Security Proof Obligations, Red-Team Methodology, and Verification Strategy

Zach Kelling

Lux Industries

2025

Abstract

Security claims without verification are marketing. This paper specifies Lux Network's proof obligations: what must be mechanically proved (Lean4), what must be symbolically verified (Halmos, Tamarin), what must be fuzzed (Forge, go-fuzz), what must be model-checked (TLA+), and what must be adversarially reviewed before any security claim is made in a Lux research paper. We define five verification tiers, assign every protocol component to a tier, and describe the toolchain, coverage requirements, and sign-off process. The methodology applies to consensus protocols (Quasar), cryptographic primitives (Pulsar, FHE), bridge protocols (Teleport), and smart contract systems. Artifacts from this process are maintained in `luxfi/formal` and `luxfi/proofs`.

1 Introduction

Lux Network makes specific security claims: Quasar provides safety under $f < n/3$ Byzantine validators [1], Corona provides 128-bit post-quantum security [2], FHE provides IND-CPA security under LWE hardness [3]. Each of these claims carries a proof obligation—a verification artifact that demonstrates the claim holds, not just a prose argument in a paper.

This document defines what constitutes adequate verification for each category of claim, the tools used to produce verification artifacts, and the process by which claims are promoted from "stated" to "verified."

2 Verification Tiers

Definition 1 (Verification Tier)**.** *A verification tier τ specifies the minimum verification rigor required before a claim of type τ may be published.*

Tier	Claim Type	Verification	Tool
T1	Mathematical proof	Machine-checked proof	Lean4
T2	Protocol security	Symbolic verification	Tamarin, ProVerif
T3	Smart contract correctness	Formal verification + fuzzing	Halmos, Forge
T4	Implementation correctness	Property-based testing + fuzzing	go-fuzz, Hypothesis
T5	Operational security	Red-team exercise + audit	Manual + automated

Table 1: Verification tiers. Higher tiers (T1) provide stronger guarantees.

2.1 Tier Assignment

Every protocol component is assigned a tier based on its security criticality:

3 Tier 1: Machine-Checked Proofs (Lean4)

3.1 Scope

Tier 1 proofs are mechanically verified mathematical statements. They prove properties of abstract protocols, not of specific implementations. The gap between the abstract model and the implementation is covered by lower tiers.

Component	Tier	Rationale
Quasar safety/liveness	T1	Consensus failure = chain death
BLS aggregation correctness	T1	Signature forgery = fund theft
Pulsar threshold scheme	T1+T2	Novel construction, PQ claims
FHE bootstrapping correctness	T1	Decryption failure = data loss
FROST-BLS protocol	T2	Multi-party, network adversary
Teleport bridge messages	T2+T3	Cross-chain, fund custody
EVM precompiles (FHE)	T3	Gas metering, state correctness
FHE Go implementation	T4	Performance-critical code
NTT SIMD engine	T4	Numerical correctness
Validator deployment	T5	Operational procedures
Key rotation procedures	T5	Human-in-the-loop

Table 2: Component-to-tier assignment.

3.2 Current Proof Obligations

1. **Quasar Safety**: No two honest validators finalize conflicting blocks under $f < n/3$ and bounded network delay.
2. **Quasar Liveness**: All valid transactions eventually finalize under the same assumptions.
3. **BLS Aggregate Unforgeability**: An adversary who does not control $\geq t$ signing keys cannot forge a valid aggregate signature.
4. **FHE Correctness**: Decryption of a bootstrapped ciphertext returns the correct plaintext with probability $\geq 1 - 2^{-\lambda}$ for security parameter λ.

Proofs are maintained in `luxfi/formal/lean4/` and are checked by CI on every commit.

3.3 Lean4 Proof Structure

```
formal/lean4/
  Quasar/
    Safety.lean          -- No conflicting finalization
    Liveness.lean        -- Eventual finalization
    ByzantineModel.lean  -- f < n/3 adversary model
  BLS/
    Aggregation.lean     -- Aggregate unforgeability
    ProofOfPossession.lean
  FHE/
    Bootstrapping.lean   -- Correctness after blind rotation
    NoiseGrowth.lean     -- Noise budget sufficient
```

4 Tier 2: Symbolic Protocol Verification (Tamarin)

4.1 Scope

Tier 2 verification models multi-party protocols under a Dolev-Yao adversary (active network attacker). Tamarin Prover is used for protocols involving key exchange, signing, and cross-chain messaging.

4.2 Current Models

1. **FROST-BLS Key Generation**: Verify that the DKG protocol produces consistent shares even if $< t$ parties are malicious.
2. **Teleport Bridge**: Verify that a cross-chain transfer completes if and only if $\geq t$ MPC nodes attest, and that no double-spend is possible.
3. **Pulsar Threshold Signing**: Verify unforgeability under concurrent sessions with a network adversary.

Models are maintained in `luxfi/proofs/tamarin/` and checked by CI with a 30-minute timeout per lemma.

