Restaking Protocol for Lux Network Security Extension

Enabling Capital-Efficient Economic Security Sharing Across Appchains and Actively Validated Services

Zach Kelling

Lux Industries

2024

Abstract

We present the **Lux Restaking Protocol** (LRP), a mechanism enabling LUX token holders to extend the economic security of their staked capital to multiple Actively Validated Services (AVSs) and application-specific appchains simultaneously. Unlike traditional staking, where each validator commitment is siloed within a single chain, LRP allows a single unit of staked LUX to simultaneously secure an arbitrary set of registered services subject to independent slashing conditions. We formalize the security model as a hypergraph of staking commitments, prove that aggregate economic security is superadditive under realistic parameter regimes, and derive tight bounds on the systemic risk introduced by correlated slashing events. Our capital-efficiency analysis demonstrates that LRP achieves a 4.7× improvement in security-per-token-staked compared to isolated appchain validation, with measurable improvements in bootstrapping economics for new appchains. We compare LRP to EigenLayer's design on Ethereum, identifying five structural advantages arising from Lux's native Beacon consensus and its existing Primary Network validator architecture. A prototype deployment on Lux Fuji testnet processes operator registrations in under 200 ms and executes slashing with sub-2-second finality, demonstrating practical viability at scale.

Contents

1 Introduction

Proof-of-stake networks achieve Byzantine fault tolerance by requiring validators to post economic collateral that may be confiscated—*slashed*—if they behave dishonestly. This arrangement transforms cryptographic security into economic security: the cost of an attack is bounded below by the value of stake at risk. However, as the number of services requiring such security grows, naive staking architectures suffer from a fundamental tension.

On one side, validators face a *capital allocation problem*: each new service they wish to secure demands a fresh deposit, fragmenting their capital and reducing the yield they earn on any given unit. On the other side, new services face a *bootstrapping problem*: accumulating enough staked capital to credibly threaten slashers requires either recruiting an entirely new validator set or offering yields so high they are economically unsustainable.

Restaking resolves this tension by permitting a single deposit to simultaneously collateralize multiple services. The key insight is that staked tokens can serve as economic security *jointly* across services so long as the aggregate slashing exposure is bounded and the rules governing each service are transparent and enforceable on-chain.

This paper describes the design, analysis, and early implementation of the Lux Restaking Protocol. Our contributions are:

1. A formal model of restaking as a hypergraph over staking commitments, with rigorous definitions of operator exposure, aggregate economic security, and systemic risk (Section 3).
2. A slashing architecture that coordinates across independent AVS slashers while preventing double-slash attacks and ensuring stake coverage invariants (Section 4).
3. An operator registration protocol with cryptographic attestation, on-chain delegation, and progressive opt-in/opt-out semantics (Section 5).
4. A marketplace design for AVS discovery and negotiation, including reputation scoring and minimum-security thresholds (Section 6).
5. Formal capital efficiency and systemic risk analyses, with comparison to EigenLayer (Sections 7–9).
6. Prototype implementation benchmarks on Lux Fuji testnet (Section 10).

1.1 Motivation

The Lux Network operates a Primary Network of validators who secure the Exchange Chain (X), Platform Chain (P), and Contract Chain (C). Appchains—application-specific blockchains—may elect their own validator sets from any subset of Primary Network validators. This architecture creates a natural restaking substrate: Primary Network validators already post their LUX stake once; extending that commitment to additional appchains and services is an incremental rather than additive cost.

As of 2025, Lux hosts 47 production appchains with a combined transaction throughput exceeding 450,000 TPS. Many emerging appchains, particularly those launched by DeFi protocols, gaming studios, and institutional asset managers, require economic security guarantees before they can attract liquidity and users. LRP directly addresses this bootstrapping bottleneck.

1.2 Paper Organization

Section 2 reviews related work and the Lux Network architecture. Section 3 develops the formal staking hypergraph model. Section 4 describes slashing conditions and the veto committee. Section 5 details operator registration and delegation. Section 6 presents the AVS marketplace design. Section 7 analyzes capital efficiency. Section 8 analyzes systemic risk. Section 9 compares LRP with EigenLayer, Babylon, and Symbiotic. Section 10 describes the prototype implementation. Section 11 provides the formal security analysis. Section 13 concludes.

2 Background

2.1 Lux Network Architecture

Lux employs a three-chain primary network:

- **P-Chain** (Platform Chain): tracks validator sets, manages staking, creates appchains.
- **X-Chain** (Exchange Chain): UTXO-based chain for asset issuance and trading.
- **C-Chain** (Contract Chain): EVM-compatible chain for smart contracts.

Validators post a minimum 2,000 LUX stake on the P-Chain and run all three primary chains. Appchain validation is opt-in: a primary validator may additionally validate any subset of registered appchains by running the appchain's VM and advertising participation on the P-Chain. Historically, appchain validators received no direct remuneration from appchains, creating a mismatch between the security provided and the incentive to provide it [1].

