Quasar: Composing Threshold Post-Quantum Signatures for Blockchain Finality

Zach Kelling

Lux Industries

2026-05-18

Abstract

Quasar is the Lux Network's post-quantum (PQ) consensus suite. Per-round finality is asserted via a single wire-format certificate, the *QuasarCert*, that composes up to five independent cryptographic legs: a BLS12-381 aggregate signature (classical fast path, discrete logarithm), a Pulsar threshold ML-DSA-65 signature (Module-LWE), an optional Corona threshold signature (Module-LWE), an optional Magnetar threshold SLH-DSA signature (hash-based), and a Z-Chain Groth16 proof rolling up N per-validator ML-DSA-65 identity signatures into a 192-byte SNARK. Three cert profiles — **Pulsar** (BLS + Pulsar + Z-Chain Groth16; PQ floor), **Aurora** (adds Corona; intra-lattice diversity), and **Polaris** (adds Magnetar; cross-family) — select which legs are populated. Each profile is independently selectable per Lux chain.

This paper formalizes the QuasarCert wire format, the three cert profile compositions, and the cross-primitive composition theorems. The central security claim is conjunctive hardness diversification: forging a profile-$\mathcal{P}$ cert requires forging *every* populated leg, each backed by an independent hardness assumption. We document the production deployment topology on Lux primary network's Q-Chain finality and the gating from the current Pulsar profile to the Aurora and Polaris profiles as the underlying Corona and Magnetar primitives reach Tier A audit status. The Pulsar profile is currently production-deployed; Aurora and Polaris are operationally ready and gated on primitive maturity.

Scope. This is the umbrella paper for the Quasar suite. The per-primitive papers — LP-020 (Quasar consensus), LP-073 (Pulsar), Corona, Magnetar, and the Lean-mechanized soundness proof at `quasar-cert-soundness.tex` — own the per-leg security arguments. This paper owns the cross-primitive composition: the wire format that binds the legs into a single envelope, the verifier that enforces the conjunction, and the theorems that bound forgery by the minimum advantage across populated legs.

Contents

1 Introduction

1.1 The composition problem

Post-quantum (PQ) deployment for blockchain finality is an exercise in restraint. The temptation to "add a PQ signature" to an existing classical finality protocol produces certificates that are larger, slower, and only as strong as the weakest leg. The opposite temptation — to replace all classical primitives with PQ primitives in a single swap — discards a decade of operational experience with BLS12-381 aggregation and produces a chain whose verification cost rises by an order of magnitude before the migration is justified by an actual quantum threat.

The Lux Network resolves this tension by treating per-round finality as the *composition* of independent cryptographic legs, each backed by a distinct hardness assumption, each independently verifiable, and each scoped to a single concern. The composition object is the *QuasarCert*: a per-round wire-format certificate that carries up to five legs — a BLS12-381 aggregate, a Pulsar threshold ML-DSA-65 signature, a Corona threshold M-LWE signature, a Magnetar threshold SLH-DSA signature, and a Z-Chain Groth16 rollup of per-validator ML-DSA-65 identity signatures — combined under a strict "conjunctive" verifier: every populated leg must verify, and the set of populated legs must match the configured cert profile exactly.

Three cert profiles are defined: **Pulsar** (BLS + Pulsar; PQ floor), **Aurora** (BLS + Pulsar + Corona; intra-lattice diversity), and **Polaris** (BLS + Pulsar + Corona + Magnetar; cross-family). Each profile is independently selectable per Lux chain. The Pulsar profile is the current production posture on Lux primary network's Q-Chain finality; Aurora and Polaris are operationally ready constructions whose deployment is gated on independent audit of the non-Pulsar primitives (Section 10).

1.2 What this paper is, and what it is not

This paper is the *umbrella* specification for Quasar finality. It complements but does not duplicate the per-primitive papers:

- LP-020 [29] defines the Quasar consensus protocol (validator set, round structure, Horizon finality), the original three-lane wire format, and the soundness theorem for the BLS+Pulsar+Z-Chain composition.
- LP-073 / Pulsar [30, 20] defines the threshold ML-DSA-65 primitive on its own terms (DKG, Sign1/Sign2/Combine, resharing, key-era lifecycle).
- LP-073-Corona / Corona [10, 28] defines the threshold Module-LWE primitive (Lux production fork of Boschini et al. ePrint 2024/1113).
- Magnetar [33] defines the threshold SLH-DSA primitive (v0.1 reveal-and-aggregate, byte-equal to FIPS 205).
- The Lean-mechanized soundness proof [22] and the BLS-removal analysis [21] are the primary formal artifacts.

The umbrella paper's job is to specify the *wire-level composition* that binds these primitives together, prove the *cross-primitive composition theorems*, and document the *production deployment topology*. It deliberately does not re-derive primitive-level soundness; for those, the reader is referred to the source papers.

1.3 Decomplecting finality

Quasar's design follows a strict orthogonality discipline. Each primitive lives in its own repository (Section 7) and has its own verifier; the QuasarCert is a passive container whose verification rule is the conjunction of leg-local verifiers; the cert profile is a single configuration value (`FinalitySchemeID`) that selects which legs are populated; the consensus engine treats the entire QuasarCert as a black box — finality is reached when the cert verifies, and the engine does not look inside it.

This is the only way to build a PQ stack that survives the migration era. A primitive that fails (because of a cryptanalytic break, a side-channel leak, or a NIST standardization change) must be swappable without rebuilding the consensus engine, without changing the wire format on the other legs, and without coupling state to the deprecated primitive. The QuasarCert achieves this by making the populated-leg set a runtime predicate over the cert bytes, not a compile-time property of the consensus state machine.

1.4 Hardness diversification, not redundancy

The central security claim of QuasarCert is conjunctive *hardness diversification*: an adversary who forges a profile-N QuasarCert must defeat the signature scheme of *each* populated leg. The Pulsar profile demands a break of BLS (co-CDH on BLS12-381, broken in BQP by Shor [42]) *and* a break of threshold ML-DSA-65 (Module-LWE/MSIS [35, 34]). The Aurora profile demands additionally a break of Corona (M-LWE [39, 10]). The Polaris profile demands a break of Magnetar (SHA3 / hash-based one-wayness via SLH-DSA [36]).

This is *not* redundancy in the sense of running the same operation twice. The Pulsar and Corona legs diversify *within* the lattice family — distinct Module-LWE parameter sets and codebases, so a parameter- or implementation-localized break need not hit both — while the Magnetar leg diversifies *across* families: a break against the lattice family as a whole does not transfer to hash-based constructions. Family-disjoint hardness comes from Magnetar, not from pairing two Module-LWE legs. The combinatorial hardness reduction in Section 5 formalises this claim under standard reductions.

1.5 Roadmap

Section 2 fixes notation, primitive definitions, and the hardness assumptions invoked throughout. Section 3 specifies the QuasarCert wire format byte-for-byte against the Go canonical at `luxfi/consensus/protocol/quasar/types.go`. Section 4 defines the three cert profiles and their composition tables. Section 5 states and proves the cross-primitive composition theorems. Section 6 discusses how QuasarCert binds into the Lux consensus engine. Section 7 describes the implementation, repository layout, and pinned versions. Section 8 reports measured cert sizes, verify times, and storage projections. Section 9 states the threat model. Section 10 documents the production deployment topology and the gating to Aurora and Polaris. Section 11 concludes.

2 Preliminaries

2.1 Notation

Throughout, λ denotes a security parameter (set to 128 for NIST PQ Category 3). For a finite set S, $x \xleftarrow{\$} S$ denotes uniform sampling. A function $\epsilon(\lambda)$ is *negligible* if for every polynomial p, $\epsilon(\lambda) < 1/p(\lambda)$ for all sufficiently large λ; we write $\mathsf{negl}(\lambda)$. The notation H_X denotes a hash function modeled as a random oracle in domain-separated context X. The string ctx_X denotes the FIPS 204 §5 context byte string for purpose X, fixed in Table 1.

2.2 Validator set

A Lux chain at consensus epoch e has a validator set $\mathcal{V}_e = \{v_1, \ldots, v_n\}$ with stake distribution $\mathrm{stk} : \mathcal{V}_e \to \mathbb{N}$ and total stake $S_e = \sum_i \mathrm{stk}(v_i)$. Each validator v_i holds:

- a BLS12-381 keypair $(sk_i^{\mathrm{BLS}}, pk_i^{\mathrm{BLS}})$ with proof of possession;
- a Pulsar threshold-secret share sk_i^{Puls} over the joint Pulsar group public key pk^{Puls}, produced by the Pulsar DKG [20];

- on the Aurora and Polaris profiles, a Corona share sk_i^{Cor} over a joint pk^{Cor} [28];
- on the Polaris profile, a Magnetar share sk_i^{Mag} over a joint pk^{Mag} [33];
- a per-validator ML-DSA-65 keypair $(sk_i^{\mathrm{ML}}, pk_i^{\mathrm{ML}})$ used for the Z-Chain rollup leg.

The Byzantine threshold is $f < n/3$ in count and $f < S_e/3$ in stake. The quorum threshold $\tau = \lfloor 2n/3 \rfloor + 1$ in count (or $\lceil 2S_e/3 \rceil$ in stake) is the minimum signer set required to assemble a valid leg. The validator set is committed to a Merkle root $\mathsf{validator\_root}_e$ at epoch boundaries.

