Cryptographic Agility Architecture for Long-Lived Blockchain Systems

Zach Kelling

Lux Industries

2024

Abstract

Long-lived blockchain systems face a fundamental tension: cryptographic primitives have finite lifespans, but blockchain state is permanent. We present Lux Network's cryptographic agility architecture, enabling live rotation of signature schemes, hash functions, and key encapsulation mechanisms without chain halts, state migration, or consensus forks. The design introduces versioned key schedules, algorithm negotiation at the validator handshake layer, and a hybrid classical–post-quantum transition path that preserves backward verification of historical blocks while enforcing forward security for new ones. We describe the wire protocol extensions, the key schedule versioning scheme, and the operational procedures for executing a primitive rotation across a live validator set. The architecture has been deployed on Lux mainnet to migrate from ECDSA to hybrid BLS+ML-DSA without downtime.

1 Introduction

Cryptographic algorithms are not permanent. Advances in cryptanalysis, hardware capability, and standardization regularly obsolete primitives that were considered secure at deployment time. For ephemeral systems, this is manageable: rotate keys, update libraries, redeploy. For blockchains, the problem is structural. Every historical block is signed with the primitive that was current at its creation time. Validators must verify the entire chain, which means they must support every primitive ever used.

Lux Network's multi-chain architecture—spanning C-Chain (EVM), Q-Chain (platform management), X-Chain (DAG assets), and application-specific subnets—amplifies this challenge. Each chain may adopt new primitives on independent timelines, and cross-chain messages (Warp, Teleport) must remain verifiable across primitive boundaries.

This paper describes the cryptographic agility layer embedded in Lux's ZAP wire protocol [2] and consensus stack [1], enabling:

1. Algorithm negotiation during peer handshake without breaking existing connections.
2. Versioned key schedules that bind algorithm identifiers to epoch ranges.
3. Hybrid classical+PQ signing during transition periods.
4. Migration procedures that require no chain halt and no state export/import.

1.1 Threat Model

We assume an adversary who can:

- Harvest ciphertexts today for future quantum decryption (harvest-now-decrypt-later).
- Exploit implementation vulnerabilities in deprecated primitives.
- Observe the timing and structure of algorithm transitions to mount downgrade attacks.

The goal is to ensure that no single primitive compromise—past, present, or future—invalidates the chain's integrity guarantees.

Algorithm ID	Primitive	Status	Security
SIG_ECDSA_SECP256K1	ECDSA (secp256k1)	Deprecated	128-bit classical
SIG_BLS12_381	BLS (BLS12-381)	Active	128-bit classical
SIG_MLDSA_65	ML-DSA-65 (FIPS 204)	Active	128-bit PQ
SIG_RINGTAIL	Ringtail threshold	Active	128-bit PQ
HASH_SHA256	SHA-256	Active	128-bit classical
HASH_SHA3_256	SHA3-256	Active	128-bit classical
KEM_MLKEM_768	ML-KEM-768 (FIPS 203)	Active	128-bit PQ

Table 1: Excerpt from the Lux algorithm registry.

2 Architecture

2.1 Algorithm Registry

Every cryptographic primitive used in Lux is registered in a global algorithm registry with a unique identifier, version, status, and security classification.

Each algorithm transitions through four lifecycle states: `CANDIDATE` → `ACTIVE` → `DEPRECATED` → `REJECTED`. A governance proposal on Q-Chain triggers state transitions. Validators enforce that blocks produced after a deprecation epoch do not use deprecated primitives for new signatures, while still accepting them for historical verification.

2.2 Versioned Key Schedules

A **key schedule** binds a set of algorithm identifiers to an epoch range. Each validator maintains keys for all active schedules simultaneously.

Definition 1 (Key Schedule)**.** *A key schedule* $\mathcal{K}_v = (epoch_{start}, epoch_{end}, sig, hash, kem)$ *specifies the algorithms a validator* v *uses for signing, hashing, and key encapsulation during epochs* $[epoch_{start}, epoch_{end})$*.*

During a transition, two key schedules overlap. Validators sign blocks with both the outgoing and incoming signature schemes. Verifiers accept a block if *either* signature is valid during the overlap window, and *only* the new signature after the overlap closes.

```
function VerifyBlockSignature(block, epoch)
schedule ← KeyScheduleForEpoch(epoch)
if schedule.isTransition then
    return Verify(block, schedule.outgoing) ∨ Verify(block, schedule.incoming)
else
    return Verify(block, schedule.current)
end if
end function
```

2.3 Algorithm Negotiation in ZAP Handshake

The ZAP wire protocol [2] handshake is extended with an `AlgorithmSet` field. During connection establishment, peers exchange their supported algorithm sets and select the strongest mutually supported configuration.

```
ZAP Handshake Extension:
```

```
AlgorithmSet {
  signatures: [SIG_BLS12_381, SIG_MLDSA_65]
  hashes:     [HASH_SHA3_256, HASH_SHA256]
  kems:       [KEM_MLKEM_768]
  min_epoch:  current_epoch - 1
}
```

Peers that do not support the minimum required algorithm set for the current epoch are rejected. This prevents downgrade attacks where an adversary forces a connection to use a weaker primitive.