2.2 Beacon Consensus

Lux's Beacon consensus achieves probabilistic finality in $\mathcal{O}(\log n)$ rounds under the honest-majority assumption, with sub-second finality observed in practice. The key property relevant to restaking is that conflicting transactions are identified and rejected rapidly: any equivocating validator (one who votes for two conflicting blocks) can be detected within one consensus epoch. This property enables efficient slashing without requiring extended unbonding periods [2].

2.3 EigenLayer and Prior Work

EigenLayer [3] pioneered restaking on Ethereum, enabling ETH stakers to opt into securing additional Actively Validated Services via delegation to operators who run AVS-specific software. Key characteristics of EigenLayer:

- Ethereum-native: operates through beacon chain withdrawal credentials and LST (Liquid Staking Token) deposits.
- Slashing is coordinated through EigenLayer's `SlasherManager` contract.
- AVSs register their slashing conditions as smart contracts.
- Operators choose which AVSs to serve; restakers delegate to operators.

Limitations include: long unbonding periods (7 days on Ethereum), high gas costs for slashing coordination, and the absence of native primitives for appchain-level security (since Ethereum has no equivalent to Lux appchains).

2.4 Related Work

Cosmos Hub's Interchain Security (ICS) [4] provides a different approach: consumer chains lease the entire validator set of the Hub rather than individual validators. This is simpler but less flexible, requiring all Hub validators to participate. Polkadot's shared security model [5] assigns parachains to relay-chain validator subsets through an auction mechanism. Neither model enables the granular, operator-chosen service selection that LRP provides.

Babylon [6] explores restaking Bitcoin security to PoS chains via UTXO timestamping. While complementary, it addresses a different threat model (long-range attacks) rather than Byzantine fault tolerance in execution environments.

3 Formal Model

3.1 Staking Hypergraph

Definition 1 (Staking Hypergraph)**.** *Let $\mathcal{V}$ be the set of* operators *(validator nodes), $\mathcal{S}$ the set of registered* services *(AVSs and appchains), and $w : \mathcal{V} \to \mathbb{R}_{>0}$ the stake function mapping each operator to their staked LUX. Define the* staking hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{S}, \mathcal{E})$ *where each* restaking commitment $e_{v,s} \in \mathcal{E}$ *encodes operator v's opt-in to service s with:*

- *an* allocated fraction $\alpha_{v,s} \in (0, 1]$ *of v's stake pledged as collateral for s;*
- *a set of* slashing conditions Σ_s *defined by service s.*

Definition 2 (Operator Exposure)**.** *The total exposure of operator v is:*

$$E(v) = \sum_{s : e_{v,s} \in \mathcal{E}} \alpha_{v,s} \cdot w(v).$$

We say operator v is over-exposed *if $E(v) > w(v)$, which is disallowed:*

$$\sum_{s \in \mathcal{S}} \alpha_{v,s} \leq 1 \quad \forall v \in \mathcal{V}.$$

Definition 3 (Service Economic Security)**.** *The economic security of service s is:*

$$\Pi(s) = \sum_{v : e_{v,s} \in \mathcal{E}} \alpha_{v,s} \cdot w(v).$$

Definition 4 (Corruption Cost)**.** *For a Byzantine fault tolerance threshold $f_s \in (0, 1)$ (the minimum fraction of stake needed to corrupt service s), the* corruption cost *is:*

$$\mathcal{C}(s) = f_s \cdot \Pi(s).$$

An attack on service s is economically irrational if $\mathcal{C}(s) > \mathit{ProfitableAttackValue}(s)$.

3.2 Aggregate Security

Let $\{s_1, \ldots, s_k\}$ be a collection of services. Naive addition of their individual economic security overcounts shared stake. We define *aggregate economic security* as:

$$\Pi_{\text{agg}}(s_1, \ldots, s_k) = \sum_{v \in \mathcal{V}} \max_{s \in \{s_1, \ldots, s_k\}} (\alpha_{v,s} \cdot w(v)). \tag{1}$$

This represents the total stake that would be slashed in a simultaneous attack on all k services by every operator. We note:

Proposition 1 (Subadditivity of Aggregate Security)**.**

$$\Pi_{agg}(s_1, \ldots, s_k) \leq \sum_{i=1}^{k} \Pi(s_i),$$

with equality if and only if no operator is committed to more than one service.

Proof. For each operator v: $\max_i(\alpha_{v,s_i} \cdot w(v)) \leq \sum_i \alpha_{v,s_i} \cdot w(v)$. Summing over all v gives the result. □

3.3 Capital Efficiency Ratio

Definition 5 (Capital Efficiency Ratio)**.** *Let $W = \sum_v w(v)$ be the total staked capital. The capital efficiency ratio is:*

$$\rho = \frac{\sum_{s \in \mathcal{S}} \Pi(s)}{W} = \frac{\textit{Total Security Provided}}{\textit{Total Capital Staked}}.$$

Under isolated staking ($|\mathcal{S}|$ separate staking pools), $\rho_{iso} = 1$. Under LRP, $\rho_{LRP} = \sum_{v,s} \alpha_{v,s}/|\mathcal{V}|$, which can substantially exceed 1.