2.3 Round digest

Each consensus round produces a *round digest* μ that all legs sign or prove against. The round digest is defined as

$$\mu \;=\; H_{\mathsf{round}}\big(\mathsf{chain\_id} \,\|\, e \,\|\, r \,\|\, \mathsf{validator\_root}_e \,\|\, \mathsf{block\_hash}\big), \tag{1}$$

where $\mathsf{chain\_id}$ is the 32-byte unique chain identifier, e is the epoch, r is the round number, and $\mathsf{block\_hash}$ is the structural hash of the block being finalized. The digest is a 32-byte SHA3-256 / Keccak-256 output. This binding makes cross-chain and cross-epoch replay structurally impossible (Theorem 3.3).

2.4 BLS12-381 aggregate signatures

We use the proof-of-possession variant of BLS aggregation [8, 9, 7]: keys are generated as $sk \xleftarrow{\$} \mathbb{Z}_r$, $pk = sk \cdot G_2 \in \mathbb{G}_2$, each public key is published with a proof of possession (a BLS signature on the public key under itself, in a domain-separated context); signatures are $\sigma = sk \cdot H_{\mathbb{G}_1}(\mu) \in \mathbb{G}_1$; aggregation is point addition, $\sigma_{\mathrm{agg}} = \sum_{i \in Q} \sigma_i$; verification is $e(G_2, \sigma_{\mathrm{agg}}) = e(apk, H_{\mathbb{G}_1}(\mu))$ with aggregated public key $apk = \sum_{i \in Q} pk_i$. Aggregated signatures are 48 bytes (compressed $\mathbb{G}_1$); verification cost is $O(1)$ in $|Q|$ and dominated by two pairings.

Security assumption. BLS-PoP is EUF-CMA under the co-CDH assumption on $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ in the random oracle model [9, 5, 7]. The co-CDH assumption is broken in BQP by Shor's algorithm on the discrete logarithm in $\mathbb{F}_{q^{12}}$, so BLS contributes no PQ security: it is the *classical* fast-path leg.

2.5 Pulsar: threshold ML-DSA-65

Pulsar [20] is the Lux production fork of the threshold ML-DSA construction of Celi et al. [13], integrated with the Lux Secret Sharing (LSS) lifecycle [40]. The signing output is byte-identical to a single-party FIPS 204 [35] ML-DSA-65 signature on the group public key pk^{Puls}, so verification uses the unmodified NIST verifier. NIST MPTC [37] classifies this as Class N1 (single-party-compatible threshold signing); the Lux deployment adds Class N4 (public-key preservation across resharing).

Security assumption. Pulsar's unforgeability reduces to the EUF-CMA security of FIPS 204 ML-DSA-65, which in turn reduces to MLWE and MSIS at NIST Category 3 (∼128-bit quantum security). The v0.1 Pulsar deployment has an aggregator-as-TCB caveat: a malicious aggregator can refuse to produce a signature (a liveness fault) but cannot produce a forgery, because the aggregator never holds enough material to bypass the threshold; this is documented in the Pulsar deployment runbook [20].

2.6 Corona: threshold M-LWE

Corona [28, 10] is a two-round lattice threshold signature over the polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N+1)$ with the Lux production parameter set $N = 256$, $q = 2^{48}+2561$ (NTT-friendly),

module dimension $\ell = 8$. It follows a Schnorr-with-aborts structure: each party samples a nonce $\mathbf{y}_i$, broadcasts a commitment $\mathbf{w}_i = \mathbf{A}\mathbf{y}_i$, the challenge is $c = H_{\mathsf{cor}}(\mu \,\|\, \sum_i \mathbf{w}_i)$, and the response is $\mathbf{z}_i = \mathbf{y}_i + c \cdot sk_i^{\mathrm{Cor}}$ (with rejection sampling). Combination is Lagrange interpolation over the ring; verification checks $\|\mathbf{z}\| \le B$ and the Fiat–Shamir relation.

Security assumption. Corona is SUF-CMA under the Module-LWE and Module-SIS assumptions in the random oracle model [10]. Corona and ML-DSA both reduce to Module-LWE; they differ in parameter set and codebase (Corona at the Ringtail-derived parameters, ML-DSA at the FIPS-204 tuning), not in hardness family. Pairing them gives parameter- and implementation-level diversity at production sizes [1, 2] — a break specific to one parameter set need not transfer to the other — but not family-disjoint hardness; that is supplied by the hash-based Magnetar leg.

2.7 Magnetar: threshold SLH-DSA

Magnetar [33] is a threshold construction of FIPS 205 [36] SLH-DSA via *reveal-and-aggregate* over the SLH-DSA scheme seed. The DKG protocol shares the joint master seed S via byte-wise Shamir secret sharing over GF(257) [41], with t-of-n reconstruction; signing reconstructs S inside an aggregator's memory, calls `slhdsa.SignDeterministic` once on the reconstructed seed, and immediately zeroizes. The signature is byte-identical to a single-party FIPS 205 SLH-DSA signature on the joint group public key.

Security assumption. Magnetar's unforgeability reduces to SLH-DSA's EUF-CMA security, which in turn reduces to the second-preimage and target-collision resistance of SHA-3 / SHAKE-192 (or SHAKE-256, depending on parameter set). The v0.1 reveal-and-aggregate construction has an aggregator-as-TCB caveat identical in spirit to Pulsar v0.1: the aggregator briefly holds the reconstructed master seed and must zeroize. The Tier B status of Magnetar reflects that this caveat has not yet been audited end-to-end; production deployment of the Polaris profile is gated on Tier A audit of Magnetar [33].

2.8 Z-Chain Groth16 rollup of ML-DSA-65

The Z-Chain VM [23] runs a Groth16 [19] prover for the relation

$$\mathcal{R}_{\mathrm{ZK}} \;=\; \Big\{ (x;\, w) \;:\; x = (\mu, h_x),\; w = \big(\{pk_i^{\mathrm{ML}}\}_i, \{\sigma_i^{\mathrm{ML}}\}_i\big) \Big\}, \tag{2}$$

with predicate

$$\Phi(x, w) \;\iff\; h_x = H_{\mathsf{Poseidon}}\big(\{pk_i^{\mathrm{ML}}\}_i \,\|\, \mu\big) \;\wedge\; \sum_i \big[\!\big[\, \mathsf{MLDSA.Verify}(pk_i^{\mathrm{ML}}, \mu, \sigma_i^{\mathrm{ML}}) = 1 \,\big]\!\big] \;\ge\; \tau,$$

where $\tau = \lfloor 2n/3 \rfloor + 1$ and ML-DSA verification is the FIPS 204 algorithm. The Groth16 proof π_{ZK} is $(A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$, totalling $48 + 96 + 48 = 192$ bytes compressed.

Security assumption. Groth16 is knowledge-sound in the algebraic group model on BLS12-381 [19]; the verifying key vk_Z is the output of a powers-of-tau ceremony plus a circuit-specific phase 2 with ≥ 1024 contributors [11]. Knowledge soundness requires the BLS12-381 discrete log assumption and is therefore *not* post-quantum: a quantum adversary running Shor on $\mathbb{F}_{q^{12}}$ can extract trapdoors and forge π_{ZK}. However, π_{ZK} commits to the public-input hash h_x which in turn commits to the validator set and round digest; forging π_{ZK} does not produce forged Pulsar / Corona / Magnetar signatures, so the conjunctive composition still requires breaking those PQ legs to forge the cert as a whole (Theorem 5.1).

2.9 Hash functions and domain separation

All hash invocations use domain-separated contexts per FIPS 204 §5 and SP 800-185 (cSHAKE / KMAC). Table 1 lists the context strings used by the QuasarCert legs.

Context string	Hash	Used by
`lux-quasar-cert-v1`	SHAKE-256	Pulsar leg of QuasarCert
`lux-corona-cert-v1`	SHAKE-256	Corona leg of QuasarCert
`lux-magnetar-cert-v1`	SHAKE-256	Magnetar leg of QuasarCert
`lux-quasar-zchain-witness-v1`	SHAKE-256	per-validator ML-DSA into Groth16
`lux-quasar-bls-domain-v1`	SHA-256	BLS hash-to-curve $H_{\mathbb{G}_1}$
`lux-quasar-round-digest-v1`	SHA3-256	round-digest μ construction
`lux-quasar-validator-root-v1`	SHA3-256	validator-set Merkle root

Table 1: Domain-separation contexts used by QuasarCert legs. Mismatches across legs prevent cross-leg signature transfer (e.g., a BLS signature cannot be replayed as a Pulsar input even if the same byte string is presented).

3 QuasarCert: Wire Format

3.1 The QuasarCert structure

The QuasarCert is the per-round wire-format certificate that carries the legs required by the chain's configured profile. The canonical Go definition lives at `luxfi/consensus/protocol/quasar/types.go` in the Lux consensus repository [24]; it is mirrored into the umbrella spec at `luxfi/quasar/SPEC.md` [27].

