LSS: A Universal Lifecycle Framework for Threshold Cryptographic Protocols

Dynamic share lifecycle for ECDSA, FROST/Lens, and lattice/Pulsar schemes

Zach Kelling

Lux Industries

2024

Abstract

Lux is not merely adding post-quantum signatures to a chain; it defines a hybrid finality architecture for DAG-native consensus, with protocol-agnostic threshold lifecycle, post-quantum threshold sealing, and cross-chain propagation of Horizon finality. LSS is the protocol-agnostic lifecycle framework that powers that systems layer.

We present **LSS** (Linear Shamir Secret Sharing), a universal dynamic-resharing extension for threshold cryptographic protocols. LSS is *not* a signature scheme; it is the lifecycle / orchestration framework that manages share generations, snapshot history, rollback, dealer/coordinator role separation, and live-resharing wire protocol — across multiple threshold-kernel families.

The principal innovation is coordinator-driven verifiable resharing via Joint Verifiable Secret Sharing (JVSS), enabling transition from (t, n) to $(t', n \pm k)$ without reconstructing the master key or interrupting signing operations. LSS solves three operational challenges that the underlying signature schemes leave open: (1) *dynamic resharing* — live expansion and contraction of the signing group without downtime or key reconstruction; (2) *automated fault tolerance* — node eviction, state rollback to previously certified shard generations, and persistence-backed recovery; (3) *multi-kernel adaptation* — a unified adapter pattern that gives the same lifecycle to ECDSA-CMP, FROST/Schnorr, and lattice threshold schemes.

The framework's adapters cover three kernel families:

- `lss_cmp.go`: ECDSA-CMP adapter (Protocol I localized nonce blinding, Protocol II collaborative nonce blinding; sub-25ms signing).
- `lss_lens.go`: classical curve threshold via Lens, replacing the placeholder `lss_frost.go` stub.
- `lss_pulsar.go`: lattice threshold via Pulsar; Generation/RollbackFrom/KeyEraID lineage; activation cert under unchanged group public key; 10/10 acceptance tests pass.

The classical foundations are HJKY proactive secret sharing, Desmedt–Jajodia secret redistribution, and Wong–Wang–Wing verifiable redistribution. The novel work is the unified adapter architecture: *one lifecycle framework, multiple threshold kernels.*

Production-deployed in the Lux Teleport omnichain bridge securing cross-chain assets across 270+ connected blockchains, and adopted by Quasar consensus for dynamic post-quantum threshold finality with rotating validator sets.

Contents

1 Introduction

Threshold signature schemes enable a group of n parties to collectively produce signatures under a shared public key, such that any subset of t or more parties can sign while any subset of $t-1$ or fewer cannot. Two dominant protocols exist:

- **CGGMP21** [5]: Threshold ECDSA with identifiable aborts, producing standard (r, s, v) signatures compatible with Ethereum, Bitcoin, and all secp256k1 chains.
- **FROST** [4]: Two-round Schnorr threshold signatures for Ed25519 and BIP-340 Taproot, compatible with Solana, TON, Cosmos, Polkadot, and Bitcoin Taproot.

Both protocols assume a *static* signer set: the participants are fixed at key generation time and cannot be changed without generating a new key pair. This is operationally infeasible for production systems requiring 24/7 uptime, where nodes must be rotated for maintenance, security incidents, or capacity changes.

Our Contribution. LSS solves this by providing a *live resharing* protocol that enables:

1. Adding new signers to an existing group without downtime.
2. Removing signers (including compromised nodes) without key reconstruction.
3. Changing the threshold t while maintaining the same public key.
4. Automatic rollback to previous shard generations on signing failure.
5. Unified multi-chain support through pluggable `SignerAdapter` interface.

The key insight is that Shamir's Secret Sharing [1] is *linear*: shares of a secret s under polynomial $f(x)$ of degree $t-1$ can be *re-shared* into a new polynomial $g(x)$ of degree $t'-1$ distributed to a potentially different set of n' participants, without ever reconstructing s in the clear. JVSS [2] makes this verifiable.

2 Preliminaries

2.1 Shamir's Secret Sharing

Definition 1 ((t, n)-Shamir Sharing)**.** *A dealer chooses a random polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ over $\mathbb{Z}_q$ where $f(0) = s$ is the secret. Share i is $s_i = f(i)$ for $i \in \{1, \ldots, n\}$. Any t shares determine f (and thus s) via Lagrange interpolation; any $t-1$ shares reveal no information about s.*

2.2 Linearity Property

The critical property enabling LSS is linearity:

$$f(x) + g(x) = (s_f + s_g) + (a_1^f + a_1^g)x + \cdots \tag{1}$$

If parties hold shares of f and independently generate shares of a polynomial g with $g(0) = 0$, the sum produces a fresh sharing of the same secret s_f under a new polynomial. This enables:

- **Refresh**: Same (t, n), new polynomial, same secret.
- **Reshare**: New (t', n'), new polynomial, same secret.