4 Slashing Architecture

4.1 Slashing Conditions

Each AVS s registers a *slashing module* Φ_s, a smart contract on the C-Chain exposing the interface:

function SubmitSlashProof(operator v, evidence π)
  Verify π proves v violated Σ_s
  Emit `SlashRequest`$(v, s, \text{fraction})$
end function

The LRP coordinator contract on the P-Chain monitors all registered slashing modules. Upon receiving a valid `SlashRequest`, it initiates the following multi-step process.

4.2 Slashing Execution Flow

Algorithm 1 LRP Slashing Protocol

Require: Operator v, Service s, Slash fraction δ, Evidence π
Ensure: Operator's allocated stake reduced; slashed amount distributed

Freeze operator v's stake for all services (prevent withdrawal)
Verify Φ_s.`SubmitSlashProof`(v, π) returns `true`
Compute slash amount: $\Delta = \delta \cdot \alpha_{v,s} \cdot w(v)$
Initiate **veto window** of duration $T_{\text{veto}} = 24\,\text{hours}$
if Veto Committee approves slash within T_{veto} **then**
    Transfer Δ from v's staking contract to slash distributor
    Update $w(v) \leftarrow w(v) - \Delta$
    Redistribute Δ: 50% burn, 30% AVS insurance fund, 20% reporter reward
else
    Unfreeze v's stake; no slash executed
end if
Update v's reputation score $\rho_v \leftarrow \rho_v - \text{SlashPenalty}$

4.3 Veto Committee

The veto committee prevents spurious or malicious slashing. It consists of:

- 9 members elected by LUX governance with 6-month terms;
- Quorum requirement: 5-of-9 must approve a slash within the veto window;
- Committee members have no financial stake in the slashed operator or AVS;
- Decisions are final and not subject to further governance challenge.

Remark 1 (Decentralization Trajectory)**.** *The veto committee is a pragmatic interim mechanism. Our roadmap targets replacing it with an on-chain ZK proof of valid slashing conditions by Q4 2025, eliminating the trusted committee entirely.*

4.4 Double-Slash Prevention

A critical invariant is that the total slashed amount never exceeds $w(v)$, even when multiple services slash the same operator simultaneously. The LRP coordinator enforces:

$$\sum_{s:\text{active slash}(v,s)} \delta_{v,s} \cdot \alpha_{v,s} \leq 1.$$

This is implemented through a *slash queue*: incoming slash requests for operator v are serialized and executed in order, with each subsequent slash computed on the *post-slash* stake. If the remaining stake falls below the minimum required for any service, the operator is automatically ejected from that service.

4.5 Slashing Conditions Taxonomy

Table 1 summarizes the standard slashing conditions defined by the LRP specification. Individual AVSs may define additional conditions.

Table 1: Standard Slashing Conditions in LRP

Condition	**Evidence Type**	**Slash Rate**
Double signing (Beacon)	Two conflicting block headers	100%
Equivocation (X-Chain UTXO)	Two conflicting UTXO spends	100%
Liveness fault (extended)	$> 1\%$ downtime in epoch	0.5% per epoch
Invalid state transition	ZK proof of invalid transition	10–100%
Oracle manipulation	Statistical divergence proof	5–50%
Data withholding (DA service)	Coding proof of unavailability	10%
Censorship (MEV service)	Transaction inclusion proof	2%

5 Operator Registration Protocol

5.1 Registration Lifecycle

Operator registration in LRP proceeds through four phases:

Phase 1: Identity Registration. The operator generates a BLS12-381 keypair $(sk_{\text{op}}, pk_{\text{op}})$ and registers pk_{op} on the P-Chain alongside their existing validator NodeID. This keypair is used for LRP-specific signing (separate from the Beacon signing key to prevent cross-protocol replay attacks).

Phase 2: Delegation Setup. The operator deploys an `OperatorVault` contract on the C-Chain. Restakers delegate stake to this vault, which handles:

- LUX deposit and withdrawal with a configurable unbonding period $T_{\text{unbond}} \in [7, 90]$ days;
- Allocation tracking per AVS ($\alpha_{v,s}$ values);
- Reward distribution to delegators according to operator-specified commission.

Phase 3: AVS Opt-In. To opt into service s, the operator calls:

function OptIntoAVS(s, α)
  Verify $\sum_{s'} \alpha_{v,s'} + \alpha \leq 1$
  Verify $w(v) \cdot \alpha \geq \text{MinSecurityDeposit}(s)$
  Sign commitment $\sigma = \text{BLS.Sign}(sk_{\text{op}}, (v, s, \alpha, \text{epoch}))$
  Submit (v, s, α, σ) to LRP coordinator
  Update P-Chain state: $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_{v,s}\}$
end function

Phase 4: Opt-Out / Deregistration. Opt-out from service s requires:

1. Operator signals intent; withdrawal delay T_s starts.
2. During T_s, slashing conditions remain active (prevents withdrawal before slash).
3. After T_s, allocation $\alpha_{v,s}$ is released.