Definition 3.1 (QuasarCert wire structure)**.** *A QuasarCert is the eight-field record*

$$\mathcal{C} \;=\; \big(\sigma^{\mathrm{BLS}},\ \sigma^{\mathrm{Puls}},\ \sigma^{\mathrm{Cor}},\ \sigma^{\mathrm{Mag}},\ \pi^{\mathrm{ZK}},\ e,\ T_{\mathrm{fin}},\ n_{\mathrm{val}}\big)$$

where the cryptographic legs are:

- $\sigma^{\mathrm{BLS}} \in \{0,1\}^{384} \cup \{\varepsilon\}$ *is a 48-byte BLS12-381 aggregate (compressed* $\mathbb{G}_1$*), or the empty byte string* ε *when absent.*
- $\sigma^{\mathrm{Puls}} \in \{0,1\}^{8\cdot 3309} \cup \{\varepsilon\}$ *is a 3309-byte FIPS 204 ML-DSA-65 signature (byte-equal to the single-party output of* $\mathsf{MLDSA.Sign}(pk^{\mathrm{Puls}}, \mu, \mathsf{ctx})$ *with* $\mathsf{ctx} =$ `lux-quasar-cert-v1`*), or* ε*.*
- $\sigma^{\mathrm{Cor}} \in \mathcal{B}^{\sim 33\ KB} \cup \{\varepsilon\}$ *is a Corona threshold signature in native binary form (length-prefixed concatenation of polynomial-ring marshal bytes; see Section 3.2). Or* ε*.*
- $\sigma^{\mathrm{Mag}} \in \{0,1\}^{8\cdot 16224} \cup \{\varepsilon\}$ *is a 16,224-byte FIPS 205 SLH-DSA-SHAKE-192f signature (byte-equal to the single-party output of* $\mathsf{SLHDSA.SignDeterministic}$ *on the reconstructed master seed), or* ε*.*
- $\pi^{\mathrm{ZK}} \in \{0,1\}^{192\cdot 8} \cup \{\varepsilon\}$ *is a Groth16 proof (compressed* $\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$*,* $48 + 96 + 48 = 192$ *bytes), or* ε*.*

The metadata fields are:

- $e \in \mathbb{N}$ *is the validator-set epoch.*
- T_{fin} *is the wall-clock finality timestamp (advisory).*
- $n_{\mathrm{val}} \in \mathbb{N}$ *is the validator-set size at epoch* e*.*

The Go canonical (verbatim from `luxfi/consensus/protocol/quasar/types.go`):

```
type QuasarCert struct {
    BLS         []byte    // BLS12-381 aggregate, 48 bytes or empty
    Pulsar      []byte    // FIPS 204 ML-DSA-65 byte-equal, 3309 B or empty
```

```
    Corona      []byte    // Corona M-LWE threshold, ~33 KB or empty
    Magnetar    []byte    // FIPS 205 SLH-DSA byte-equal, ~16 KB or empty
    MLDSAProof  []byte    // Groth16 over BLS12-381, 192 B or empty
    Epoch       uint64
    Finality    time.Time
    Validators  int
}
```

3.2 Encoding rules

The QuasarCert is encoded via the native binary encoder at `luxfi/consensus/protocol/quasar/corona_gob.g` (despite the file name, this is a custom length-prefixed encoder, not Go's `encoding/gob`; the file name is preserved for caller compatibility). The encoder uses three length-prefixed fields with big-endian four-byte length headers:

```
[4-byte BE C_len ][C bytes ]
[4-byte BE Z_len ][Z bytes ]
[4-byte BE Delta_len ][Delta bytes]
```

This native encoder is $\sim$1.01$\times$ the size of Go gob (measured in `cert_size_compare_test.go`) but is deterministic, language-agnostic, and stable across Go versions. The native encoding is the canonical wire byte string for cross-language verifiers (Rust, C++, Python, WASM).

For the Corona leg specifically, C is a single `ring.Poly` (a polynomial in $R_q = \mathbb{Z}_q[X]/(X^N + 1)$), Z is a `Vector[ring.Poly]` of length 8 (the module rank), and *Delta* is another `Vector[ring.Poly]` of length 8. With $N = 256$ coefficients $\times$ 8 bytes per coefficient $\times$ 17 polynomials = 34,816 bytes raw; the measured native binary encoding is 33,052 bytes (the difference is from the modular coefficient packing).

For the BLS leg, the 48-byte compressed $\mathbb{G}_1$ point uses the IETF BLS draft [8] serialization with the $\mathsf{POP\_TAG}$ infinity / compression flag in the leading three bits.

For the Pulsar leg, the FIPS 204 ML-DSA-65 signature is the byte-equal output of the single-party verifier; the signing context string is `lux-quasar-cert-v1`.

For the Magnetar leg, the FIPS 205 SLH-DSA signature is the byte-equal output of the single-party verifier; the signing context string is `lux-magnetar-cert-v1`.

For the π^{ZK} leg, the Groth16 proof uses compressed serialization of $(A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ per the BLS12-381 IETF draft.

3.3 Verification predicate

Definition 3.2 (QuasarCert verification)**.** *Given a profile $\mathcal{P}$ (`Pulsar`, `Aurora`, or `Polaris`) with populated-leg specification $\mathcal{L}(\mathcal{P}) \subseteq \{\mathrm{BLS}, \mathrm{Puls}, \mathrm{Cor}, \mathrm{Mag}, \mathrm{ZK}\}$, a block B, a validator set $\mathcal{V}_e$, and a candidate cert $\mathcal{C}$, $\mathsf{VerifyQuasarCert}(B, \mathcal{V}_e, \mathcal{C}, \mathcal{P})$ returns 1 iff:*

1. ***Profile match**: $\{\ell : \mathcal{C}.\ell \neq \varepsilon\} = \mathcal{L}(\mathcal{P})$. A cert with extra populated legs is malformed; a cert with missing populated legs is malformed.*
2. ***Validator count consistency**: $\mathcal{C}.n_{\mathrm{val}} = |\mathcal{V}_e|$.*
3. ***Round digest**: $\mu = H_{\mathsf{round}}(\mathsf{chain\_id} \,\|\, e \,\|\, r \,\|\, \mathsf{validator\_root}_e \,\|\, \mathsf{block\_hash})$ is recomputed locally from B and $\mathcal{V}_e$.*
4. ***BLS leg**: if $\mathrm{BLS} \in \mathcal{L}(\mathcal{P})$, then $e(G_2, \sigma^{\mathrm{BLS}}) = e(apk_{\mathrm{BLS}}, H_{\mathbb{G}_1}(\mu))$ where apk_{BLS} is the aggregated public key of validators in the quorum.*
5. ***Pulsar leg**: if $\mathrm{Puls} \in \mathcal{L}(\mathcal{P})$, then $\mathsf{MLDSA.Verify}(pk^{\mathrm{Puls}}, \mu, \mathsf{ctx}, \sigma^{\mathrm{Puls}}) = 1$ with $\mathsf{ctx} =$ `lux-quasar-cert-v1`.*
6. ***Corona leg**: if $\mathrm{Cor} \in \mathcal{L}(\mathcal{P})$, then $\mathsf{Corona.Verify}(pk^{\mathrm{Cor}}, \mu, \sigma^{\mathrm{Cor}}) = 1$.*
7. ***Magnetar leg**: if $\mathrm{Mag} \in \mathcal{L}(\mathcal{P})$, then $\mathsf{SLHDSA.Verify}(pk^{\mathrm{Mag}}, \mu, \mathsf{ctx}, \sigma^{\mathrm{Mag}}) = 1$ with $\mathsf{ctx} =$ `lux-magnetar-cert-v1`.*

8. ***Z-Chain leg:*** *if* $\mathrm{ZK} \in \mathcal{L}(\mathcal{P})$, *then* $\mathsf{Groth16.Verify}(vk_Z, h_x, \pi^{\mathrm{ZK}}) = 1$ *with* $h_x = H_{\mathsf{Poseidon}}(\{pk_i^{\mathrm{ML}}\} \,\|\, \mu)$.

The verification predicate is the strict conjunction of all populated legs. If even one populated leg fails, the entire cert is rejected. Verification is stateless: each call recomputes the round digest and validator roots from the supplied block and validator set, so there is no replay cache to corrupt.

3.4 Replay protection via round-digest binding

The round digest μ binds the cert to the chain ID, epoch, round, validator-set root, and block hash. Any change in any of these inputs produces a different μ, and by collision resistance of SHA3-256 the probability of matching a different cert's μ is bounded by 2^{-256}.

Theorem 3.3 (Cross-chain / cross-epoch replay impossibility)**.** *Let $\mathcal{C}$ be a QuasarCert that verifies for block B on chain c at epoch e, round r. For any block $B' \neq B$ on chain c', epoch e', round r' such that $(c, e, r, \mathsf{block\_hash}) \neq (c', e', r', \mathsf{block\_hash}')$, $\mathsf{VerifyQuasarCert}(B', \mathcal{V}_{e'}, \mathcal{C}, \mathcal{P})$ returns 0 except with negligible probability.*

Proof sketch. Suppose the verifier accepts $\mathcal{C}$ against B'. The verifier recomputes μ' from B' and $\mathcal{V}_{e'}$; this is deterministic and depends only on the supplied block and validator set. If $(c, e, r, \mathsf{block\_hash}) \neq (c', e', r', \mathsf{block\_hash}')$, then $\mu' \neq \mu$ except with collision probability $\leq 2^{-256}$ over SHA3-256. Every populated leg of $\mathcal{C}$ was produced for μ, not μ':

- BLS: EUF-CMA of BLS-PoP ensures $e(G_2, \sigma^{\mathrm{BLS}}) \neq e(apk, H_{\mathbb{G}_1}(\mu'))$ because the signed message has changed.
- Pulsar: ML-DSA-65 EUF-CMA ensures rejection.
- Corona: SUF-CMA of Corona ensures rejection.
- Magnetar: SLH-DSA EUF-CMA ensures rejection.
- Z-Chain: Groth16 knowledge soundness ensures the proof bound to $h_x = H_{\mathsf{Poseidon}}(\{pk_i^{\mathrm{ML}}\} \,\|\, \mu)$ does not also verify against $h_{x'} = H_{\mathsf{Poseidon}}(\{pk_i^{\mathrm{ML}}\} \,\|\, \mu')$, assuming Poseidon collision resistance.

Each populated leg independently rejects with overwhelming probability; by union bound, the cert is rejected except with probability $\leq 2^{-256} + |\mathcal{L}(\mathcal{P})| \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$. □

Corollary 3.4 (Profile non-substitutability)**.** *A cert valid under profile $\mathcal{P}$ cannot be presented as a cert under profile $\mathcal{P}' \neq \mathcal{P}$, because the profile-match clause (Definition 3.2 clause 1) fails: either the populated-leg set is too big (downward substitution) or too small (upward substitution).*

This structural binding is critical because it means the consensus engine does not need a separate replay cache or sequence-number tracker: the cert verifier is stateless and the binding is encoded in the cert bytes themselves.

