianceReject, reason_i).
    P_i refuses to participate in round 1.
    Halt for P_i.

  // Step 3 — PolicyHash computation
```

```
PolicyHash_i = H_pol(C_hash ‖ canonical(ctx) ‖ m).
// canonical(ctx) is the L3RS-1 canonical encoding of ctx.

// Step 4 — modified signing protocol
P_i runs Σ.Sign with the following modification:
  every Fiat–Shamir challenge c in any ZK proof inside Σ
  is computed as
    c = H_FS(transcript ‖ "PG-bind" ‖ PolicyHash_i)
  instead of
    c = H_FS(transcript).
// Note: every honest party computes the same PolicyHash_i
// because (C_hash, ctx, m) are common inputs.

// Step 5 — output
Output σ on success.
```

7.4 Theorem: Policy Soundness

Theorem 7.1 (Policy Soundness). *Let C be a deterministic total predicate with hash C_hash, and let $\Sigma\hat{}\{(t,n)\}$ be a (t,n)-threshold signature scheme that UC-realizes F_DSig under static corruption with at most $t-1$ corrupted parties. Then $PG[\Sigma]\hat{}\{(t,n)\}$ UC-realizes F_DSig* with the same corruption bound. In particular, if any honest party P_i computes b_i = reject in Step 2, no signature σ verifying under the joint public key pk on input (m, ctx, C_hash) can be produced by any subset of t parties of which P_i is a member, except with negligible probability in λ.*

Proof sketch. We prove the policy-soundness clause; the full UC realization follows by standard simulator construction relaying messages between F_DSig and F_DSig*. Suppose for contradiction that an honest party P_i computes b_i = reject in Step 2 yet a verifying signature σ on (m, ctx, C_hash) under PolicyHash exists, output by a quorum that includes P_i. Since C is deterministic, every honest party that runs Step 2 on the same ctx computes the same value, hence b_j = reject for every honest j. By assumption Step 2 causes P_i to refuse all round-1 messages, so the protocol transcript at honest parties contains no round-1 message from P_i. $\Sigma\hat{}\{(t,n)\}$ requires t valid round-1 contributions to produce a signature; in the dishonest-majority model with at most $t-1$ corruptions and $t = (t,n)$ with t honest parties, the absence of one honest contribution leaves at most $t-1$ contributions, which is insufficient. Therefore no σ exists, contradicting the assumption. The negligible-probability term accounts for the (negligible) probability that the random oracle H_FS produces a collision in PolicyHash that masks the absence of P_i's contribution. □

7.5 Theorem: Unforgeability Preservation

Theorem 7.2 (Unforgeability Preservation). *If $\Sigma\hat{}\{(t,n)\}$ is EUF-CMA secure as a threshold signature scheme realizing F_DSig, then $PG[\Sigma]\hat{}\{(t,n)\}$ is EUF-CMA secure as a threshold signature scheme realizing F_DSig*. Concretely, for any PPT adversary A against $PG[\Sigma]$ making q_S signing queries and q_H random-oracle queries, there exists a PPT adversary B against Σ such that $Adv\hat{}EUF\text{-}CMA(A, PG[\Sigma]) \le Adv\hat{}EUF\text{-}CMA(B, \Sigma) + q_H \cdot 2\hat{}\{-\lambda\} + q_S \cdot 2\hat{}\{-\lambda\}$, with B's running time approximately equal to A's plus $O(q_S \cdot |C|)$ for compliance evaluation and $O(q_H)$ for random-oracle programming.*

Proof sketch. By reduction. Given B against Σ, we construct A against PG[Σ] as a wrapper. A receives pk from its Σ challenger and forwards pk to its environment. For each signing query (m, ctx) from the environment, A first evaluates b ← C(ctx); if b = reject, A returns Aborted to the environment. Otherwise A queries its Σ-oracle on m to obtain σ, computes PolicyHash = H_pol(C_hash ‖ ctx ‖ m), programs the random oracle H_FS on the relevant transcript points to be consistent with the Fiat–Shamir challenges that would have been produced in PG[Σ], and returns σ to the environment. Random-oracle programming can fail if A's environment has previously queried H_FS on the same transcript, which happens with probability at most q_H · 2^{-λ}. Each signing query may also fail if a collision in PolicyHash matches an unrelated query, with probability at most q_S · 2^{-λ}. A forgery against PG[Σ] yields a forgery against Σ on the corresponding (m, ctx) tuple via the canonical encoding of ctx into the message space, completing the reduction. The bound follows. □

7.6 Theorem: Policy Replay Resistance

Theorem 7.3 (Policy Replay Resistance). *A signature σ produced under PG[Σ] with parameters (C_hash, ctx, m) is computationally unforgeable as a signature for any (C_hash', ctx', m') ≠ (C_hash, ctx, m). In particular, an upgrade of the ComplianceModule that produces a new C_hash invalidates all prior signatures as authorizations under the new policy: any verifier that re-derives PolicyHash on the new C_hash and re-checks the Fiat–Shamir transcript will reject.*

Proof sketch. Direct from the random-oracle property of H_pol over PolicyHash and the binding of PolicyHash into H_FS. Changing any of (C_hash, ctx, m) yields a uniformly fresh PolicyHash with probability 1 − 2^{-256}, which in turn yields uniformly fresh Fiat–Shamir challenges. A signature transcript that verifies under one set of challenges fails to verify under a different set with probability 1 − negl(λ). □

7.7 Instantiations

We instantiate PG[Σ] over three concrete threshold protocols:

7.7.1 PG[CMP-NI] for ECDSA

CMP-NI's online signing phase contains four zero-knowledge proofs that are converted to non-interactive form via Fiat–Shamir: (i) the Paillier-encryption knowledge proof, (ii) the discrete-log proof for the nonce share, (iii) the multiplication-consistency proof, and (iv) the range proof on the partial signature share. Each of these four proofs has its Fiat–Shamir challenge replaced by H_FS(transcript ‖ "PG-bind" ‖ PolicyHash). The protocol's round structure, identifiable abort, and proactive resharing properties are preserved.

7.7.2 PG[DKLs23] for ECDSA

DKLs23's online phase contains the OT-extension consistency proof and the partial-signature consistency proof. Both have their Fiat–Shamir challenges modified per the compiler. The DKLs preprocessing phase is independent of any specific (m, ctx) and is therefore unmodified by the compiler — it can be reused across multiple signing ceremonies, each binding to its own PolicyHash at online-phase commencement.