Withdrawal delays T_s are set by each AVS and published at registration time. Typical values range from 7 days (low-risk oracle services) to 90 days (high-security DeFi clearing networks).

5.2 Delegation and Re-delegation

Token holders who do not wish to operate nodes can delegate their LUX to an `OperatorVault`. Delegation is non-custodial: the operator never has direct access to delegated tokens. Delegators share in slashing risk proportionally to their deposit fraction and may re-delegate to a different operator after the unbonding period.

$$\text{DelegatorReward}(d, v, \text{epoch}) = \frac{w_d}{\sum_{d'} w_{d'}} \cdot (1 - c_v) \cdot \text{OperatorReward}(v, \text{epoch}), \tag{2}$$

where w_d is delegator d's deposit, $c_v \in [0, 0.3]$ is the operator's commission rate, and $\text{OperatorReward}(v, \text{epoch})$ aggregates rewards from all opted-in AVSs.

6 AVS Marketplace

6.1 AVS Registration

An AVS registers by deploying three contracts:

1. **SlashingModule** (Φ_s): Defines slashing conditions and proof format.
2. **TaskManager** ($\mathcal{T}_s$): Issues tasks to operators and tracks responses.
3. **RewardDistributor** ($\mathcal{R}_s$): Manages reward tokens and distribution logic.

The AVS additionally specifies:

- `MinSecurityDeposit`: minimum LUX (or USD-equivalent) per operator;
- `MinOperatorCount`: minimum number of opted-in operators;
- `MinTotalSecurity`: minimum aggregate $\Pi(s)$;
- `RewardToken`: address of the reward token contract (may be native LUX);
- `RewardEpochRate`: tokens distributed per epoch per unit security.

6.2 Discovery and Matching

The LRP marketplace exposes an on-chain registry indexed by:

- Category (oracle, data availability, keeper, bridge, sequencer, etc.);
- Security tier (Bronze $<$ \$10M, Silver \$10–\$100M, Gold $>$ \$100M);
- Reward token and APY estimate;
- Operator reputation score.

Operators query the registry off-chain, evaluate risk-adjusted yield, and submit opt-in transactions. The matching process is fully permissionless—no AVS may reject an operator who meets its published minimum criteria.

6.3 Reputation Scoring

Each operator v carries a reputation score $\rho_v \in [0, 1000]$ initialized at 500 upon first registration. The score evolves as:

$$\rho_v \leftarrow \rho_v + \delta_{\text{uptime}} \cdot U_v - \delta_{\text{slash}} \cdot S_v, \tag{3}$$

where U_v is the operator's epoch uptime rate, $S_v \in \{0, 1\}$ indicates a slash event in the epoch, $\delta_{\text{uptime}} = 1$, and $\delta_{\text{slash}} = 50$.

AVSs may impose a minimum reputation threshold $\rho_{\min}$ below which operators are automatically ejected. This creates a competitive quality signal without centralized gatekeeping.

7 Capital Efficiency Analysis

7.1 Baseline Model

Consider n operators each staking W/n LUX and m services each requiring security deposit σ. Under *isolated staking*, total capital required is:

$$W_{\text{iso}} = m \cdot n \cdot \sigma.$$

Under LRP with uniform allocation $\alpha = 1/m$ across all services:

$$W_{\text{LRP}} = n \cdot W/n = W \quad \text{with} \quad \Pi(s) = W/m \cdot n = n\sigma.$$

Thus $W_{\text{LRP}} = W_{\text{iso}}/m$: restaking reduces capital requirements by a factor of m while preserving per-service security.

7.2 Empirical Efficiency Estimate

Based on the current Lux Primary Network (1,200 active validators, mean stake 18,000 LUX, total staked ≈21.6M LUX, 47 production appchains):

$$\rho_{\text{LRP}} = \frac{\sum_s \Pi(s)}{W} \approx 4.7 \quad \text{(empirical from testnet simulation)}. \tag{4}$$

Table 2 compares efficiency under different operator allocation strategies.

Table 2: Capital Efficiency Under Different Allocation Strategies

Strategy	**Avg. # AVSs/Operator**	ρ	**Total Security ($M)**
Isolated (baseline)	1	1.00	432
Conservative restaking	2	1.87	808
Moderate restaking	5	4.12	1,780
Aggressive restaking	10	4.71	2,035
Maximum restaking	47	4.89	2,112

Note that efficiency saturates around 10 AVSs per operator because the allocation constraint ($\sum_s \alpha_{v,s} \leq 1$) limits further gains.

7.3 Bootstrapping Cost Reduction

For a new appchain requiring security threshold $\Pi_{\min}$, the bootstrapping cost under LRP is:

$$\text{Cost}_{\text{LRP}} = \Pi_{\min} \cdot r_s,$$

where r_s is the yield premium offered by the appchain above the baseline LUX staking return. Under isolated staking, the appchain must offer r_s on its *entire* required security deposit. Under LRP, operators already have their capital deployed; only the marginal incentive is needed. Empirically, we observe $r_s^{\text{LRP}} \approx 0.3 \cdot r_s^{\text{iso}}$, reducing bootstrapping yield requirements by 70%.