4 Cert Profiles

A *cert profile* fixes which legs of the QuasarCert are populated and verified. Profiles are runtime selectors: a single Lux chain is configured with one profile via the `FinalitySchemeID` field on its consensus config (see `luxfi/consensus/config/pq_mode.go`). The profile is part of the chain's bootstrap genesis and may be rotated only via a coordinated multi-window upgrade (Section 4.4).

4.1 Profile registry

Three profiles are defined.

`Pulsar`**: PQ floor**

- **Composition**: BLS ‖ Puls ‖ ZK.
- **Primitives**: BLS12-381 aggregate (classical), threshold ML-DSA-65 (M-LWE), Z-Chain Groth16 rollup.
- `FinalitySchemeID`: `BLSPlusCorona` (legacy enum name; the "Corona" label here historically referred to the threshold-protocol family slot, now instantiated with Pulsar's M-LWE kernel).
- **Cert size**: $48 + 3309 + 192 + \mathsf{meta} \approx 3549$ bytes per cert at $n = 21$ validators.
- **Use case**: Minimum PQ posture — every cert carries a NIST-standardized PQ threshold signature *and* a classical fast-path leg. This is the current production deployment on Lux primary network's Q-Chain.

`Aurora`**: intra-lattice diversity**

- **Composition**: BLS ‖ Puls ‖ Cor ‖ ZK.
- **Primitives**: BLS, threshold ML-DSA-65 (M-LWE), threshold Corona (M-LWE), Z-Chain Groth16 rollup.
- `FinalitySchemeID`: `TripleQuantum` with the Corona leg enabled.
- **Cert size**: $48 + 3309 + 33052 + 192 + \mathsf{meta} \approx 37$ KB per cert.
- **Use case**: Defense in depth across two Module-LWE parameter sets and codebases. A cryptanalytic break specific to one parameter set or implementation (Pulsar's or Corona's, but not Module-LWE as a whole) still leaves one valid PQ leg on every cert. This is the migration target for chains that desire intra-lattice parameter and implementation diversification — though not family-disjoint hardness — without adopting Magnetar.

`Polaris`**: cross-family maximum assurance**

- **Composition**: BLS ‖ Puls ‖ Cor ‖ Mag ‖ ZK.
- **Primitives**: BLS, threshold ML-DSA-65, threshold Corona, threshold SLH-DSA (hash-based), Z-Chain Groth16 rollup.
- `FinalitySchemeID`: `TripleQuantum` with Corona AND Magnetar enabled.
- **Cert size**: $48 + 3309 + 33052 + 16224 + 192 + \mathsf{meta} \approx 53$ KB per cert.
- **Use case**: Cross-family defense in depth. Lattice (Pulsar / Corona) plus hash-based (Magnetar). A cryptanalytic break against the entire lattice family does not break the cert.
- **Status**: Reserved. Magnetar is Tier B (v0.1 reveal-and-aggregate); production deployment of the Polaris profile is gated on Tier A audit of Magnetar's reveal-and-aggregate construction (Section 10).

4.2 Composition table

Table 2 summarizes the legs per profile and the hardness assumptions invoked.

4.3 Profile invariants

All profiles must satisfy:

1. **BLS classical leg is always populated.** The fast-path requirement: a classical aggregate verifier on commodity CPU gives sub-millisecond verification, which dominates the hot-path block-propagation budget.
2. **At least one PQ leg is always populated.** The PQ-finality requirement: a chain whose finality verifier could be defeated by a quantum adversary alone is not PQ-finalizing.

Profile	BLS	Puls	Cor	Mag	ZK	Hardness
`Pulsar`	•	•	—	—	•	co-CDH ∧ M-LWE/MSIS ∧ KEA
`Aurora`	•	•	•	—	•	co-CDH ∧ M-LWE ∧ KEA
`Polaris`	•	•	•	•	•	co-CDH ∧ M-LWE ∧ SHA3-OW ∧ KEA

Table 2: Cert profile composition. "Hardness" is the conjunctive hardness *family* required to forge a cert under the profile (Theorem 5.1). Aurora and Polaris populate two Module-LWE legs (Pulsar at the FIPS-204 parameters, Corona at the Ringtail-derived parameters): distinct parameter sets and codebases within the same M-LWE family, giving parameter/implementation diversity rather than a second hardness family. "KEA" is the knowledge-of-exponent / algebraic group assumption that backs Groth16; "SHA3-OW" is SHA-3 one-wayness / SLH-DSA EUF-CMA.

3. **The set of populated cert legs matches the profile selector exactly.** Extra populated legs or missing populated legs is malformed (Definition 3.2 clause 1). This invariant is the load-bearing structural binding that prevents profile substitution attacks.
4. **Domain separation between legs is enforced.** The context strings in Table 1 ensure that a Pulsar signature on μ cannot be replayed as a Corona signature on μ (even though they sign the same round digest, they sign with different domain-separation contexts).
5. **Validator set is shared across legs at the same epoch.** Each leg signs under the same set's primitive-specific public keys; the BLS PoP, Pulsar group key, Corona group key, Magnetar group key, and per-validator ML-DSA keys are all rooted in the same $\texttt{validator\_root}_e$ Merkle commitment.

4.4 Profile rotation

Profile rotation is a coordinated multi-window event because the cert wire format and the populated-leg set change. The rotation protocol is:

1. **Pre-window**. Both old and new profiles' primitives run in parallel: validators produce certs under the old profile only, but each validator already holds shares for the new profile's legs (so the DKG ceremonies for added legs have completed before the window opens).
2. **Rotation window** of W blocks ($W \approx 1000$ on Lux production). During the window, blocks may carry either profile's cert. Cert-acceptance logic accepts either profile.
3. **Post-window**. Only the new profile's cert is accepted. Blocks proposed with the old profile are rejected.

This is documented in the deployment runbook at `luxfi/pulsar/DEPLOYMENT-RUNBOOK.md` for the Pulsar-leg-specific rotation, and generalises to all profile rotations in the umbrella spec's `PROFILES.md`.

4.5 Why three profiles and not one

The natural design temptation is to fix a single profile (probably Polaris, maximum assurance) and force every Lux chain to adopt it. We reject this for two reasons.

First, **cert size and verify cost are not free**. The Polaris profile is roughly 15× the size of the Pulsar profile and correspondingly more expensive to verify. For chains where the threat model does not warrant cross-family diversification, the extra cost is wasted.

Second, **profile portability is a deployment guarantee**. Different Lux chains have different threat models: a public L1 validator network might choose Aurora; a regulated-finance L1 might require Polaris; a high-throughput application chain might stay on Pulsar. Profile is per-chain, not global, so each chain can adopt the appropriate posture without forcing a network-wide upgrade.

The profile selector is the single point of variation; everything else — primitives, wire format, verification logic — is shared across all chains. This is the Hickey-simple separation of concerns: one cert wire format, one verifier, one profile selector, three profile compositions.

5 Composition Theorems

This section states and proves the central security claim of QuasarCert: forging a profile-$\mathcal{P}$ cert is at least as hard as the hardest leg in $\mathcal{L}(\mathcal{P})$.

5.1 Adversary model

We consider a non-uniform polynomial-time adversary $\mathcal{A}$ against the conjunctive cert verifier. Let $\mathsf{Corrupt}$ be the subset of $\mathcal{V}_e$ that $\mathcal{A}$ controls; let $f = |\mathsf{Corrupt}|/|\mathcal{V}_e|$. We require $f < 1/3$ (Byzantine threshold), so $|\mathsf{Honest}| = n - |\mathsf{Corrupt}| \geq 2n/3$ and $\tau = \lfloor 2n/3 \rfloor + 1 \leq |\mathsf{Honest}|$.

$\mathcal{A}$ is given:

- All primitive public keys: pk_i^{BLS}, pk^{Puls}, pk^{Cor}, pk^{Mag}, $\{pk_i^{\mathrm{ML}}\}_i$.
- Signing oracles for the corrupt set: for each $v_i \in \mathsf{Corrupt}$, $\mathcal{A}$ may invoke $\mathsf{BLS.Sign}(sk_i^{\mathrm{BLS}}, \cdot)$, the Pulsar threshold signing protocol with sk_i^{Puls}, etc.
- The Z-Chain verifying key vk_Z and the Groth16 trusted setup transcript (i.e., $\mathcal{A}$ knows vk_Z in full).
- Random oracle access to all hash functions, programmable in the proof.

$\mathcal{A}$ wins the soundness game if it outputs $(B^\star, \mathcal{C}^\star)$ such that $\mathsf{VerifyQuasarCert}(B^\star, \mathcal{V}_{e^\star}, \mathcal{C}^\star, \mathcal{P}) = 1$ and the block $B^\star$ was not endorsed by a stake-weighted honest quorum on round digest $\mu^\star$ derived from $B^\star$ and $\mathcal{V}_{e^\star}$.

We consider two adversary variants:

$\mathcal{A}_C$ — **classical PPT.** Standard probabilistic polynomial-time non-uniform Turing machine. Access to classical random oracles.

$\mathcal{A}_Q$ — **quantum PPT.** Polynomial-time on a quantum Turing machine; can run Shor's algorithm on classical discrete logarithms; has access to QROM [3]; subject to Grover speedup against unstructured search.