2.4 Hybrid Classical–Post-Quantum Signing

During the transition from classical to post-quantum signatures, Lux uses a hybrid approach: each block carries both a classical (BLS) and a post-quantum (ML-DSA or Ringtail) signature. This is the foundation of Quasar's triple-proof finality [1].

The hybrid signature is not a concatenation but a structured composite:

$$\sigma_{\text{hybrid}} = (\sigma_{\text{classical}}, \sigma_{\text{PQ}}, \text{binding}) \tag{1}$$

$$\text{binding} = H(\sigma_{\text{classical}} \| \sigma_{\text{PQ}} \| \text{epoch}) \tag{2}$$

The binding hash prevents an attacker from stripping one signature component without detection.

3 Migration Procedure

A primitive rotation follows a four-phase protocol:

1. **Proposal** (epoch e): A governance transaction on Q-Chain proposes the new algorithm, specifying the overlap window length.
2. **Key Distribution** (epochs $e+1$ to $e+k$): Validators generate and register new keys for the incoming algorithm. For threshold schemes, this requires a DKG round.
3. **Overlap** (epochs $e+k+1$ to $e+k+w$): Both old and new algorithms are accepted. Validators sign with both. Monitoring tracks adoption percentage.
4. **Cutover** (epoch $e+k+w+1$): The old algorithm moves to `DEPRECATED`. Only new-algorithm signatures are accepted for new blocks. Historical verification continues to accept old signatures.

Phase	Duration	Validator Action
Proposal	1 epoch	Vote on governance transaction
Key Distribution	k epochs	Generate + register new keys
Overlap	w epochs	Dual-sign all blocks
Cutover	1 epoch	Drop old signing path

Table 2: Migration phases. Typical values: $k = 10$, $w = 100$ epochs.

No chain halt occurs at any phase. Validators that fail to generate new keys by the cutover epoch are excluded from the active set until they comply.

4 Cross-Chain Considerations

Warp messages and Teleport transfers carry signatures from the source chain's current key schedule. The receiving chain must maintain a registry of valid key schedules per source chain to verify cross-chain messages produced during any epoch.

1: **function** VerifyWarpMessage(msg, sourceChain)
2: $epoch \leftarrow msg$.sourceEpoch
3: $schedule \leftarrow$ SourceKeySchedule($sourceChain, epoch$)
4: **return** VerifyAggregate(msg.signature, $schedule$.algorithm)
5: **end function**

This requires that receiving chains track source chain key schedule transitions. Q-Chain broadcasts key schedule updates as Warp messages to all registered chains.

5 Security Analysis

Theorem 1 (No Downgrade)**.** *An adversary controlling $< n/3$ validators cannot force the network to accept a block signed with a deprecated algorithm after the cutover epoch.*

Proof. Block acceptance requires $\geq 2n/3$ valid signatures under the current key schedule. Since honest validators ($> 2n/3$) reject deprecated-algorithm signatures after cutover, no block using only the old algorithm can achieve quorum. □

Theorem 2 (Historical Integrity)**.** *Deprecating an algorithm does not invalidate previously finalized blocks.*

Proof. Verification dispatches on the epoch embedded in the block header. Blocks produced during epoch e are verified against the key schedule active at e, regardless of the current schedule. The algorithm registry retains verification-only support for deprecated algorithms indefinitely. □

6 Conclusion

Cryptographic agility is not optional for systems expected to operate across decades. Lux's architecture treats primitive rotation as a first-class operational concern, with versioned key schedules, negotiated handshakes, and structured hybrid signatures providing a migration path that requires no downtime, no state export, and no consensus fork. The same mechanism that enabled the ECDSA-to-BLS transition now supports the classical-to-PQ transition via Quasar's triple-proof finality [1].

References

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

[2] Kelling, Z. (2023). *ZAP: A Binary Wire Protocol for Sub-Microsecond Serialization*. Lux Industries Research.

[3] NIST (2024). *Post-Quantum Cryptography Standardization: FIPS 203, 204, 205*. National Institute of Standards and Technology.

[4] Kelling, Z. (2024). *Post-Quantum Cryptographic Suite for Blockchain Infrastructure*. Lux Industries Research.

[5] Kelling, Z. (2024). *Ringtail: Lattice-Based Threshold Signatures for Validator Networks*. Lux Industries Research.

[6] Mosca, M. (2018). *Cybersecurity in an Era with Quantum Computers*. IEEE Security & Privacy, 16(5), 38–41.

[7] Barker, E. & Roginsky, A. (2020). *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. NIST SP 800-131A Rev. 2.