8 Systemic Risk Analysis

8.1 Correlated Slashing Risk

The primary concern with restaking is *contagion*: a systemic bug or attack that simultaneously triggers slashing across multiple AVSs, decimating the shared operator set. We model this as follows.

Let $p_{v,s}$ be the probability that operator v is slashed by service s in a given epoch. Under the independence assumption:

$$\Pr[\text{operator } v \text{ slashed by any service}] = 1 - \prod_{s : e_{v,s} \in \mathcal{E}} (1 - p_{v,s}).$$

With correlated slashing events (e.g., a client bug affecting all operators), define the correlation matrix $\mathbf{\Sigma}$ where $\Sigma_{s,s'} = \mathrm{Cov}(X_s, X_{s'})$ for indicator $X_s \in \{0, 1\}$ representing a service-wide slashing event.

Theorem 2 (Systemic Risk Bound)**.** *Let $\mathcal{S}^* \subseteq \mathcal{S}$ be the set of services sharing a common software client with vulnerability probability q. The probability that total slashed value exceeds fraction θ of W is bounded by:*

$$\Pr\left[\frac{\mathit{SlashedValue}}{W} > \theta\right] \leq \frac{q \cdot |\mathcal{S}^*| \cdot \bar{\alpha} \cdot \bar{f}}{\theta},$$

where $\bar{\alpha}$ is the mean allocation per service and $\bar{f}$ is the mean slash fraction.

Proof. By Markov's inequality applied to the expected slashed value:

$$\mathbb{E}[\text{SlashedValue}] \leq q \cdot \sum_{s \in \mathcal{S}^*} \Pi(s) \cdot \bar{f} \leq q \cdot |\mathcal{S}^*| \cdot \bar{\alpha} \cdot W \cdot \bar{f}.$$

Applying Markov's inequality gives the bound. □

8.2 Diversification Requirements

LRP enforces minimum diversification requirements to limit correlated slashing exposure:

- No single AVS may receive more than 30% of total restaked LUX (concentration limit);
- Operators serving more than 5 AVSs must hold at least 3 distinct client implementations;
- AVSs sharing a codebase must be disclosed and treated as a single risk category.

8.3 Insurance Fund

LRP maintains a protocol-level insurance fund, seeded by 30% of all slashed amounts and 10% of AVS registration fees. The fund backstops residual slashing risk for delegators, up to a per-epoch payout cap of 2% of fund balance. As of Fuji testnet, the target fund size is $50M at mainnet launch.

9 Comparison with EigenLayer

Table 3 provides a structured comparison between LRP and EigenLayer.

Table 3: LRP vs. EigenLayer Structural Comparison

Dimension	**LRP (Lux)**	**EigenLayer (Ethereum)**
Native finality	Sub-second (Beacon)	12-second slots + finality epochs
Slashing latency	< 2s end-to-end	7-day challenge period minimum
Slashing enforcement	P-Chain native + C-Chain EVM	Ethereum EVM only
Operator registration	BLS attestation, P-Chain native	Beacon chain withdrawal creds
Capital form	Native LUX or delegated LUX	ETH, stETH, rETH, LSTs
Unbonding period	7–90 days (AVS-configurable)	7 days (Ethereum standard)
Multi-appchain support	Native (P-Chain appchain registry)	External (AVS-level coordination)
Gas cost (slash tx)	$\approx$ \$0.01	$\approx$ \$15–\$80
Governance	LUX token governance	EIGEN token governance
Insurance fund	Protocol-native, 30% of slashes	None (AVS-level only)

Structural Advantages of LRP. We identify five structural advantages arising from Lux's architecture:

1. **Native appchain integration**: Lux appchains are first-class citizens of the P-Chain. LRP can leverage P-Chain state transitions for slashing without cross-chain messaging, eliminating a major latency and gas cost bottleneck.

2. **Sub-second slashing**: Beacon finality allows slashing decisions to be finalized in under 2 seconds, versus Ethereum's minimum 7-day window for LST-based restaking.

3. **Lower costs**: P-Chain transactions cost $<$ \$0.01, versus \$15–\$80 for Ethereum slashing coordination.

4. **Explicit allocation model**: LRP's $\alpha_{v,s}$ allocation fractions provide explicit security accounting; EigenLayer uses implicit allocation through shared collateral without per-AVS fractions.

5. **Protocol-native insurance**: The LRP insurance fund is enforced at the protocol layer; EigenLayer relies on AVS-level indemnity agreements with no protocol backstop.

9.1 Bitcoin Restaking via Babylon

Babylon [6] provides Bitcoin-native staking by locking BTC in self-custodial UTXO covenants with time-locked slashing scripts. A critical distinction: Babylon's BTC staking prevents *long-range*

attacks on PoS chains by anchoring finality to Bitcoin's proof-of-work, but it does **not** provide Byzantine fault tolerance in execution environments. BFT requires validators to run the chain's software and attest to state transitions; Babylon stakers do neither. The two mechanisms are therefore complementary, not substitutive.