5.2 Main composition theorem

Theorem 5.1 (QuasarCert conjunctive soundness)**.** *Let $\mathcal{P}$ be a profile with leg set $\mathcal{L}(\mathcal{P})$. Under the hardness assumptions of Section 2, for any non-uniform PPT adversary $\mathcal{A}$ in either variant $\mathcal{A}_C$ or $\mathcal{A}_Q$,*

$$\mathsf{Adv}^{\mathrm{forge}}_{\mathcal{A}}(\mathcal{P}) \;\leq\; \min_{\ell \in \mathcal{L}(\mathcal{P})} \epsilon_\ell(\mathcal{A}),$$

where $\epsilon_\ell(\mathcal{A})$ is $\mathcal{A}$'s advantage against the standalone unforgeability game of the primitive in leg ℓ.

Proof. Fix a profile $\mathcal{P}$ and an adversary $\mathcal{A}$ that produces a winning $(B^\star, \mathcal{C}^\star)$. By Definition 3.2, every leg $\ell \in \mathcal{L}(\mathcal{P})$ of $\mathcal{C}^\star$ verifies under the primitive-specific verifier on round digest $\mu^\star$.

We exhibit, for each $\ell \in \mathcal{L}(\mathcal{P})$, a reduction $\mathcal{B}_\ell$ that uses $\mathcal{A}$ to break the standalone unforgeability game of the primitive at leg ℓ with advantage $\geq \mathsf{Adv}^{\mathrm{forge}}_{\mathcal{A}}(\mathcal{P})$.

BLS leg ($\ell = \mathrm{BLS}$). $\mathcal{B}_{\mathrm{BLS}}$ plays $\mathcal{A}$ in the QuasarCert game while embedding the challenge $pk^\star$ from the BLS EUF-CMA challenger into one slot of $\{pk_i^{\mathrm{BLS}}\}$. $\mathcal{B}_{\mathrm{BLS}}$ simulates all signing oracles for the corrupt set using $\mathcal{B}$'s own access to the BLS signing oracle for the embedded key. When $\mathcal{A}$ outputs $\mathcal{C}^\star$, $\mathcal{B}$ extracts $\sigma^{\mathrm{BLS}}_{\mathrm{agg}}$, computes $\sigma^\star$ by subtracting the contributions of non-target keys (using the BLS aggregate-verification structure), and returns $\sigma^\star$ on $\mu^\star$ as the BLS forgery. By the EUF-CMA security of BLS-PoP [9], $\mathcal{A}$'s forgery succeeds with probability $\leq \epsilon^{\mathrm{co-CDH}}_{\mathrm{BLS}}$.

Pulsar leg ($\ell = \text{Puls}$). $\mathcal{B}_{\text{Puls}}$ embeds the ML-DSA-65 challenge public key $pk^\star$ as the joint Pulsar group public key pk^{Puls}. $\mathcal{B}$ simulates Pulsar threshold-signing rounds for honest parties by sampling secret shares for them and invoking the FIPS 204 ML-DSA-65 challenger's signing oracle on $pk^\star$ for the joint signing message; the byte-equivalence property of Pulsar (any threshold output is byte-identical to the single-party FIPS 204 output, by construction) ensures the simulation is perfect. When $\mathcal{A}$ outputs $\mathcal{C}^\star$, $\mathcal{B}$ extracts σ^{Puls} and returns it on $\mu^\star$ as the ML-DSA-65 forgery. By EUF-CMA of FIPS 204 ML-DSA-65 under MLWE/MSIS [35, 14], the forgery succeeds with probability $\leq \epsilon_{\text{Puls}}^{\text{MLWE/MSIS}}$.

Corona leg ($\ell = \text{Cor}$). Identical structure to the Pulsar reduction, but with Corona's M-LWE-based two-round threshold [10, 28]. $\mathcal{B}_{\text{Cor}}$ embeds the Corona challenge into pk^{Cor} and simulates Round-1 commitments and Round-2 responses using the rejection-sampling oracle. By SUF-CMA of Corona under M-LWE/MSIS, the forgery probability is $\leq \epsilon_{\text{Cor}}^{\text{MLWE}}$.

Magnetar leg ($\ell = \text{Mag}$). $\mathcal{B}_{\text{Mag}}$ embeds the SLH-DSA challenge public key as pk^{Mag}. Because Magnetar v0.1 is reveal-and-aggregate (the joint master seed is reconstructed in the aggregator's memory, then a single FIPS 205 `SLHDSA.SignDeterministic` call is made), the simulation reduces to invoking the FIPS 205 challenger's signing oracle on $pk^\star$. By EUF-CMA of SLH-DSA under SHA-3 / SHAKE preimage and target-collision resistance [36, 4], the forgery probability is $\leq \epsilon_{\text{Mag}}^{\text{SHA3-OW}}$.

Z-Chain leg ($\ell = \text{ZK}$). $\mathcal{B}_{\text{ZK}}$ embeds the BLS12-381 discrete-log challenge into vk_Z via the standard Groth16 trapdoor reduction [19]. Because π^{ZK} is knowledge-sound in the AGM, an extractor can extract the witness (the per-validator ML-DSA signatures) from $\mathcal{A}$'s proof; the witness contains $\geq \tau$ ML-DSA-65 forgeries (by Byzantine-quorum honesty) which can be returned as an ML-DSA challenger forgery. The forgery probability is bounded by $\epsilon_{\text{ZK}}^{\text{KEA}} + \epsilon_{\text{MLDSA}}$.

Conjunction. Because the verification predicate is the conjunction over $\ell \in \mathcal{L}(\mathcal{P})$, $\mathcal{A}$'s winning cert $\mathcal{C}^\star$ contains a valid forgery at every populated leg. Each reduction $\mathcal{B}_\ell$ uses $\mathcal{C}^\star$ to win the standalone ℓ-game with the same probability as $\mathcal{A}$. Therefore

$$\mathsf{Adv}_{\mathcal{A}}^{\text{forge}}(\mathcal{P}) \;\leq\; \min_{\ell \in \mathcal{L}(\mathcal{P})} \epsilon_\ell(\mathcal{A}),$$

because the min is over a set of upper bounds. □

5.3 Hardness diversification

Corollary 5.2 (Hardness diversification). *For any profile $\mathcal{P}$ with* $\text{Puls} \in \mathcal{L}(\mathcal{P})$,

$$\mathsf{Adv}_{\mathcal{A}_Q}^{\text{forge}}(\mathcal{P}) \;\leq\; \epsilon_{\text{Puls}}^{\text{MLWE/MSIS}}(\mathcal{A}_Q).$$

Even if $\mathcal{A}_Q$ runs Shor and breaks the BLS and Groth16 legs in polynomial time, forging the cert still requires forging the Pulsar leg.

Proof. Direct from Theorem 5.1: the upper bound is the minimum over populated legs. The classical legs (BLS, Z-Chain) have $\epsilon = O(1)$ under $\mathcal{A}_Q$ (i.e., trivially breakable); the PQ legs have $\epsilon = \mathsf{negl}(\lambda)$. The minimum is the PQ leg, in particular the Pulsar leg (since $\text{Puls} \in \mathcal{L}(\mathcal{P})$ on every profile). □

Corollary 5.3 (Cross-family diversification under Polaris). *For $\mathcal{P} =$ `Polaris`,*

$$\mathsf{Adv}_{\mathcal{A}}^{\text{forge}}(\texttt{Polaris}) \;\leq\; \min\bigl(\epsilon_{\text{Puls}}^{\text{MLWE}}, \epsilon_{\text{Cor}}^{\text{MLWE}}, \epsilon_{\text{Mag}}^{\text{SHA3-OW}}\bigr).$$

A cryptanalytic break against the entire algebraic-lattice family (Module-LWE, which both Pulsar and Corona rest on) still leaves the Magnetar / SHA-3 leg standing.

Proof. Direct from Theorem 5.1. The three populated PQ legs invoke two independent hardness families: Module-LWE [35, 14, 39, 34, 10] — carried by both Pulsar (FIPS-204 parameters) and Corona (Ringtail-derived parameters), distinct parameter sets and codebases within the same family — and SHA-3 preimage / target-collision [36, 4]. A parameter- or implementation-localized break of one Module-LWE leg need not transfer to the other, but the family-disjoint guarantee is supplied by Magnetar: structural attacks against lattice constructions have not transferred to symmetric / hash-based constructions [1, 2]. The minimum over the three legs is the upper bound. □

5.4 Why this is not weaker than the OR composition

A common confusion in PQ cert design is whether the legs should be OR-composed ("valid if any leg verifies") or AND-composed ("valid only if all populated legs verify"). The OR composition is *weaker*: forging the cert requires forging only the weakest leg. The AND composition (conjunctive) is what we use: forging requires forging *all* populated legs.

The accompanying analysis in `lux/proofs/pq-finality-no-bls.tex` [21] spells this out for the BLS+PQ composition:

- Under AND composition: $\mathsf{Adv}^{\text{forge}} \leq \min(\epsilon_{\text{BLS}}, \epsilon_{\text{PQ}})$. A quantum adversary still must forge the PQ leg.
- Under OR composition: $\mathsf{Adv}^{\text{forge}} \leq \min(\epsilon_{\text{BLS}}, \epsilon_{\text{PQ}})$? No — *the bound is wrong.* Under OR, the cert verifies as soon as one leg does, so the adversary need only forge the *easier* leg, which under quantum attack is BLS ($\epsilon_{\text{BLS}} = O(1)$). PQ security is reduced to zero.

QuasarCert uses the AND composition: a cert with the Pulsar profile verifies iff the BLS leg verifies *and* the Pulsar leg verifies *and* the Z-Chain leg verifies. This is what gives the conjunctive hardness diversification of Theorem 5.1.