2.3 Joint Verifiable Secret Sharing (JVSS)

JVSS [2] adds verifiability: each party publishes Pedersen commitments $C_k = g^{a_k} h^{b_k}$ for each coefficient of their polynomial. Other parties verify their shares against these commitments. A complaint mechanism handles malicious dealers.

3 LSS Protocol

3.1 Architecture

LSS operates as a wrapper around CGGMP21 and FROST:

Component	Purpose
`Suite`	Unified entry point, protocol selection
`WithCMP(pool)`	CMP/CGGMP21 backend for ECDSA
`WithFROST(pool)`	FROST backend for Schnorr/EdDSA
`Coordinator`	Manages resharing ceremonies
`Dealer`	Distributes JVSS shares during reshare
`Rollback`	Maintains generation history
`SignerAdapter`	Chain-specific signature formatting

3.2 Dynamic Resharing Protocol

Theorem 1 (Reshare Correctness)**.** *After protocol execution, the new shares $\{s'_j\}_{j \in P_{new}}$ form a valid (t', n')-Shamir sharing of the same secret $s = f(0)$, and the group public key $Y = g^s$ is unchanged.*

Proof. By linearity of Lagrange interpolation:

$$s'_j = \sum_{i \in P_{old}} g_i(j) = \sum_{i \in P_{old}} \lambda_i \cdot s_i \cdot L_{i,j}(t')$$

At $j = 0$:

$$\sum_{i \in P_{old}} g_i(0) = \sum_{i \in P_{old}} \lambda_i \cdot s_i = s$$

since $\sum \lambda_i s_i$ is Lagrange interpolation of f at $x = 0$. The new polynomial has degree $t' - 1$, so any t' shares suffice to recover s. The public key $Y = g^s$ is unchanged since s is unchanged. □ □

3.3 Fault Tolerance and Rollback

Definition 2 (Generation History)**.** *LSS maintains a chain of certified shard generations: $C_0 \to C_1 \to \cdots \to C_k$. If signing under generation k fails (node eviction, timeout, abort), the coordinator rolls back to C_{k-1} and retries.*

Theorem 2 (Rollback Safety)**.** *If generation C_{k-1} was certified (all parties confirmed valid shares), then rollback to C_{k-1} preserves the ability to sign with the original secret key s.*

Algorithm 1 LSS Dynamic Reshare: $(t, n) \to (t', n')$

Require: Current config C_{old} with shares $\{s_i\}_{i \in P_{old}}$, new set P_{new}, new threshold t'
Ensure: New config C_{new} with shares $\{s'_j\}_{j \in P_{new}}$, same public key Y

Phase 1: Coordinator Preparation
Coordinator broadcasts $(P_{new}, t', \text{generation}_{k+1})$ to all parties in $P_{old} \cup P_{new}$
Each old party $p_i \in P_{old}$ generates resharing polynomial $g_i(x)$ of degree $t' - 1$
$g_i(0) = \lambda_i \cdot s_i$ where λ_i is Lagrange coefficient for P_{old} subset
Phase 2: Verifiable Distribution (JVSS)
for all $p_i \in P_{old}$ **do**
    Compute commitments $\{C_{i,k} = g^{a_{i,k}}\}_{k=0}^{t'-1}$ and broadcast
    for all $p_j \in P_{new}$ **do**
        Send share $g_i(j)$ privately to p_j
    end for
end for
Phase 3: Verification and Aggregation
for all $p_j \in P_{new}$ **do**
    Verify each received share against commitments
    File complaint if verification fails
    Aggregate: $s'_j = \sum_{i \in P_{old}} g_i(j)$
end for
Phase 4: Certification
All parties verify $Y' = Y$ (public key unchanged)
Coordinator certifies generation $k + 1$, stores C_{new} in persistence layer
Previous generation k retained for rollback

3.4 Protocol Variants

Protocol I: Localized Nonce Blinding. Each party generates their own blinding factor for nonce computation. Lower latency (∼8ms) but requires an honest majority assumption for nonce generation.

Protocol II: Collaborative Nonce Blinding. Nonce blinding is jointly computed via another round of JVSS. Higher latency (∼25ms) but secure against dishonest majority up to $t - 1$ corruptions.