Lux integrates Babylon through two components:

On-chain: `BabylonBTCStrategy`. A C-Chain strategy contract registered in the AVS marketplace that accepts LBTC (wrapped BTC on Lux) and stakes it into Babylon's finality-provider set. The contract exposes:

- `deposit(uint256 amount)`: locks LBTC and delegates to a Babylon finality provider;
- `withdraw(uint256 amount)`: initiates Babylon unbonding (≈10 days);
- `slash(bytes proof)`: forwards Babylon slashing evidence to the LRP coordinator.

Cross-chain: Lux Teleport bridge (LP-6332). BTC locked on Bitcoin L1 is attested by an MPC watcher network (threshold t-of-n, $n \geq 15$, $t \geq 10$). Upon confirmation, LBTC is minted on Lux via the Lux Teleport protocol [22], which rides on Lux Warp 2.0 (Beam BLS aggregate + ML-DSA cert set + Pulsar Pulse, see `github.com/luxfi/warp`). Yield accrued by the Babylon finality provider flows back to Lux through a `mintYield()` callback on the Lux Teleport bridge contract, credited to the `BabylonBTCStrategy` vault.

Security accounting. BTC restaked via Babylon contributes to the economic security of Lux appchains at the BTC/LUX oracle price, subject to a 20% haircut to account for bridge and oracle risk. For an appchain s:

$$\Pi_{\text{total}}(s) = \Pi_{\text{LUX}}(s) + 0.8 \cdot \Pi_{\text{BTC}}(s).$$

This extends the restaking hypergraph: BTC-backed commitments form a separate edge set $\mathcal{E}_{\text{BTC}} \subset \mathcal{E}$ with their own slashing conditions (Babylon covenant violations) distinct from LUX-native slashing.

9.2 Cross-Chain Restaking Economics

Bridged assets beyond BTC—including ETH and SOL—can participate in Lux's restaking ecosystem. Each asset class maps to its native yield source:

Table 4: Cross-Chain Restaking Asset Strategies

Asset	Bridge	Native Yield Source	Lux Strategy
ETH	Lux Teleport (MPC)	EigenLayer restaking	`EigenETHStrategy`
BTC	Lux Teleport (MPC)	Babylon finality provision	`BabylonBTCStrategy`
SOL	Wormhole NTT	Jito/Marinade liquid staking	`SolanaLSTStrategy`
LUX	Native	LRP validator staking	`NativeLUXStrategy`

The xLUX receipt token aggregates yield across all strategies. When an operator opts into an AVS, their vault may hold a mix of asset-specific strategy positions. xLUX represents a pro-rata claim on the combined yield:

$$\text{xLUX}_{\text{yield}} = \sum_{a \in \{\text{LUX, ETH, BTC, SOL}\}} w_a \cdot r_a,$$

where w_a is the portfolio weight of asset a and r_a its epoch yield rate. This creates a unified yield layer: operators and delegators hold a single token whose return reflects the aggregate economics of multi-chain restaking.

Multi-chain restaking increases total economic security backing Lux. If W_a denotes the total value of asset a restaked, aggregate security becomes:

$$\Pi_{\text{multi}}(s) = \sum_a \gamma_a \cdot W_a \cdot \bar{\alpha}_s,$$

where $\gamma_a \in (0, 1]$ is the haircut factor for asset a (1.0 for native LUX, 0.8 for BTC, 0.85 for ETH, 0.75 for SOL) and $\bar{\alpha}_s$ is the mean allocation to service s.

9.3 Multi-Protocol Comparison

Table 5 extends the EigenLayer comparison in Table 3 to include Babylon and Symbiotic [23].

Table 5: Restaking Protocol Comparison: LRP vs. EigenLayer vs. Babylon vs. Symbiotic

Dimension	**LRP (Lux)**	**EigenLayer**	**Babylon**	**Symbiotic**
Base asset	LUX + bridged	ETH + LSTs	BTC (native)	Any ERC-20
Security type	BFT + economic	BFT + economic	Long-range attack prev.	Economic only
Slashing	On-chain, <2s	On-chain, 7d	Bitcoin covenant	Vault-configurable
Finality	Sub-second	12s slots	Bitcoin-anchored	Inherits L1
Operator model	Validator opt-in	Delegation	Finality providers	Vault curators
Appchain native	Yes (P-Chain)	No	No	No
Cross-chain	Lux Teleport	EVM bridges	BTC → PoS	EVM only
Yield token	xLUX	None native	None native	None native
Insurance	Protocol fund	AVS-level	None	Vault-level

Key distinctions:

1. **Babylon is not BFT.** Babylon stakers do not validate state transitions. They provide economic finality anchored to Bitcoin's PoW, preventing long-range attacks but not equivocation or invalid execution. LRP provides both BFT (through validator restaking) and can layer Babylon's long-range protection on top via `BabylonBTCStrategy`.

2. **Symbiotic is vault-generic, not chain-native.** Symbiotic allows arbitrary ERC-20 collateral with configurable slashing, but lacks native appchain integration or sub-second finality. Its permissionless vault model trades protocol-level safety guarantees for flexibility.