5.5 Subject-binding soundness

Theorem 5.4 (Subject-binding soundness)**.** *Suppose $\mathcal{C}$ is a valid QuasarCert for block B at chain c, epoch e, round r under profile $\mathcal{P}$. Then $\mathcal{C}$ is not a valid cert for any (c', e', r', B') with $(c, e, r, B) \neq (c', e', r', B')$, except with negligible probability.*

Proof. Theorem 3.3 establishes this for the BLS / Pulsar / Corona / Magnetar legs (each leg is bound to a unique μ). The Z-Chain leg is bound to $h_x = H_{\mathsf{Poseidon}}(\{pk_i^{\text{ML}}\} \,\|\, \mu)$; collision resistance of Poseidon [18] ensures a Z-Chain proof for μ does not verify against $\mu' \neq \mu$. The profile-match clause (Definition 3.2 clause 1) is deterministic in the cert byte string, and is independent of the block being verified. □

Remark 5.5 (Static vs adaptive corruption)**.** *The proofs above are stated under static corruption: the corrupt set* Corrupt *is fixed before $\mathcal{A}$ begins the soundness game. Adaptive corruption is handled via secure-erasure or programmable-RO hybrids [15] with at most an $O(n \cdot Q_H)$ multiplicative loss, where Q_H is the random-oracle query count. The Lux production deployment uses Go-side memory zeroing of all per-signature randomness (and HSM-backed signing on M-Chain custody deployments) to satisfy the erasure precondition; this is documented in the per-primitive deployment runbooks [20, 28, 33].*

Remark 5.6 (Aggregator-TCB in v0.1 primitives)**.** *Pulsar v0.1 and Magnetar v0.1 share a common aggregator-as-TCB caveat: the threshold-combine step is performed by a single aggregator that holds enough material to construct the final signature. A malicious aggregator can fail to produce a signature (a liveness fault) but cannot produce a forgery, because the aggregator's material is derived solely from honest parties' contributions: forging a signature requires forging at the underlying primitive level, not just compromising the aggregator. The reductions above hold even when the aggregator is adversarial. The Tier B status of Magnetar v0.1 (and the*

documented gating of the Polaris profile on Tier A audit) reflects that this caveat has not yet been audited end-to-end [33].

6 Consensus Engine Binding

6.1 The Lux consensus engine

The Lux consensus engine [24] is a unified driver for two modes: linear-chain (Nova) and DAG (Nebula). Both modes follow the same per-round protocol family — Photon (committee selection), Wave (threshold voting + Fast Probabilistic Consensus), Focus (consecutive-success counter), Prism (DAG geometry / cuts), Flare (cert / skip via $2f+1$ quorum), Horizon (DAG safe-prefix), Ray (linear finality driver), Field (DAG finality driver). The full package layout is documented at `luxfi/consensus/protocol/`.

Quasar lives at `protocol/quasar/` and is the cert-producing layer that the consensus driver invokes once a $2f+1$ quorum is observed. The driver does not know what kind of cert Quasar produces; it only knows that Quasar emits a `QuasarCert` byte string that it forwards to the block's `Cert` field. Verification on incoming blocks is symmetrical: the driver passes the cert bytes to `quasar.Verify`; the engine treats the result as a black-box boolean.

This is the orthogonality discipline: the consensus engine handles ordering, voting, and reachability; Quasar handles cryptographic finality; the QuasarCert is the only shared interface between them.

6.2 Per-round witness pattern

A consensus round at height h proceeds as follows. The committee C_h is selected via Photon's K-of-N scheme; the proposer broadcasts the block B_h over the ZAP wire protocol [29]; voting proceeds via Wave until a $2f+1$ quorum is observed; the engine invokes `quasar.Sign(committee, block, profile)` to produce $\mathcal{C}_h$.

Inside `quasar.Sign`, the legs are signed in parallel:

- The BLS leg is the aggregate of per-validator BLS signatures on the round digest μ. Each validator signs locally and broadcasts; the engine collects $\geq \tau$ signatures and aggregates by point addition (Section 2).
- The Pulsar leg is produced by the Pulsar two-round threshold protocol [20]: Round-1 commitments $\mathbf{w}_i = \mathbf{A}\mathbf{y}_i$ are broadcast; the aggregator collects them, computes the challenge $c = H_{\mathsf{Puls}}(\mu \,\|\, \sum_i \mathbf{w}_i)$, and forwards to Round-2; validators emit $\mathbf{z}_i$; the aggregator combines via Lagrange interpolation and emits the byte-equal FIPS 204 signature.
- The Corona leg (on Aurora / Polaris) follows the same two-round structure on the Module-LWE ring R_q [28, 10].
- The Magnetar leg (on Polaris) follows the v0.1 reveal-and-aggregate protocol [33]: validators emit Shamir shares of the master seed; the aggregator reconstructs S, calls `slhdsa.SignDeterministic`, and zeroizes S.
- The Z-Chain leg is produced asynchronously by Z-Chain validators: per-validator ML-DSA-65 signatures are streamed to Z-Chain, which produces a Groth16 proof of valid signature quorum over a window of $W \approx 1024$ rounds. The proof is emitted once per window, not per block; for blocks within a window without a fresh proof, the cert carries the most recent valid proof bound to the same epoch.

The legs are signed in parallel because they are independent cryptographic operations on the same round digest μ. The cert is assembled by concatenating the leg bytes per the wire format of Section 3, and is broadcast as part of the block's `Cert` field.

6.3 Verification flow

On the receiving side, the consensus engine invokes `quasar.Verify(cert, block, validators, profile)` which performs the predicate of Definition 3.2. The verify function is stateless — it depends only on its arguments — and returns a boolean. The engine accepts the block iff verify returns true; rejection causes the block to be dropped from the local DAG or chain.

Because each leg has its own verifier, the legs may be verified in parallel. The Lux `quasar.Verify` implementation dispatches each leg to its primitive verifier as a goroutine, with a fast-fail: if any leg returns false, the others are cancelled. This achieves the lowest verify latency (bounded by the slowest leg, not the sum).

6.4 Quasar in the linear-chain and DAG drivers

In the linear-chain driver (`Nova`), Quasar is invoked once per finalized block. The cert is stored on the block and propagated to peers via ZAP.

In the DAG driver (`Nebula`), Quasar is invoked once per finalized vertex along the DAG's safe-prefix commitment line [29]. The cert is stored on the vertex and the DAG store (`core/dag/`) indexes certs by vertex ID for light-client querying.

Both drivers use the *same* `quasar.QuasarCert` byte structure — the wire format is independent of the consensus mode. This is critical because Lux supports mixed-mode L1 topologies: a chain may run linear-chain consensus while its parent runs DAG consensus, and certs flow between them via Warp messaging.

6.5 Warp messaging carries cert finality

Cross-chain messaging on Lux uses the Warp protocol [22, 31]: a message from chain c_1 to chain c_2 is signed by c_1's validators and is verified by c_2's validators using c_1's known public keys.

A Warp message carries the source chain's QuasarCert as the finality witness. The destination chain runs `quasar.Verify(cert, block, validators, profile)` against the source chain's profile and validator set, then trusts the message as finalized.

This means the QuasarCert is the universal finality envelope: every cross-chain interaction in the Lux ecosystem carries a verifiable PQ-finality witness. The wire-format stability commitment in Section 3 is what makes this work: a single `QuasarCert` can be verified against any Lux chain's validator set, profile, and primitives.

6.6 Z-Chain rollup async dependency

The Z-Chain Groth16 leg has a unique structure: it is not produced per-block on the source chain; it is produced asynchronously by Z-Chain validators over a window. This means the ZK leg of $\mathcal{C}_h$ may carry a Groth16 proof produced at height $h - k$ for some $k \in [0, W)$, where W is the rollup window size.

The freshness invariant is that the proof must be bound to the same *epoch* as the block; an epoch boundary forces a fresh proof. This is enforced by the round-digest construction (Section 2): μ includes the epoch e, so a proof bound to epoch e cannot be replayed for epoch $e' \neq e$.

Within an epoch, the proof's freshness window provides bandwidth amortization: a single Groth16 proof can cover up to $W \approx 1024$ blocks before a new proof is required. This amortizes the ∼400 ms CPU prover time (or ∼5–15 ms GPU prover time; see Appendix B of [22]) across the window.

The trade-off: a block at height h may carry a proof produced for the validator-set state at height $h - k$. If a validator changes keys mid-epoch, the proof remains valid against the older root, but new validators may not appear in the proof. This is acceptable because the

validator set is stable within an epoch (validator-set rotations happen at epoch boundaries by construction).

6.7 Dynamic resharing

The Pulsar / Corona / Magnetar group public keys must remain stable across validator-set rotations, otherwise a key rotation would invalidate the cert format. This is achieved via verifiable secret resharing [40]: when the validator set rotates from $\mathcal{V}_e$ to $\mathcal{V}_{e+1}$, each retiring validator shares its threshold secret with the incoming set under the same group public key.

The Lux Secret Sharing (LSS) framework [40] is the generic adapter for this lifecycle. Pulsar instantiates LSS with the Pulsar key-era / resharing / activation-cert lifecycle [20]; Corona will do the same once its Tier A documentation completes (current status Tier A docs landed [28]); Magnetar's reshare semantics under the v0.1 reveal-and-aggregate construction are degenerate (because the master seed is the durable secret, not the per-party shares; resharing is a fresh DKG).

The key observation is that resharing is *orthogonal* to the QuasarCert format: the cert carries no resharing transcript; the resharing is governed by the LSS lifecycle and committed to the `validator_root`$_e$ Merkle root at epoch boundaries. The cert sees only the post-resharing public keys, never the resharing process itself.