4.3 Tamarin Model Example

```
// Teleport bridge: no double-spend
lemma no_double_spend:
  "All src dst amt #i #j.
    Transfer(src, dst, amt) @ #i &
    Transfer(src, dst, amt) @ #j
    ==> #i = #j"
```

5 Tier 3: Smart Contract Verification (Halmos + Forge)

5.1 Scope

Tier 3 covers Solidity smart contracts and EVM precompiles. Halmos provides symbolic execution (proving properties hold for all possible inputs); Forge provides concrete fuzzing.

5.2 Verification Targets

1. **FHE Precompiles**: Gas metering is correct (no underpriced operations that enable DoS). State transitions preserve ciphertext validity.
2. **Teleport Contracts**: Token mint/burn balance invariant. Signature verification matches Tamarin model.
3. **Governance**: Vote weight calculation, proposal execution conditions.

5.3 Coverage Requirements

Verification Method	Target	Minimum
Halmos symbolic execution	All public functions	100%
Forge fuzzing	All state-changing functions	10K runs
Forge invariant testing	All storage invariants	10K runs
Slither static analysis	All contracts	Zero high findings

Table 3: Smart contract verification coverage requirements.

6 Tier 4: Implementation Testing (go-fuzz, Hypothesis)

6.1 Scope

Tier 4 covers the Go and Rust implementations of cryptographic primitives. Property-based testing and fuzzing catch implementation bugs that formal proofs of the abstract protocol do not cover.

6.2 Properties Tested

1. **NTT Roundtrip**: For all polynomials p, $\text{INTT}(\text{NTT}(p)) = p$.
2. **Encrypt-Decrypt Roundtrip**: For all parameters, keys, and messages, $\text{Dec}(\text{Enc}(m)) = m$.
3. **Homomorphic Correctness**: For all a, b, $\text{Dec}(\text{Add}(\text{Enc}(a), \text{Enc}(b))) = a + b$.
4. **Serialization Roundtrip**: For all keys and ciphertexts, $\text{Unmarshal}(\text{Marshal}(x)) = x$.
5. **Error Propagation**: Invalid inputs produce explicit errors, never panics.

Fuzzing targets are maintained in `luxfi/fhe/fuzz/` and `luxfi/lattice/fuzz/`. CI runs fuzz campaigns for 10 minutes per target per commit.

7 Tier 5: Red-Team and Adversarial Review

7.1 Scope

Tier 5 covers operational security: deployment procedures, key management workflows, incident response, and human-in-the-loop processes that cannot be fully automated.

7.2 Red-Team Protocol

1. **Scope**: The red team receives a deployment-identical testnet environment with real validator keys (testnet-only).
2. **Objective**: Compromise a validator's signing key, forge a block, double-spend across chains, or halt consensus.
3. **Duration**: 2 weeks per engagement.
4. **Reporting**: Findings are classified as Critical/High/Medium/Low. Critical and High findings block mainnet deployment.
5. **Remediation**: Each finding is addressed with a code change and a regression test (Tier 3 or 4).

7.3 External Audit Requirements

8 TLA+ Model Checking

For distributed protocol properties that are expensive to prove in Lean4 but amenable to bounded model checking, TLA+ specifications are used:

1. **Quasar Leader Election**: Verify absence of livelock in the leader rotation mechanism.

Component	Audit Requirement
Consensus (Quasar)	2 independent audits
Cryptography (Pulsar, FHE)	1 audit + 1 academic review
Bridge (Teleport)	2 independent audits
Smart contracts	1 audit per major release

Table 4: External audit requirements by component.

2. **Threshold DKG**: Verify that key shares are consistent even under $< t$ Byzantine parties within bounded rounds.

3. **Key Rotation**: Verify that the cryptographic agility migration [4] maintains the invariant that at least one valid key schedule is active at all times.

TLA+ specifications are maintained in `luxfi/formal/tla/` and model-checked by TLC with bounded state spaces.

9 Conclusion

Every security claim in a Lux research paper carries a verification obligation. The obligation is discharged by the appropriate tool at the appropriate tier: Lean4 for mathematical proofs, Tamarin for protocol security, Halmos and Forge for smart contracts, go-fuzz for implementations, and red-team exercises for operational security. A claim without a corresponding verification artifact is a hypothesis, not a result. The artifacts are maintained in `luxfi/formal` and `luxfi/proofs`, checked by CI, and referenced from the papers that make the claims.

References

[1] Kelling, Z. (2025). *Quasar: Quantum-Secure Multi-Engine Consensus with Triple-Proof Quantum Finality*. Lux Industries Research.

[2] Lux Industries. *Corona: Two-Round Module-LWE Threshold Signatures*. Lux Industries, 2026. `github.com/luxfi/corona`. Module-LWE (Ringtail-derived), two-round.

[3] Kelling, Z. (2025). *Threshold Fully Homomorphic Encryption for Confidential DeFi*. Lux Industries Research.

[4] Kelling, Z. (2024). *Cryptographic Agility Architecture for Long-Lived Blockchain Systems*. Lux Industries Research.

[5] Meier, S. et al. (2013). *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*. 25th International Conference on Computer Aided Verification (CAV).

[6] de Moura, L. & Ullrich, S. (2021). *The Lean 4 Theorem Prover and Programming Language*. CADE-28.

[7] Halmos Contributors (2024). *Halmos: Symbolic Bounded Model Checker for EVM*. a]16z Research.

[8] Lamport, L. (2002). *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.