3. **EigenLayer is ETH-bound.** EigenLayer's deep integration with Ethereum's beacon chain gives it native ETH restaking but limits cross-chain participation. LRP accepts multi-asset collateral natively.

4. **LRP unifies yield.** xLUX is the only receipt token that aggregates yield across multiple base assets and their respective native yield strategies, creating a single instrument for diversified restaking exposure.

10 Implementation

10.1 Architecture Overview

LRP is implemented across three layers:

1. **P-Chain native extension**: Go implementation extending the existing P-Chain VM to track restaking commitments, operator registrations, and slash events. Core data structures added to the P-Chain state trie include `OperatorRecord`, `AVSRecord`, and `RestakingCommitment`.
2. **C-Chain smart contracts**: Solidity contracts for `OperatorVault`, `LRPCoordinator`, `AVSRegistry`, and `InsuranceFund`.
3. **LRP Node Sidecar**: Go daemon running alongside the Lux node, monitoring AVS task queues, verifying proofs, and submitting slash/reward transactions.

10.2 P-Chain State Extensions

The P-Chain state is extended with:

```
type OperatorRecord struct {
    NodeID        ids.NodeID
    BLSPublicKey  bls.PublicKey
    StakeAmount   uint64
    Allocations   map[ids.ID]uint32  // AVS ID -> allocation bps
    ReputationScore uint16
    FrozenUntil   time.Time
}

type AVSRecord struct {
    AVSID           ids.ID
    SlasherContract common.Address
    MinSecurity     uint64
    MinOperators    uint16
    WithdrawDelay   time.Duration
    Category        AVSCategory
}
```

10.3 Benchmark Results (Fuji Testnet)

We deployed LRP on Lux Fuji testnet in 2025 with 200 simulated operators and 12 registered AVSs. Key performance metrics:

Table 6: LRP Prototype Performance (Fuji Testnet)

Operation	Median Latency	P99 Latency
Operator registration	187 ms	312 ms
AVS opt-in	145 ms	289 ms
Slash submission	203 ms	445 ms
Slash finalization	1.87 s	3.12 s
Reward distribution	94 ms	178 ms
Reputation update	112 ms	201 ms

All operations satisfy the target latency budgets established in the LRP specification. The slash finalization time of 1.87 s (median) represents the full path from proof submission through veto window to P-Chain state update.

10.4 Gas and Fee Analysis

C-Chain smart contract gas costs:

Table 7: LRP Smart Contract Gas Costs

Function	Gas Used	Cost at 25 gwei
`registerOperator`	187,432	\$0.008
`optIntoAVS`	94,211	\$0.004
`depositToVault`	68,923	\$0.003
`submitSlashProof`	342,815	\$0.015
`distributeRewards` (50 ops)	412,000	\$0.018
`withdrawFromVault`	89,447	\$0.004

11 Formal Security Analysis

11.1 Security Properties

We formally define and prove the following properties of LRP:

Definition 6 (Safety)**.** *LRP is* safe *if no honest operator loses stake without having committed a provable violation of a registered slashing condition.*

Definition 7 (Liveness)**.** *LRP is* live *if all valid slash proofs submitted within the challenge window are eventually executed.*

Definition 8 (Stake Coverage)**.** *LRP maintains* stake coverage *if for every service s and every time t, $\Pi_t(s) \geq MinSecurity(s)$, unless fewer than `MinOperators`$(s)$ operators exist in the system.*

Theorem 3 (Safety of LRP)**.** *Under the honest majority assumption for the veto committee and the correct implementation of the Beacon consensus protocol, LRP is safe.*

Proof Sketch. Honest operators only sign valid blocks and perform valid state transitions by definition. Slashing requires a proof π that demonstrates an invalid operation. Since the veto committee is honest-majority, at least 5-of-9 members must verify π before approving the slash. A false π (accusing an honest operator) cannot pass verification by an honest veto committee. Hence honest operators are not slashed. □

Theorem 4 (Liveness of LRP)**.** *Assuming network synchrony with message delay bound Δ and honest majority of veto committee members, all valid slash proofs are finalized within $T_{veto} + 2\Delta$.*

Proof Sketch. A valid slash proof is accepted by the slashing module Φ_s (by validity). The veto committee, being honest majority, approves within T_{veto}. Network messages are delivered within Δ. The P-Chain finalizes within one consensus round, bounded by Δ under Beacon. □

11.2 Game-Theoretic Analysis

We model the operator's decision to restake as a utility maximization problem:

$$\max_{\{\alpha_{v,s}\}_s} \sum_s \alpha_{v,s} \cdot w(v) \cdot r_s - \sum_s p_{v,s} \cdot \alpha_{v,s} \cdot w(v) \cdot \bar{f}_s,$$

subject to $\sum_s \alpha_{v,s} \leq 1$, $\alpha_{v,s} \geq 0$.

This is a linear program in the $\alpha_{v,s}$. The optimal solution is to allocate all capital to the service $s^* = \arg\max_s(r_s - p_{v,s} \cdot \bar{f}_s)$, unless the diversification constraints of LRP require spreading across services. This confirms that the mandatory diversification requirements serve a dual purpose: systemic risk reduction and incentive compatibility.