7.7.3 PG[FROST] for EdDSA

FROST has a single Schnorr challenge per signature. The challenge c = H_msg(R ‖ pk ‖ m) is replaced by c = H_msg(R ‖ pk ‖ m ‖ "PG-bind" ‖ PolicyHash). The output signature remains a standard Schnorr signature verifiable by any RFC 8032 EdDSA verifier — the modification is invisible to verifiers because

it lives inside the challenge function, which standard EdDSA verifiers compute themselves from (R, pk, m). For T3RRA's policy-bound verification, an additional verifier that re-derives PolicyHash and recomputes the challenge is used in tandem with the standard verifier. Honest verifiers that wish to enforce the policy binding use the additional verifier; chain-level verification remains compatible with standard ed25519.

7.8 PolicyHash as a Cryptographic Object

PolicyHash is the central object connecting compliance to cryptography. Three properties of PolicyHash are worth stating explicitly. First, it is computable by anyone who knows (C_hash, ctx, m): there is no secret in its computation, so any verifier — auditor, regulator, counterparty — can independently compute it. Second, it is collision-resistant under the random-oracle assumption on H_pol: distinct (C_hash, ctx, m) tuples produce distinct PolicyHash values with probability 1 − negl(λ). Third, it binds the signature to a specific compliance evaluation, which means a signature produced under PolicyHash_1 cannot be replayed against an asset whose evaluation produces PolicyHash_2.

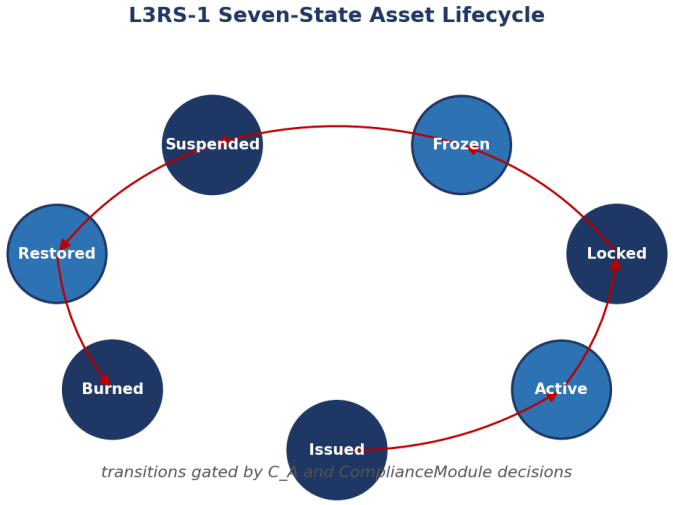


Figure 5. *L3RS-1 lifecycle — every state transition gated by the same compliance predicate C.*

8. Compliance-Gated Matching

8.1 The Standard CLOB Model

In the standard market-microstructure literature an order is a tuple $\omega = (\text{side}, \text{price}, \text{size})$. A central limit order book (CLOB) maintains, for a given asset, two sorted price levels (bids and asks) with FIFO time priority within each level. The matching engine processes events sequentially; at each event time, if the best bid $\geq$ best ask, the engine produces a fill against the resting order at the better timestamp. This standard model is compliance-blind: the engine has no awareness of who is on either side of a fill.

8.2 The Compliance-Gated Order

Definition 8.1 (Compliance-Gated Order). A compliance-gated order on T3RRA is a tuple $\omega = (\text{side}, \text{price}, \text{size}, \text{I}, \text{pk}, \text{vc}, \text{j}, \pi)$ where I is the L3RS-1 Asset_ID, pk is the order owner's public key bound to its T3RRA wallet, vc is the verifiable credential issued by T3RRA's KYC partner attesting the owner's identity tier, j is the owner's declared jurisdiction (drawn from ISO 3166-1), and π is a proof that at order placement time C_I(ctx_self) = accept, where ctx_self is the owner's self-context against the asset's ComplianceModule. The order is admitted to the book only if Verify(π) succeeds and (pk, vc) is a valid binding under the KYC issuer's signing key.

8.3 The Three-Way Matching Rule

Definition 8.2 (Compliance-Gated Match). Given two compliance-gated orders $\omega\_b = (\text{buy}, \text{p\_b}, \text{s\_b}, \text{I}, \text{pk\_b}, \text{vc\_b}, \text{j\_b}, \pi\_b)$ and $\omega\_s = (\text{sell}, \text{p\_s}, \text{s\_s}, \text{I}, \text{pk\_s}, \text{vc\_s}, \text{j\_s}, \pi\_s)$ with p_b $\geq$ p_s, the matching engine constructs the prospective transfer context ctx = (pk_s, pk_b, I, min(s_b, s_s), (j_s, j_b), (vc_s, vc_b), travel_rule_payload, ts, nonce) and evaluates b $\leftarrow$ C_I(ctx). The match commits if and only if b = accept. If b = reject, the engine emits a structured reject reason mapped to an L3RS-1 clause and the orders remain on the book in their original positions.