4 Multi-Chain Adaptation

LSS provides a unified `SignerAdapter` interface for chain-specific signature formatting:

```
type SignerAdapter interface {
    ChainType() string
    FormatSignature(r, s *big.Int, v byte) ([]byte, error)
    DeriveAddress(pubkey []byte) (string, error)
    ValidateTransaction(tx []byte) error
}
```

Chain	Backend	Sig Format	Latency
Ethereum/EVM	CMP (ECDSA)	(r, s, v) 65B	12ms
Bitcoin	FROST (Schnorr)	(R, s) 64B	8ms
Bitcoin Taproot	FROST Taproot	BIP-340 64B	8ms
Solana	FROST (Ed25519)	Ed25519 64B	8ms
TON	FROST (Ed25519)	Ed25519 64B	8ms
Cosmos	FROST (Ed25519)	Ed25519 64B	8ms
XRPL	CMP (ECDSA)	DER-encoded	12ms
Polkadot	FROST (Ed25519)	Sr25519/Ed25519	8ms
Cardano	FROST (Ed25519)	Ed25519 64B	8ms
Sui	FROST (Ed25519)	Ed25519 64B	8ms
Aptos	FROST (Ed25519)	Ed25519 64B	8ms
NEAR	FROST (Ed25519)	Ed25519 64B	8ms
Stellar	FROST (Ed25519)	Ed25519 64B	8ms
Tezos	FROST (Ed25519)	Ed25519 64B	8ms
Algorand	FROST (Ed25519)	Ed25519 64B	8ms
StarkNet	CMP (Stark)	Stark ECDSA	15ms
TRON	CMP (ECDSA)	(r, s, v) 65B	12ms

Table 1: Multi-chain signing via LSS adapters

5 Post-Quantum Extension

LSS integrates with Corona [6] lattice-based threshold signatures for post-quantum security. Corona uses Learning With Errors (LWE) to provide threshold signatures secure against quantum computers:

- **128-bit security**: $n = 512, q = 2^{32}$, signature ∼4KB
- **192-bit security**: $n = 768, q = 2^{48}$, signature ∼6KB
- **256-bit security**: $n = 1024, q = 2^{64}$, signature ∼8KB

The hybrid deployment uses classical (FROST/CMP) for day-to-day signing and Corona for finality certificates (via Quasar consensus [7]), ensuring both current efficiency and quantum-future safety.

6 Security Analysis

Theorem 3 (LSS Security)**.** *The LSS resharing protocol preserves the security of the underlying threshold scheme (CGGMP21 or FROST). Specifically:*

1. *An adversary corrupting $< t'$ parties after reshare cannot forge signatures.*
2. *An adversary corrupting $< t$ parties before reshare AND $< t'$ parties after reshare cannot forge (even with shares from both generations).*
3. *The secret s is never reconstructed in the clear during resharing.*

Theorem 4 (Forward Security)**.** *After a successful reshare from generation k to $k+1$, shares from generation k provide no advantage to an adversary attacking generation $k+1$, even if the adversary obtains all n shares from generation k after the reshare completes.*

7 Performance

Operation	**3-of-5**	**5-of-7**	**10-of-15**
Key Generation	12ms	28ms	82ms
Signing (Protocol I)	8ms	12ms	25ms
Signing (Protocol II)	15ms	22ms	40ms
Reshare	45ms	95ms	280ms
Rollback	< 1ms	< 1ms	< 1ms

Table 2: Performance benchmarks (Apple M1 Max, Go 1.22)

8 Deployment

LSS is production-deployed in the Lux Teleport omnichain bridge, where it provides:

- 2-of-3 default threshold for bridge MPC signing.
- Live signer rotation with 7-day timelock (users can exit during rotation).
- Automatic rollback on signing failure (coordinator detects abort, reverts to certified generation).
- Multi-chain signing for 270+ connected blockchains via unified adapter interface.
- Zero downtime during maintenance — nodes rotate one at a time without interrupting bridge operations.

9 Conclusion

LSS bridges the gap between theoretical threshold signature schemes and production operational requirements. By exploiting the linearity of Shamir's Secret Sharing, LSS enables live membership changes, automatic fault recovery, and unified multi-chain signing under a single framework wrapping both CGGMP21 and FROST. The integration with Pulsar post-quantum signatures provides a smooth upgrade path to quantum-resistant operations.

All source code is open-source at `github.com/luxfi/threshold/protocols/lss`.

References

[1] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[2] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *CRYPTO*, 1991.

[3] P. Feldman, "A Practical Scheme for Non-Interactive Verifiable Secret Sharing," in *FOCS*, 1987.

[4] C. Komlo and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures," in *SAC*, 2020.

[5] R. Canetti *et al.*, "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts," in *ACM CCS*, 2020.

[6] Z. Kelling, "Corona: Post-Quantum Lattice-Based Threshold Signatures," Lux Industries, Inc., 2024.

[7] Z. Kelling and V. Seesahai, "Quasar Consensus: Hybrid BLS + Corona Dual-Signature Finality," Lux Industries, Inc., 2022.

[8] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, 2007.

[9] P. Wuille *et al.*, "BIP-340: Schnorr Signatures for secp256k1," 2020.

[10] Z. Kelling and V. Seesahai, "Lux Teleport: Sovereign Omnichain Liquidity," Lux Industries, Inc., 2022.