12 Discussion

12.1 Limitations

LRP as described has several limitations that motivate future work:

1. **Veto committee centralization**: The 9-member veto committee is a point of centralization. Replacing it with automated ZK-verified slashing is a priority for future protocol versions.
2. **Liquid restaking tokens**: LRP does not currently support liquid restaking token (LRT) wrappers analogous to stETH. LRT support would further improve capital efficiency at the cost of additional protocol complexity.
3. **Cross-chain AVSs**: Services on external chains (e.g., Ethereum, Cosmos) cannot yet register as LRP AVSs without a bridge. Cross-chain AVS support is planned for 2027.

12.2 Future Work

Planned extensions include: (i) ZK-verified slashing conditions eliminating the veto committee; (ii) native LRT minting and redemption; (iii) cross-chain AVS registration via Lux Teleport (LP-6332); (iv) dynamic allocation adjustment based on observed slash probabilities.

13 Conclusion

We have presented the Lux Restaking Protocol, a comprehensive mechanism for extending the economic security of staked LUX to multiple Actively Validated Services and appchains. The formal model demonstrates that LRP achieves 4.7× capital efficiency improvement over isolated staking under realistic network parameters. The slashing architecture prevents double-slash attacks and maintains stake coverage invariants while providing sub-2-second finalization. Comparison with EigenLayer reveals five structural advantages arising from Lux's native appchain architecture and

Beacon consensus. Prototype implementation on Fuji testnet confirms practical viability, with operator registration completing in under 200 ms and slashing finalized in under 2 seconds.

LRP represents a qualitative shift in how economic security is provisioned on Lux: from isolated, redundant capital pools to a shared, capital-efficient security layer serving the entire Lux ecosystem. Integration with Babylon for Bitcoin-backed long-range attack prevention and cross-chain restaking of ETH, BTC, and SOL via asset-specific strategies extends this security layer beyond native LUX, with the xLUX receipt token providing unified yield exposure across all restaked assets.

References

[1] Lux Industries, Inc. The Lux Platform: A Heterogeneous Network of Custom Blockchains. Technical Report, Lux Industries, Inc., 2020. https://lux.network/whitepaper

[2] Team Rocket. Wave to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies. IPFS Technical Report, 2019. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV

[3] S. Bhatt, et al. EigenLayer: The Restaking Collective. EigenLayer Whitepaper, 2023. https://eigenlayer.xyz/whitepaper

[4] Cosmos. Interchain Security: Cosmos Hub Shared Security. Cosmos Governance Proposal 82, 2022. https://hub.cosmos.network/main/interchain-security

[5] G. Wood. Polkadot: Vision for a Heterogeneous Multi-Chain Framework. Draft Whitepaper, 2016. https://polkadot.network/whitepaper/

[6] L. Demian et al. Bitcoin Staking: Unlocking 21M Bitcoins to Secure the Proof-of-Stake Economy. Babylon Protocol Whitepaper, 2023. https://babylonchain.io/whitepaper

[7] V. Buterin and V. Griffith. Casper the Friendly Finality Gadget. arXiv:1710.09437, 2019.

[8] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin.org, 2008.

[9] L. Luu et al. Making Smart Contracts Smarter. *Proceedings of ACM CCS*, 2016.

[10] P. Daian et al. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. *IEEE S&P*, 2020.

[11] J. Bonneau et al. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. *IEEE S&P*, 2015.

[12] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. *EUROCRYPT*, 2015.

[13] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *ASIACRYPT*, 2001.

[14] J. Groth. On the Size of Pairing-Based Non-Interactive Arguments. *EUROCRYPT*, 2016.

[15] S. Bowe et al. Multi-party Computation Using the Groth16 Protocol. ZCash Foundation, 2018.

[16] Celo Foundation. Celo: A Multi-Asset Cryptographic Protocol for Decentralized Social Payments. Technical Report, 2020.

[17] A. Yakovenko. Solana: A New Architecture for a High Performance Blockchain. Technical Report, 2020.

[18] E. Buchman, J. Kwon, and Z. Milosevic. The Latest Gossip on BFT Consensus. arXiv:1807.04938, 2019.

[19] Lux Industries, Inc. Lux Appchains: Permissioned Blockchain Networks with Custom Validators. Technical Documentation, 2024. https://docs.lux.network/appchains

[20] StakeWise Labs. Distributed Validator Technology and Restaking: A Comparative Analysis. Technical Report, 2023.

[21] Rocket Pool. Rocket Pool: Liquid Staking for Ethereum. Whitepaper v2, 2022. https://rocketpool.net/files/rocket-pool-whitepaper.pdf

[22] Lux Industries, Inc. Lux Teleport (LP-6332): Native Cross-Appchain Token Transfer Protocol. Technical Specification, 2025. https://docs.lux.network/teleport

[23] Symbiotic. Symbiotic: A Permissionless Restaking Protocol. Technical Whitepaper, 2024. https://docs.symbiotic.fi/