8.4 Strategy-Proofness

Theorem 8.1 (Strategy-Proofness Under Honest Compliance). *Assume (i) price priority and FIFO time priority within levels, (ii) the matching engine is honest and does not front-run, (iii) C_I is deterministic and total, and (iv) the KYC issuer's signing key is unforgeable in the sense of EUF-CMA. Then for any honest counterparty P with true credential vc and true jurisdiction j: P's expected fill outcome under truthful submission (vc, j) is at least as good as P's expected fill outcome under any other strategy (vc', j') with (vc', j') $\neq$ (vc, j).*

Proof sketch. We show that no deviation strictly improves P's outcome. Under matching rule 8.2, P's outcome on any prospective match depends only on the value of C_I(ctx) at fill time, where ctx is constructed by the matching engine from the credentials on file. If P submits (vc', j') $\neq$ (vc, j), one of three cases applies. Case 1 — vc' is forged: by EUF-CMA of the KYC issuer's signing key, vc' is rejected at order admission with probability $1 - \text{negl}(\lambda)$, so the order is not admitted and the deviation strictly worsens P's outcome (no fills). Case 2 — vc' is a credential issued to a different account pk': any fill accrues to pk', not to pk, so P does not benefit. Case 3 — j' is misrepresented but vc is truthful: the credential vc binds the jurisdiction declared at issuance, so the matching engine constructs ctx with the bound jurisdiction; the misrepresentation has no effect on C_I(ctx). In all three cases the deviation does not improve P's outcome. □

8.5 Strategy-Proofness Under Continuous Trading

Theorem 8.2 (Continuity). *Theorem 8.1 holds under continuous trading: for any sequence of orders {ω_1, ω_2, …, ω_n} processed by the matching engine, no honest counterparty can improve its outcome on order ω_k by deviating in any prior order ω_1, …, ω_{k−1}.*

Proof sketch. Each order in the sequence is bound to the counterparty's credential at submission time, and the credential cannot be silently modified without re-onboarding. Re-onboarding produces a new pk', and any fills under (pk', vc') accrue to pk' rather than to the original pk, so the deviation does not improve the original P's outcome on ω_k. The argument of Theorem 8.1 then applies independently at each k. □

8.6 Sanctioned-Address Refusal

A practical consequence of Theorems 8.1 and 8.2 is the property regulators most often request: a sanctioned address cannot fill on T3RRA, even if it bids the best price. Concretely, if a counterparty's credential vc is updated by the KYC issuer to flag sanctioned status (e.g., because the address appears on an OFAC SDN list), the next match attempt against that counterparty constructs ctx with the sanctioned-status flag, and any reasonable C_I returns reject. The order remains visible on the book but unfillable. There is no path around the matching engine to settlement, because the only path is through the policy-gated wallet signature, which is also gated by the same C_I via §7.

8.7 Compliance-Gated RFQ

RFQ matching is a variant of the above with one important difference: in RFQ, the counterparty receives a quote that is contingent on its credentials. In T3RRA's compliance-gated RFQ, the quote is computed only if C_I(ctx) returns accept on the requesting party's context, with the contra side being the RFQ desk. If C_I returns reject, the requesting party receives a structured reject reason and no quote. This prevents quote-leakage attacks where a sanctioned party uses RFQ as a probe to learn pricing.

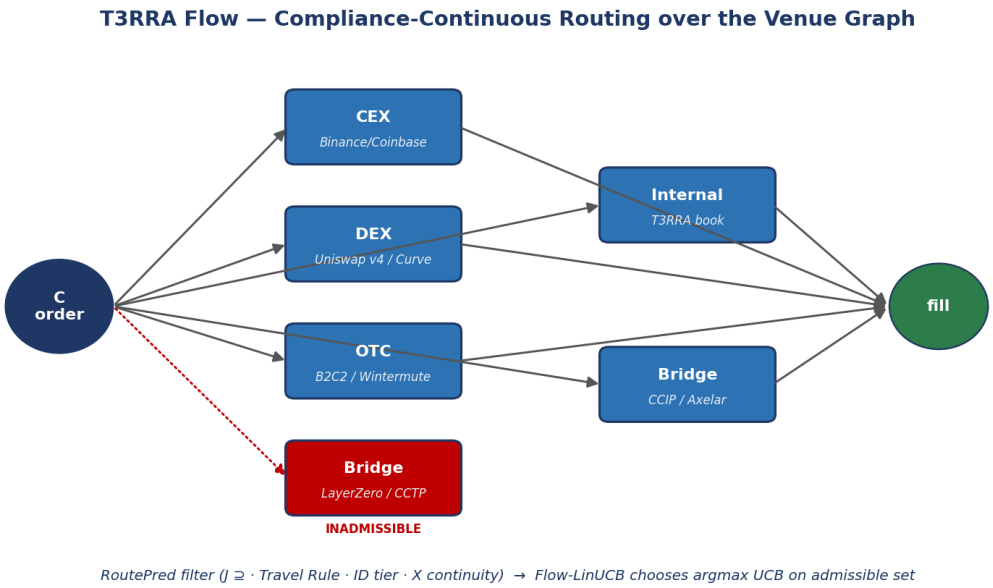


Figure 6. *Venue graph filtered by the admissibility predicate.*

9. The Route Admissibility Predicate

9.1 The Venue Graph

Let G = (V, E) be a directed multigraph where V is the set of venues and E is the set of swap edges. A vertex v ∈ V is a (venue, asset) pair: a single venue may appear as multiple vertices, one for each asset it lists. An edge e ∈ E from u = (venue_u, asset_u) to v = (venue_v, asset_v) represents the ability to execute a swap from asset_u at venue_u to asset_v at venue_v under quality-of-execution parameters that we annotate as edge labels. The label on e is a tuple (depth, fee, latency, jurisdictions, vasp_id, certificate_capable, identity_tier_required), each field with the obvious interpretation.

9.2 Route

Definition 9.1 (Route). A route R from a source asset I_s to a destination asset I_d is a finite sequence of edges R = (e_1, e_2, …, e_k) in G such that the head of e_1 has asset = I_s, the tail of e_k has asset = I_d, and the head of e_{i+1} equals the tail of e_i for all i ∈ [1, k−1]. The size σ_i carried on hop e_i is determined by the trade size, the upstream slippage on e_1, …, e_{i−1}, and the depth of e_i.

9.3 The Predicate

Definition 9.2 (Route Admissibility Predicate). Let I be the asset traded, s the originating party, r the beneficiary, σ the trade size, and R = (e_1, …, e_k) a route. R is admissible for (I, s, r, σ) if and only if the following four conjuncts hold for every edge e_i ∈ R: (J) jurisdictions(e_i) ⊆ J(I) — every jurisdiction the hop touches is included in the asset's jurisdictional mask. (T) σ_i < dm(I) ∨ travel_rule_ok(e_i, σ_i) — either the hop is below the asset's de minimis threshold, or the hop returns a verifiable Travel Rule receipt over the IVMS 101 payload constructed from (s, r, σ_i, e_i). (ID) identity_tier_required(e_i) ≤ min(tier(s), tier(r)) and ≥ ID(I) — every party at the hop satisfies both the asset's minimum identity tier and the hop's own minimum identity tier. (X) cross_chain(e_i) ⟹ certificate_continuity(e_i, e_{i−1}) — every cross-chain hop carries a continuation of the L3RS-1 certificate from the prior hop, where certificate_continuity is defined in §10.