7 Implementation

7.1 Repository topology

The Quasar suite is implemented across the following Lux repositories, each of which is independently versioned and tagged:

Repo	Role	Pinned Tag
`luxfi/quasar`	Umbrella spec / profile registry / composition proofs	v0.1.0
`luxfi/consensus`	Quasar consensus engine, cert wire impl	v1.22.85
`luxfi/pulsar`	Threshold ML-DSA-65 primitive (Tier A)	v1.0.9
`luxfi/corona`	Threshold Module-LWE primitive (Tier A docs)	v0.6.0
`luxfi/magnetar`	Threshold SLH-DSA primitive (Tier B)	v0.2.0
`luxfi/crypto`	BLS / ML-DSA / SLH-DSA / ML-KEM primitives	v1.19.x
`luxfi/threshold`	Threshold MPC protocol library	v1.7.0+
`luxfi/chains/quantumvm`	Z-Chain Groth16 rollup VM	v1.2.3+
`luxfi/lens`	Curve-side threshold sister kernel	pinned

Table 3: Quasar suite repository layout. Each repository is an independent unit of release; cross-repo coordination happens at profile-rotation events (see Section 4.4).

The umbrella repository `luxfi/quasar` [27] owns only the wire-format contract (`SPEC.md`), the profile registry (`PROFILES.md`), the primitive version pins (`PRIMITIVES.md`), and the cross-primitive composition proofs (`proofs/`). It contains no Go code; it is documentation + proofs + reference vectors.

The consensus engine `luxfi/consensus` [24] contains the Go canonical implementation of the QuasarCert wire format, the signing and verification entry points, and the integration with the linear-chain (Nova) and DAG (Nebula) drivers. The cert struct lives at `protocol/quasar/types.go` (line 40), and the native binary encoder lives at `protocol/quasar/corona_gob.go`.

Each primitive lives in its own repo with its own documentation package and its own NIST MPTC [37] submission trajectory. The umbrella does not re-implement primitives, and the primitives do not depend on the umbrella; they are connected only via the wire-format contract.

7.2 Pulsar: threshold ML-DSA-65 (Tier A)

The Pulsar primitive [20] is the threshold ML-DSA-65 construction at `luxfi/pulsar`, pinned at `v1.0.9` (`5d00be9`). It is Tier A on the NIST MPTC submission readiness ladder: 14 documentation artifacts complete, EasyCrypt theory shells 13/13, Lean mechanization 5/5, Jasmin verified-implementation shells 3/3, KAT determinism asserted, 19/19 NIST N1 interop tests passing, cryptographer sign-off APPROVED WITH GATES, cut-submission dry-run validated for submission to NIST MPTC v0.1 on 2026-11-16.

Pulsar provides:

- Joint Pedersen-style DKG over R_q with explicit hiding / binding proofs and a complaint workflow [20, 17, 38].
- Two-round threshold signing with rejection sampling, producing byte-equal FIPS 204 ML-DSA-65 signatures.
- Verifiable secret resharing across validator-set rotations via LSS [40].
- Activation certificates that bind a generation's public key into the next.
- Identifiable abort on Round-1 / Round-2 message inconsistency.

The Pulsar primitive sits in the QuasarCert as the σ^{Puls} leg. It is the PQ-floor primitive — present on every profile, NIST-standardized via FIPS 204, single-party-compatible via Class N1.

7.3 Corona: threshold M-LWE (Tier A documentation)

Corona [28, 10] is the Lux production fork at `luxfi/corona`, pinned at `v0.6.0`. The documentation shape is Tier A complete (full Pulsar-template package landed: SUBMISSION + NIST-SUBMISSION + SPEC + PROOF-CLAIMS + AXIOM-INVENTORY + FIPS-TRACEABILITY + TRUSTED-COMPUTING-BASE + PATENTS + CRYPTOGRAPHER-SIGN-OFF + DEPLOYMENT-RUNBOOK); the cryptographer verdict is APPROVED WITH GATES. Open Tier A gates: EasyCrypt theory shells (GATE-1, v0.7.0), Lean-EasyCrypt bridge (GATE-2, v0.7.0), dudect 10^9 on threshold layer (GATE-3, v0.8.0), external audit (GATE-4, v0.8.0). Construction soundness is inherited from Boschini et al. ePrint 2024/1113 / IEEE S&P 2025.

The upstream academic name was "Corona"; this has been retired in the Lux ecosystem, and both the production fork and the protocol family are now "Corona".

Corona provides the σ^{Cor} leg of the QuasarCert on the Aurora and Polaris profiles. The Corona group key is derived from a parallel DKG run alongside the Pulsar DKG at epoch boundaries.

7.4 Magnetar: threshold SLH-DSA (Tier B, v0.1)

Magnetar [33] is the v0.1 reveal-and-aggregate threshold SLH-DSA construction at `luxfi/magnetar`, pinned at `v0.2.0`. The construction is byte-wise Shamir VSS over GF(257) on the SLH-DSA scheme seed; Combine reconstructs the seed in aggregator memory, calls single-party `slhdsa.SignDeterministic`, and zeroizes. This mirrors Pulsar's v0.1 trust model: a brief aggregator-as-TCB window during which the seed must be erased.

Tier B reflects that the construction has been implemented and internally reviewed, but has not yet been externally audited and does not have the full Tier A artifact bundle (EasyCrypt theory, Lean mechanization, Jasmin verified implementation). The Polaris profile depends on Magnetar reaching Tier A; the production deployment plan reserves Polaris for after Magnetar's Tier A audit completes (target 2027 Q3 per `luxfi/magnetar`'s `SUBMISSION-STATUS.md`).

Magnetar's signature is byte-identical to FIPS 205 SLH-DSA on the reconstructed seed: `TestN1_ByteEquality` across $(3, 2)$, $(5, 3)$, $(7, 4)$ committees verifies this property in CI. The threshold output passes the unmodified NIST SLH-DSA verifier.

7.5 BLS: BLS12-381 aggregate

The BLS primitive [25] lives at `luxfi/crypto/bls` and tracks the IETF BLS draft [8]. The aggregate signature is produced via point addition; the threshold variant (for chains that desire threshold-BLS custody) lives at `luxfi/threshold/protocols/bls` [26].

The QuasarCert BLS leg uses the aggregate variant, not threshold: each validator signs individually with its BLS-PoP key, and the aggregate is computed by the proposer. The proof-of-possession defeats the rogue-key attack [5, 7]: each public key is published with a BLS signature on the key under itself, in a domain-separated context.

7.6 Z-Chain Groth16 rollup

The Z-Chain Groth16 VM lives at `luxfi/chains/quantumvm` [23]. The circuit verifies ML-DSA-65 verification at $\sim 2^{22.5}$ R1CS constraints per signature, amortized to $\sim 2^{20}$ constraints per cert at $n = 21$ via the shared-matrix optimization (Appendix B of [22]).

The trusted setup is the powers-of-tau ceremony plus a circuit-specific phase 2 with ≥ 1024 contributors [11]. The verifying-key hash is rooted on Z-Chain as $\mathsf{zchain\_vk\_root}$ and committed into the validator-set Merkle root.

The Z-Chain leg is asynchronous: it produces one Groth16 proof per window of ~ 1024 blocks (amortizing prover cost). Within an epoch, blocks reuse the most recent valid proof.

The PLONK upgrade path is documented in Appendix D of [22]; the proof system is swappable without changing the wire format because the Z-Chain leg is π^{ZK}, a 192-byte opaque proof byte string.

7.7 Build and test reproducibility

The full Quasar suite is reproducibly buildable from source. Reproducibility is anchored on:

- Pinned Go versions: 1.26.1 in `luxfi/consensus` and the primitive repos.
- Pinned module versions: `go.mod` files in each repo pin all dependencies at exact tags.
- KAT vectors: each primitive ships golden test vectors that assert byte-equality with the NIST reference implementation (for FIPS 203/204/205) or with the academic reference implementation (for Corona; see `luxfi/corona/vectors/`).
- Constant-time review: each primitive ships a `CONSTANT-TIME-REVIEW.md` (Pulsar) / `ct/` (Corona, Magnetar) documenting which functions have been audited to be constant-time, and which use the dudect statistical test (currently 10^7 samples; target 10^9 for Tier A).

Build commands:

```
# Consensus engine
cd ~/work/lux/consensus && GOWORK=off go build ./... && GOWORK=off go test ./...

# Pulsar primitive
cd ~/work/lux/pulsar && go test ./... -tags=production_kat
cd ~/work/lux/pulsar && ./scripts/cut-submission.sh --dry-run

# Corona primitive
cd ~/work/lux/corona && go test ./...

# Magnetar primitive
cd ~/work/lux/magnetar && go test ./...

# Umbrella spec proofs (LaTeX)
cd ~/work/lux/quasar && latexmk -pdf proofs/composition.tex
```

7.8 GPU dispatch

The GPU-native crypto stack [32] provides Metal (Apple), CUDA (NVIDIA), and WGSL (cross-vendor) dispatchers for the verify-heavy operations:

- BLS aggregate verify: GPU dispatch at ≥ 64 signers gives $\sim$ 20–24$\times$ speedup over CPU (Metal M2 Pro measurement; `lux-pq-consensus-benchmarks/`).
- ML-DSA-65 verify: GPU dispatch at ≥ 64 signatures gives $\sim$ 8–12$\times$ speedup.
- Groth16 verify: GPU dispatch reduces the three-pairing check from 1–3 ms CPU to 200–500 µs GPU.

The break-even point is $\sim$ 64 signatures because of the $\sim$ 100 µs Metal/CUDA dispatch overhead. Below that threshold, the CPU single-verify path is faster.

For the QuasarCert specifically, GPU dispatch is most useful at the aggregate-verify boundary: when a validator receives a batch of certs (during catch-up sync) it can dispatch the BLS legs and the Groth16 legs as GPU batches, amortizing the dispatch overhead.