```
admissible(R, I, s, r, σ) ⇔ ∀ e_i ∈ R . (J)(e_i) ∧ (T)(e_i) ∧ (ID)(e_i) ∧ (X)(e_i)
```

9.4 Tractability

Theorem 9.1 (Tractability). *Route discovery under the admissibility predicate over a venue graph G = (V, E) admits an algorithm running in time O((|E| + |V| log|V|) · |Cert|), where |Cert| is the number of distinct cross-chain certificate states reachable in G. For the typical T3RRA configuration with |V| ≈ 500, |E| ≈ 5000, and |Cert| ≤ 4, this evaluates to a handful of milliseconds.*

Proof sketch. The admissibility predicate decomposes per-hop with a single one-step dependency: conjuncts (J), (T), and (ID) depend only on the hop itself, while conjunct (X) depends on the hop and the immediately preceding hop's certificate state. We construct a state-augmented graph G' whose vertices are pairs (v, cert) ∈ V × Cert, where Cert is the set of possible certificate states (typically: {none, src_chain_X, dst_chain_X}). For each edge e in G we create edges in G' that respect the certificate-continuity transition: if e is intra-chain, the certificate state is unchanged; if e is a cross-chain hop, the state advances per the L3RS-1 §10 transition function.

We mark each edge in G' as admissible if conjuncts (J), (T), (ID) hold (a O(1) per-edge check given precomputed jurisdiction masks and tier metadata) and the (X) transition is valid. We assign cost ∞ to inadmissible edges and price $\cdot$ gas $\cdot$ latency to admissible ones. Dijkstra's shortest-path algorithm with a Fibonacci heap on G' then finds the minimum-cost admissible route in $O(|E'| + |V'| \log|V'|)$ where $|V'| = |V| \cdot |\text{Cert}|$ and $|E'| = |E| \cdot |\text{Cert}|$. The bound follows. $\square$

9.5 Online Updates

The venue graph is updated continuously as venues come online, change their compliance posture, change their depth, or are reviewed by the DAO. The Dijkstra search is re-run on demand for each quote; we do not memoize routes across quotes because the underlying graph is too volatile. For high-throughput operation we maintain a per-asset-pair shortlist of the top-k admissible routes from the most recent quote and revalidate them in parallel for the new quote.

9.6 Integration with the Bandit

Within the admissible subgraph, route ranking is performed by a contextual multi-armed bandit. The bandit's arms are admissible routes; its context is the trade parameters (size, urgency, investor profile, asset class); its reward is realized fill quality (slippage vs midpoint, time-to-fill, completion rate). The bandit operates strictly within the admissible subgraph: exploration capital is allocated only to admissible routes. The DAO sets hard caps on the maximum exploration fraction per trade.

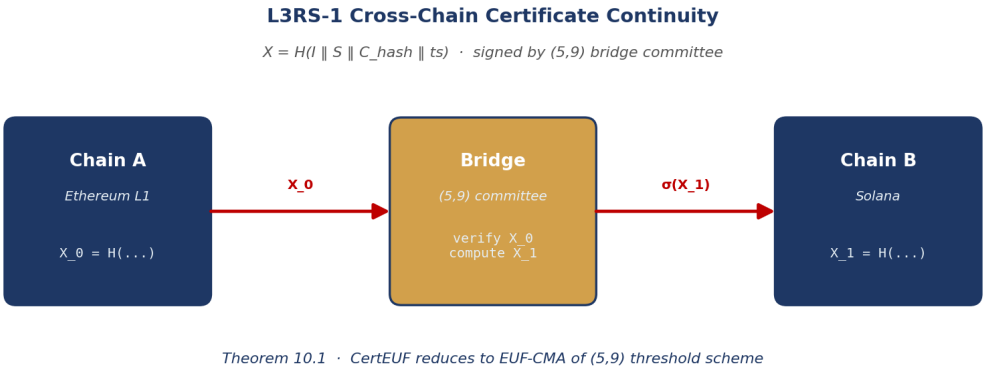


Figure 7. *Cross-chain certificate continuity across the (5,9) bridge committee.*

10. Cross-Chain Certificate Unforgeability

10.1 The Certificate

Definition 10.1 (Cross-Chain Certificate). A T3RRA cross-chain certificate is a tuple X = (I, S, C_hash, ts, src_chain, dst_chain, dst_addr, nonce, σ_quorum) where I is the L3RS-1 Asset_ID, S is the lifecycle state at source at time ts, C_hash is the ComplianceModule hash at source at time ts, ts is the source-chain timestamp, src_chain and dst_chain are chain identifiers, dst_addr is the destination address, nonce is a fresh nonce drawn from a per-asset namespace, and σ_quorum is a (t_b, n_b)-threshold signature over the canonical encoding canonical(X \ σ_quorum) produced by the bridge committee.

10.2 The Seven Invariants

#	Invariant	Implementation Mechanism
I1	Identity continuity: I unchanged across hops	Asset_ID is the primary key in every chain's mirror contract
I2	State continuity: S(dst) = S(src)	State diff signed by source committee, verified on dst
I3	Compliance continuity: C_hash unchanged	Source C_hash bound into σ_quorum; dst rejects mismatch
I4	Reserve continuity: ΣSupply_chains = R	Daily attestation reconciliation across mirror contracts
I5	Lock-mint or burn-mint, never both	Per-asset bridge mode immutable at issuance
I6	No silent re-minting	Every mint requires verifiable burn or lock witness
I7	Replay protection on certificate consumption	Per-asset nonce namespace tracked in dst registry

10.3 The CertEUF Game

Game 10.1 (CertEUF). The certificate-unforgeability game CertEUF is played between a challenger Ch and an adversary A. Setup: Ch runs KeyGen for the bridge committee with parameters (t_b, n_b), producing 9 key shares (sk_1, …, sk_9) and a joint public key pk. Ch sends pk to A. A statically corrupts a set Q ⊆ {1, …, 9} with |Q| ≤ t_b − 1 = 4 and receives the corresponding shares. Query phase: A submits up to q signing queries of the form X_j = (I_j, S_j, C_hash_j, ts_j, src_j, dst_j, addr_j, nonce_j) and receives σ_quorum_j produced by Ch using the honest 5-of-9 quorum. Forgery: A outputs (I*, S*, C_hash*, ts*, src*, dst*, addr*, nonce*, σ*). Win condition: A wins if all of the following hold: (a) σ* verifies under pk on canonical((I*, S*, C_hash*, ts*, src*, dst*, addr*, nonce*)); (b) (I*, S*, C_hash*, ts*, src*, dst*, addr*, nonce*) was not submitted as a signing query; (c) the destination registry on chain dst* has no record of nonce* under namespace I*.

10.4 Theorem

Theorem 10.1 (Certificate Unforgeability). *If the underlying threshold signature scheme Σ_b^{(t_b, n_b)} = (5,9) is EUF-CMA secure under static corruption with up to 4 corrupted parties, then for any PPT adversary A against CertEUF, the advantage Adv^CertEUF(A) is bounded by Adv^EUF-CMA(B, Σ_b) for some PPT B with running time approximately equal to A's. Concretely, no PPT adversary wins CertEUF with non-negligible probability in λ.*

Proof sketch. By reduction to the EUF-CMA security of Σ_b. Given a PPT adversary A against CertEUF, we construct B against Σ_b. B receives pk from its EUF-CMA challenger and forwards pk to A. B answers each CertEUF signing query X_j by querying its own Σ_b-oracle on canonical(X_j) and forwarding the resulting signature to A. When A outputs a forgery (X*, σ*), B outputs (canonical(X*), σ*) as its EUF-CMA forgery. By condition (b) of CertEUF, X* was not a signing query, so canonical(X*) was not a Σ_b query (canonical is injective on the certificate field tuple); by condition (a), σ* verifies under pk; therefore (canonical(X*), σ*) is a valid EUF-CMA forgery. Condition (c) is enforced by the destination registry, not by B, but it ensures that even a legitimate certificate cannot be replayed once consumed. Combining unforgeability with replay protection gives the full security guarantee. The bound Adv^CertEUF(A) ≤ Adv^EUF-CMA(B, Σ_b) follows. □

10.5 Tamarin Model (Roadmap)

We are preparing a Tamarin Prover model of the cross-chain protocol, including the bridge committee, the source mirror contract, the destination mirror contract, the registry, and a Dolev-Yao network adversary. The model encodes the seven invariants of §10.2 as Tamarin lemmas and is being prepared as a public artifact (github.com/t3rra/tamarin) for replication and adversarial review by the academic community. Until the mechanization is complete and reviewed, Theorem 10.1 should be read as a paper proof.

10.6 Bridge Committee Composition and Rotation

The 9 bridge committee members are independent validator operators selected by DAO governance. Selection criteria include geographic distribution, regulatory jurisdiction, hardware attestation capability, and historical operational uptime. Rotation is on a 12-month schedule with the option for emergency rotation by the GovernanceOverride. Rotation is itself a key-resharing ceremony — the public key of the bridge committee is preserved across rotation, so destination mirror contracts do not need reconfiguration. Rotation produces a public LegalMirror entry under L3RS-1 §13.

11. Hardware Attestation

11.1 Attestation as a Cryptographic Object

Each MPC party in T3RRA's wallet runs in an attested hardware environment. The attestation is itself a signed statement from a hardware manufacturer's root certificate authority asserting that the running code is the expected code on the expected hardware. T3RRA treats attestation as a first-class cryptographic object: every signing ceremony begins with each party publishing an attestation chain, and the ceremony aborts if any chain fails to validate.

11.2 Per-Environment Attestation Roots

Environment	Attestation Root	Mechanism
iOS Secure Enclave	Apple Anonymous Attestation CA	DeviceCheck App Attest API
Android StrongBox	Google Attestation Root	Hardware Key Attestation
AWS Nitro Enclave	AWS Nitro Attestation Root	Nitro Attestation Document, signed measured boot
FIPS 140-3 L3 HSM (backend)	Vendor manufacturer root (Thales, Entrust, AWS CloudHSM)	PKCS#11 attestation
Independent recovery custodian HSM	Vendor manufacturer root (independent)	Vendor-specific attestation API

11.3 Attestation Chain Validation

Validation proceeds in three steps. Step 1: each attestation is verified against its root certificate authority's public key, which is hard-coded into T3RRA's protocol artifact and cannot be silently changed. Step 2: the attestation's nonce is checked to ensure it matches a fresh challenge generated for the current ceremony, defeating replay. Step 3: the attestation's measurement (the expected hash of the running code) is checked against the L3RS-1 LegalMirror entry for the current T3RRA wallet release. Any failure aborts the ceremony with reason AttestationFailure and emits an AbortReport.

12. RNG, Constant-Time, and Side-Channel Posture

12.1 Random Number Generation

Every party uses a CSPRNG seeded by hardware-rooted entropy. On mobile devices, entropy is drawn from the Secure Enclave / StrongBox hardware RNG. On backend nodes, entropy is drawn from the FIPS 140-3 Level 3 HSM's certified entropy source. The CSPRNG is HMAC-DRBG (NIST SP 800-90A) with SHA-256, instantiated per process and reseeded on every signing ceremony. We avoid the historical pitfalls of biased nonces in ECDSA by using the deterministic nonce generation of RFC 6979 augmented with fresh randomness — a hybrid that protects against both bad RNGs and against some classes of fault attacks.

12.2 Constant-Time Implementation

All cryptographic operations on secret material are implemented in constant time with respect to the secret. We use audited curve libraries (libsecp256k1 for secp256k1, dalek-ed25519 for edwards25519) that are themselves constant-time. T3RRA's Paillier implementation for CMP-NI is constant-time with respect to ciphertexts and secrets; deviations from constant time are flagged as audit findings.

12.3 Software Bill of Materials

T3RRA publishes a Software Bill of Materials (SBOM) for every release, signed with Sigstore, listing every dependency, its version, and its cryptographic checksum. The SBOM is itself a LegalMirror artifact under L3RS-1 §13 and is reproducible from the source repository.

13. Travel Rule as a Cryptographic Precondition

13.1 IVMS 101 Canonical Form

The InterVASP Messaging Standard 101 (IVMS 101) is the canonical data model for Travel Rule payloads. T3RRA constructs the IVMS 101 payload deterministically from the transfer context, with the canonical encoding specified by the IVMS 101 standard. The canonical form is required for verifiable receipts: a VASP receipt is signed over the canonical encoding, and any party can re-derive the encoding from the context.

13.2 Protocol Coverage

Protocol	Mechanism	Adapter Status
TRP	REST + JWS receipts	Production
TRISA	gRPC + mTLS + signed payloads	Production
Sygna Bridge	REST + sender/receiver private routing	Production
OpenVASP	P2P with whisper-style routing	Beta

13.3 The Pre-Sign Check

Before any keyshare holder participates in round 1 of a signing ceremony for an asset whose ID tier is L2 or higher and whose size σ exceeds dm(I), every party independently performs the following three checks: (a) the counterparty VASP identifier in the context is discoverable in at least one configured VASP directory and resolves to a recognized public key; (b) the VASP returns a valid Travel Rule receipt over the IVMS 101 canonical encoding constructed from the transfer context, signed under the discovered public key; (c) the receipt's signing key chains to a recognized VASP attestation root. If any check fails, the party emits AbortReport(sid, P_i, TravelRuleAbort, condition) and refuses round 1, halting the signing ceremony.

14. Concrete Parameters

Parameter	Value
Curve (ECDSA)	secp256k1 (SECG)
Curve (EdDSA)	edwards25519 (RFC 8032)
Threshold (user wallet)	(2, 3)
Threshold (bridge committee)	(5, 9)
Hash (commitment)	SHA-256, dom string “T3RRA-COM-v1”
Hash (Fiat–Shamir)	SHA-3-256, dom string “T3RRA-FS-v1”
Hash (PolicyHash)	SHA-3-256, dom string “T3RRA-POLICY-HASH-v1”
Hash (cross-chain certificate)	SHA-256, dom string “L3RS-1-CERT-v1”
Hash (Asset_ID)	SHA-256, dom string “L3RS-1-AID-v1”
KEM (transport)	ML-KEM-768 (NIST FIPS 203)
KDF	HKDF-SHA-256 (RFC 5869)
AEAD (transport)	AES-256-GCM and ChaCha20-Poly1305
Nonce generation (ECDSA)	RFC 6979 + hybrid randomness
Proactive resharing interval (user)	90 days
Proactive resharing interval (bridge)	30 days
Identifiable abort	Required for all signing protocols
ZK soundness error	≤ 2^{-128} per proof
Random oracle query bound	q_H ≤ 2^{60}
RNG	HMAC-DRBG with SHA-256, hardware-seeded
FIPS posture (backend)	FIPS 140-3 Level 3 HSMs in AWS Nitro

15. Mechanization Roadmap and Honest Caveats

This section names what is not yet proven, not yet measured, and not yet mechanized. The crypto community can spot a glossed-over caveat at thirty paces; we would rather state every gap explicitly than have a reviewer find one we omitted.

15.1 Paper Proofs vs Mechanized Proofs

All theorems in this document — Theorems 5.1, 7.1, 7.2, 7.3, 8.1, 8.2, 9.1, and 10.1 — are paper proofs. We have not yet mechanized any of them in EasyCrypt, Tamarin, Coq, or any other proof assistant. Mechanization is in progress; the target is the following:

Theorem	Mechanization Target	Status	ETA
Thm 7.1 (Policy Soundness)	EasyCrypt	In progress	Q4 2026
Thm 7.2 (Unforgeability Preservation)	EasyCrypt	In progress	Q4 2026
Thm 7.3 (Policy Replay)	EasyCrypt (corollary of 7.2)	Not started	Q1 2027
Thm 8.1, 8.2 (Strategy-Proofness)	Mechanized model in Lean 4	Not started	Q2 2027
Thm 9.1 (Tractability)	Standard CS, no mechanization needed	Complete (paper)	—
Thm 10.1 (Certificate Unforgeability)	Tamarin Prover (full protocol model)	In progress	Q3 2026

15.2 Other Caveats

- The compliance fitness venue scoring component of §9 has not been validated against a long-horizon dataset of regulated venue activity, because no such dataset is yet public.
- The Lean 4 mechanization of Theorems 8.1 and 8.2 will require a model of the matching engine as a labelled transition system; this model is not yet built.
- Threshold post-quantum signatures are not production-ready in any scheme we are willing to deploy. See §19.
- The L3RS-1 conformance matrix at Profile F (Appendix C of the v3.1 whitepaper) has been internally verified but has not yet been audited by an independent L3RS-1 conformance auditor; that audit is scheduled for Q2 2026.
- Identifiable abort in CMP-NI is theoretically present, but T3RRA's implementation has not yet been independently verified to identify every possible deviation pattern; this is an audit item.
- The DKLs23 implementation is currently a research prototype and has not yet been audited for production deployment.

16. Benchmarks Methodology and Targets

16.1 Methodology

All benchmarks are governed by a single methodology: (a) hardware specified per row; (b) git commit hash of the T3RRA wallet specified at the time of measurement; (c) reproducibility harness with build instructions and synthetic transcripts published at github.com/t3rra/benchmarks; (d) third-party reproductions welcomed and posted to the same repository upon submission. Numbers labeled "Target" are aspirational; numbers labeled "Measured" come from a specific commit hash and a specific hardware configuration. We prefer to under-promise and reproduce than to publish marketing numbers.

16.2 Signing Latency Targets

Protocol	Hardware	Status	p50	p99
CMP-NI (mobile, warm cache)	iPhone 15 Pro + 2× AWS Nitro c6i.xlarge	Target	180 ms	350 ms
CMP-NI (back-end-only)	3× AWS Nitro c6i.xlarge	Target	45 ms	90 ms
DKLs23 (online phase)	iPhone 15 Pro + 2× AWS Nitro c6i.xlarge	Target	90 ms	200 ms
FROST (mobile, warm cache)	iPhone 15 Pro + 2× AWS Nitro c6i.xlarge	Target	70 ms	160 ms
DKG (one-time, full)	iPhone 15 Pro + 2× AWS Nitro c6i.xlarge	Target	1.2 s	2.5 s
Proactive Refresh (all-backend)	3× AWS Nitro c6i.xlarge	Target	0.8 s	1.6 s
Cross-chain Certificate (5-of-9)	9× AWS Nitro c6i.xlarge geo-distributed	Target	320 ms	750 ms

16.3 Throughput Targets

Subsystem	Target	Bottleneck
Wallet signing (single key)	200 sigs/sec	MPC round trips
Marketplace matching	50,000 orders/sec	Compliance evaluation throughput
Flow route discovery	1,000 quotes/sec per asset pair	Dijkstra over 500-vertex graph
Cross-chain certificate generation	300 certs/sec	Quorum signature aggregation

17. Reproducibility, Audits, and Bounty

17.1 Reproducible Builds

Every T3RRA wallet release is built reproducibly: identical source produces identical binaries. Build attestations are signed with Sigstore and published alongside the release. The reproducibility harness allows any third party to rebuild from source and verify the binary checksum.

17.2 Audit Schedule

Auditor	Scope	Status
NCC Group	CMP-NI implementation, Paillier ZK proofs, key handling	Engaged Q2 2026
Trail of Bits	DKLs23 implementation and OT extension	Engaged Q3 2026
Kudelski Security	FROST implementation, ed25519 stack	Engaged Q3 2026
Quarkslab	Mobile share environment, Secure Enclave / StrongBox integration	Engaged Q4 2026
Cure53	Front-end, RPC layer, browser extension	Engaged Q4 2026
ChainSecurity	Smart contract layer (issuance, marketplace, bridge)	Engaged Q2 2026
Spearbit	Smart contract layer, route discovery	Engaged Q3 2026
OpenZeppelin	Smart contract layer, governance override	Engaged Q4 2026

17.3 Bug Bounty

T3RRA operates an Immunefi bug bounty with the following payout structure: $2,500,000 maximum for protocol-level breaks of any of the seven L3RS-1 theorems or the four T3RRA theorems (Thm 7.1, 7.2, 7.3, 8.1, 8.2, 9.1, 10.1); $1,000,000 for theft-of-funds vulnerabilities; $500,000 for permanent freeze of funds; $250,000 for protocol-insolvency vulnerabilities; lower tiers for less critical findings. All audit reports and bounty payouts are published in full.

18. Comparative Analysis

18.1 Protocol Provenance Table

Vendor	Threshold ECDSA	Threshold Schnorr	Year of Public Spec
Fireblocks	CMP variant	—	2020
Blockdaemon Builder Vault	DKLs (Sepior origins)	—	2019
Coinbase WaaS	Internal (undisclosed)	—	Unknown
Safeheron	GG20 (open source)	—	2020
ZenGo	Lindell17 (2-of-2)	—	2017
T3RRA	PG[CMP-NI], PG[DKLs23]	PG[FROST]	2026

18.2 Capability Matrix

Capability	T3RRA	Fireblocks	Blockdaemon	Coinbase WaaS	Safeheron	ZenGo
UC-secure threshold ECDSA	Yes (CMP-NI/ DKLs23)	Yes (CMP)	Yes (DKLs)	Undisclosed	Yes (GG20)	Yes (L17, 2/2)
Identifiable abort	Yes	Yes	Yes	Undisclosed	Yes	N/A
Proactive resharing	Yes (90d/30d)	Yes	Yes	Undisclosed	Yes	N/A
Hardware attestation chain	Yes (multi-root)	Yes (HSM only)	Yes (HSM)	Undisclosed	Partial	Partial
FROST / threshold Schnorr	Yes	Partial	No	Undisclosed	No	No
Policy-gated signing (cryptographic)	Yes (Thm 7.1)	Policy engine (UI)	Policy engine (UI)	Policy engine (UI)	No	No
Compliance-gated matching	Yes (Thm 8.1)	N/A	N/A	N/A	N/A	N/A
Route admissibility predicate	Yes (Thm 9.1)	No	No	No	No	No
Cross-chain certificate (formal)	Yes (Thm 10.1)	No	No	No	No	No
L3RS-1 conformance	Profile F	No	No	No	No	No
Public bibliography in spec	Yes (22 refs)	Marketing only	Marketing only	Marketing only	Yes (paper)	Yes (paper)
Mechanized proofs (target)	EasyCrypt + Tamarin	No public	No public	No public	No public	No public

18.3 Where T3RRA Leads

Three independent moats. (1) Standards alignment: no competitor binds their architecture to a published normative meta-standard with a conformance matrix and theorem map. (2) Cryptographic policy enforcement: no competitor enforces compliance inside the Fiat–Shamir challenge of the signing protocol; competitors enforce at the policy engine, which is upgradeable software an attacker can hope

to bypass. (3) End-to-end formalization: no competitor publishes formal definitions, security games, and proof sketches for compliance-gated matching, route admissibility, and cross-chain certificate unforgeability.

18.4 Where T3RRA Must Catch Up

(1) Battle-tested codebase: Fireblocks has a multi-year head start on adversarial environment exposure. (2) Number of integrated chains: T3RRA launches with EVM, Bitcoin, and Solana, behind Fireblocks's chain coverage. (3) Mechanized proofs: Safeheron and ZenGo have published mechanized work for some of their constructions; T3RRA's mechanizations are in progress and not yet published. We treat each of these as a roadmap item, not a structural disadvantage.

19. Post-Quantum Roadmap

Threshold post-quantum signatures are not production-ready in any scheme we are willing to deploy. ML-DSA (Dilithium) does not have a published threshold construction with acceptable round complexity. Raccoon (del Pino et al.) is a strong candidate but has not yet been peer-reviewed at the level required for a $1B+ regulated-asset deployment. Threshold variants of Falcon and the Espitau–Niot–Prest constructions are at earlier stages still. We will not deploy threshold PQ signing until a construction has cleared a recognized peer-review process — IACR ePrint with twelve months of public scrutiny, plus an independent security audit — regardless of marketing pressure.

19.1 Staged Roadmap

Phase	Window	Posture
Phase 1 (current)	2026	Hybrid: classical threshold signing + ML-KEM-768 for transport
Phase 2	2027	Hybrid signatures: classical threshold + non-threshold PQ co-signature on critical operations (issuance, governance override, cross-chain certificate)
Phase 3	2028+	Threshold PQ pilot for low-value assets once a peer-reviewed protocol exists with two independent academic implementations
Phase 4	2029+	Production threshold PQ for all asset types as protocols mature

19.2 Quarterly Posture Reports

T3RRA commits to publishing a quarterly PQ posture report that updates the staged roadmap against the current state of the art. The report names every candidate construction we are tracking, the peer-review status, and our internal evaluation against our deployment criteria.

20. Open Problems

We list problems we have not solved and which we believe are interesting research directions. We will fund grants for credible work on any of them; please reach out via research@t3rra.io.

- OP1 — Threshold ML-DSA with sub-second round complexity. The leading academic candidates have not yet achieved the latency required for production wallet operation. A peer-reviewed construction here unblocks Phase 3 of the PQ roadmap (§19).
- OP2 — Mechanized verification of policy-gated signing in EasyCrypt. We have a paper proof of Theorems 7.1, 7.2, and 7.3; a mechanized proof would meaningfully strengthen the result and allow inclusion in formal-methods venues.
- OP3 — Formal microstructure analysis of compliance-gated matching under adversarial counterparties. Theorems 8.1 and 8.2 assume honest counterparties; the strategy-proofness of the construction in the presence of dishonest counterparties (e.g., colluding to manipulate the compliance evaluation timing) is open.
- OP4 — Optimal scoring of venue compliance fitness over long horizons. We currently use a 24-hour rolling window; the right window depends on the asset class and the regulatory environment, and a principled choice is open.
- OP5 — Verifiable inclusion proofs for LegalMirror artifacts that work across both IPFS and AWS GovCloud anchors with a single Merkle commitment. A compact construction here simplifies regulator integration.
- OP6 — Incentive-compatible bridge committee rotation that resists collusion. The current 5-of-9 quorum committee rotates by governance vote; a fully automatic rotation that resists collusion is open.
- OP7 — Adversarial robustness of the contextual bandit in §9.6 against reward poisoning by a malicious venue. A bandit that converges to a near-optimal policy under bounded adversarial reward perturbation would meaningfully improve T3RRA's liquidity routing.
- OP8 — Tighter security loss in Theorem 7.2. Our reduction adds q_H · 2^{-λ} + q_S · 2^{-λ}; a tight reduction or matching attack would clarify the exact security level achievable by PG[Σ].
- OP9 — A one-message threshold ECDSA protocol with identifiable abort. Existing protocols require at least two online rounds; a single-message protocol would dramatically improve mobile-share user experience.

21. Bibliography

[1] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. FOCS 2001.

[2] R. Canetti, N. Makriyannis, U. Peled. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. ACM CCS 2020.

[3] J. Doerner, Y. Kondi, E. Lee, a. shelat. Secure Two-party Threshold ECDSA from ECDSA Assumptions. IEEE S&P 2018.

[4] J. Doerner, Y. Kondi, E. Lee, a. shelat. Threshold ECDSA from ECDSA Assumptions: The Multiparty Case. IEEE S&P 2019.

[5] J. Doerner, Y. Kondi, E. Lee, a. shelat. Threshold ECDSA in Three Rounds. IEEE S&P 2024.

[6] R. Gennaro, S. Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. ACM CCS 2018.

[7] R. Gennaro, S. Goldfeder. One Round Threshold ECDSA with Identifiable Abort. ePrint 2020/540.

[8] Y. Lindell. Fast Secure Two-Party ECDSA Signing. CRYPTO 2017.

[9] C. Komlo, I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. SAC 2020.

[10] R. Ostrovsky, M. Yung. How to Withstand Mobile Virus Attacks. PODC 1991.

[11] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung. Proactive Secret Sharing or: How to Cope with Perpetual Leakage. CRYPTO 1995.

[12] S. K. D. Maram et al. CHURP: Dynamic-Committee Proactive Secret Sharing. ACM CCS 2019.

[13] R. del Pino et al. Raccoon: A Side-Channel Resistant Lattice-Based Signature Scheme. NIST PQC 2024.

[14] T. Espitau, T. Niot, T. Prest. Threshold Signatures from Lattices. CRYPTO 2024.

[15] NIST FIPS 203. Module-Lattice-Based Key-Encapsulation Mechanism Standard. 2024.

[16] NIST FIPS 204. Module-Lattice-Based Digital Signature Standard. 2024.

[17] NIST FIPS 205. Stateless Hash-Based Digital Signature Standard. 2024.

[18] FATF. International Standards on Combating Money Laundering, Recommendation 16. 2012, updated 2023.

[19] InterVASP. IVMS 101: InterVASP Messaging Standard. 2020, updated 2024.

[20] D. Pointcheval, J. Stern. Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology, 2000.

[21] D. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang. High-speed high-security signatures (Ed25519). CHES 2011.

[22] T. Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979, 2013.

[23] H. Krawczyk, P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, 2010.

[24] L3RS-1 v1.0.0. Layer-3 Regulated Asset Standard. February 2026.

[25] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, B. Schmidt. Tamarin Prover for the Symbolic Analysis of Security Protocols. CAV 2013.

[26] G. Barthe, B. Grégoire, S. Heraud, S. Zanella-Béguelin. Computer-Aided Security Proofs for the Working Cryptographer. CRYPTO 2011 (EasyCrypt).

22. Disclaimer

This specification is for technical and informational purposes only and does not constitute an offer to sell or a solicitation to buy any security, token, or financial instrument in any jurisdiction. Statements about future product capabilities, performance targets, audit outcomes, and mechanization timelines are forward-looking and subject to change. The T3RRA platform is operated subject to the laws of each jurisdiction in which it is offered.

Theorems 5.1, 7.1, 7.2, 7.3, 8.1, 8.2, 9.1, and 10.1 are paper proofs. Mechanized proofs in EasyCrypt, Tamarin, and Lean 4 are in progress per §15 and will be released as public artifacts when complete. Benchmarks marked Target in §16 are aspirational; measured benchmarks will be published quarterly via the public reproducibility harness at github.com/t3rra/benchmarks beginning Q3 2026. Audit reports per §17 will be published in full on completion.

This specification supersedes rev A. Rev A described the cryptographic toolkit; rev B treats the four T3RRA constructions as cryptographic objects with definitions, games, theorems, and proofs. Future revisions will incorporate (a) measured benchmarks replacing targets, (b) mechanized proofs replacing paper proofs, (c) audit findings, and (d) any L3RS-1 amendments adopted by the working group.