8 Evaluation

8.1 Cert sizes

Table 4 gives the measured cert sizes per profile at the production parameter set ($n = 21$ validators, Corona M=8 N=7 ring degree 256 q=$2^{48} + 0x4A01$ (Module-LWE, Ringtail params), Pulsar M-LWE $N = 256$ q=$2^{48} + 2561$, Magnetar SLH-DSA-SHAKE-192f).

Profile	BLS	Pulsar	Corona	Magnetar	Groth16	Total
`Pulsar`	48 B	3,309 B	—	—	192 B	∼3,549 B
`Aurora`	48 B	3,309 B	33,052 B	—	192 B	∼36,601 B
`Polaris`	48 B	3,309 B	33,052 B	16,224 B	192 B	∼52,825 B

Table 4: Cert size per profile at $n = 21$. Sizes are dominated by the threshold-signature wire format on each leg. Native binary encoding (`corona_gob.go`) is 1.01$\times$ gob; numbers above are the native binary form (the canonical wire bytes).

The Pulsar profile is bandwidth-efficient: $\sim$ 3.5 KB per cert is dominated by the Pulsar ML-DSA-65 signature ($\sim$ 93 % of the cert). The Aurora profile is $\sim$ 10$\times$ larger due to the Corona threshold-signature wire format ($\sim$ 33 KB). The Polaris profile adds the Magnetar leg ($\sim$ 16 KB).

Comparison with naive per-validator ML-DSA. A naive cert that includes per-validator ML-DSA-65 signatures (no threshold combination, no Groth16 rollup) would carry $n \cdot 3{,}309 = 69{,}489$ bytes at $n = 21$, or 330,900 bytes at $n = 100$. The QuasarCert Pulsar profile is 19$\times$ smaller at $n = 21$ and 93$\times$ smaller at $n = 100$, with no degradation in PQ-security.

8.2 Storage projections

For a chain producing one block per 500 ms (the Lux Q-Chain target block time), 86,400 blocks/day. Table 5 gives the storage growth per profile.

For the Aurora / Polaris profiles, the cert storage cost is non-trivial. Two amortizations apply.

Epoch-mode amortization. In "epoch mode" the Corona and Magnetar legs are signed once per epoch (over a Merkle root of that epoch's block hashes), not per block. With epoch size $k = 1000$ blocks, the amortized Corona / Magnetar cost is $\sim$ 33 B/block and $\sim$ 16 B/block

Profile	Per 10K blocks	Per day	Per year
`Pulsar`	33.8 MB	292 MB	107 GB
`Aurora`	349 MB	3.0 GB	1.1 TB
`Polaris`	503 MB	4.3 GB	1.6 TB

Table 5: Storage projection per cert profile at one block per 500 ms. The Pulsar profile sustains $\sim$ 100 GB of cert storage per year; the Aurora and Polaris profiles require $\sim$ 1 TB/year. Archival nodes hold these in full; light clients sync via Z-Chain epoch proofs and store $\sim$ 10 KB/year.

respectively, reducing the Aurora profile to $\sim$ 3.6 KB/block and Polaris to $\sim$ 3.6 KB/block plus a one-time epoch-end overhead.

Z-Chain Groth16 amortization. The Z-Chain leg is already amortized: one 192-byte proof per window of $\sim$ 1024 blocks. The per-block Z-Chain cost is therefore $\sim$ 0.2 B/block, negligible compared to the other legs.

8.3 Verify times

Table 6 gives measured CPU verify times on Apple M1 Max (10 cores, darwin/arm64) from the `luxfi/consensus` bench harness [24].

Operation	Time	Source
BLS single verify	820 µs	`crypto/bls BenchmarkVerify`
BLS aggregate verify (100 signers)	875 µs	`quasar BenchmarkBLSAggregatedVerification/100`
ML-DSA-65 verify (cold)	254 µs	`utxo/mldsafx BenchmarkMLDSA65Verify`
ML-DSA-65 verify (cached)	3 µs	`utxo/mldsafx BenchmarkMLDSA65VerifyCached`
SLH-DSA-192f verify	1,920 µs	`utxo/slhdsafx BenchmarkSLH192fVerify`
Groth16 verify (CPU)	1–3 ms	not yet in bench harness
Pulsar threshold verify	$O(1)$ after DKG	amortized
Full QuasarCert verify (Pulsar profile)	1.85 ms	`quasar BenchmarkQuasarBlockProcessing`
Full QuasarCert verify (Aurora profile, est.)	$\sim$4–5 ms	projection
Full QuasarCert verify (Polaris profile, est.)	$\sim$6–8 ms	projection

Table 6: Measured and projected QuasarCert verify times. The Pulsar profile clears in 1.85 ms; Aurora and Polaris are correspondingly larger due to the additional lattice and hash-based leg verifies. GPU batch verification amortizes 10–100$\times$ across multiple certs at $\geq$ 64 signers.

The 357 µs legend. Earlier Lux drafts (`lux-triple-proof-consensus`, `lux-master-security-model`) quoted "357 µs epoch finality". This number does not match any measured operation in the current code. The closest real candidates are BLS single keygen (350 µs) and ML-DSA-65 sign (495 µs); the actual QuasarCert verify on Pulsar profile is 1.85 ms CPU. Papers should quote the measured 2–5 ms CPU QuasarCert verify (or $\sim$ 0.5–1.5 ms GPU batched), not the 357 µs legend.

8.4 Cert assembly latency

The cert assembly latency is dominated by the slowest leg's signing operation, because legs are signed in parallel. Table 7 gives the measured per-leg signing times.

The Lux Q-Chain target block time is 500 ms. The Pulsar-profile cert assembly fits well within this budget ($\sim$ 90 ms wall-clock across signing and verify). The Aurora-profile cert

Leg signing operation	Time (M1 Max)
BLS single sign	350 µs
BLS aggregate (100 sigs)	5.3 ms
ML-DSA-65 sign	495 µs
Pulsar Round 1 (precomputation)	∼50 ms per validator
Pulsar Round 2 (response)	∼30 ms per validator
Pulsar combine ($n = 21$)	∼40 ms
Corona Round 1 + Round 2 + combine	∼50–100 ms
Magnetar reveal + sign + zeroize	∼5–10 ms (SLH-DSA-192f sign 1.92 ms verify)
Groth16 prove (CPU)	∼400 ms per ML-DSA verification
Groth16 prove (GPU batched)	∼5–15 ms amortized

Table 7: Per-leg signing times. The cert assembly latency is dominated by the Pulsar / Corona threshold-combine step (∼ 40–100 ms), not the BLS aggregate or the FIPS 204 single-party sign.

assembly is ∼ 200 ms; Polaris is ∼ 250 ms. All profiles fit within the 500 ms block-time budget with significant headroom.

8.5 Throughput at scale

The ZAP wire protocol [29] sustains ≥ 100K TPS per chain on Lux production hardware (`luxfi/consensus/bench` measurements; 11.5M TPS in optimal end-to-end conditions on 50-connection batching). The cert assembly cost is amortized across the block payload, not per-transaction; a 500 ms block with 50,000 transactions has a per-tx overhead of ∼ 90 µs for the Pulsar profile.

8.6 GPU dispatch impact

GPU batch verification (≥ 64 signers / signatures) provides:

- BLS aggregate verify: ∼ 875 µs CPU → ∼ 40 µs GPU (M2 Pro), 20× speedup.
- ML-DSA-65 verify: ∼ 254 µs CPU → ∼ 20 µs GPU, 10× speedup.
- Groth16 verify: ∼ 1–3 ms CPU → ∼ 200–500 µs GPU, 5–10× speedup.

For a validator catching up after downtime, GPU batch verify amortizes the entire cert-history replay cost. At 20× speedup over CPU, a year of Pulsar-profile certs (∼ 63M certs) verifies in ∼ 30 minutes GPU vs. ∼ 10 hours CPU.

The break-even for GPU dispatch is ∼ 64 signers / signatures, because of the ∼ 100 µs Metal/CUDA dispatch overhead. Below this threshold, the CPU path is faster.

8.7 Comparison with naive PQ designs

Table 8 compares QuasarCert with two naive alternatives.

Design	Cert size	Verify time	PQ security
BLS-only (legacy)	48 B	875 µs	none
Per-validator ML-DSA	69 KB	5.3 ms	yes
QuasarCert Pulsar profile	3.5 KB	1.85 ms	yes
QuasarCert Aurora profile	37 KB	∼4–5 ms	yes (intra-lattice)
QuasarCert Polaris profile	53 KB	∼6–8 ms	yes (cross-family)

Table 8: QuasarCert vs. naive PQ designs at $n = 21$. The Pulsar profile is 20× smaller and 3× faster to verify than per-validator ML-DSA, with the same EUF-CMA bound. The intra-lattice and cross-family profiles trade size for additional hardness diversification.

9 Threat Model

9.1 Adversarial setting

QuasarCert is designed against an adversary who controls a strict-minority subset of validators and observes all network traffic. Specifically:

- **Byzantine fraction**: $f < 1/3$ in count and stake. Quorum threshold $\tau = \lfloor 2n/3 \rfloor + 1$.
- **Static vs adaptive corruption**: theorems are stated for static corruption; adaptive corruption is handled via programmable-RO or secure-erasure hybrids [15] with $O(n \cdot Q_H)$ multiplicative loss. Production deployment uses Go-side memory zeroing for all per-signature randomness, satisfying the secure-erasure precondition.
- **Network adversary**: full Dolev-Yao on the message passing layer; the adversary can drop, reorder, replay, and forge messages. Replay is prevented by round-digest binding (Theorem 3.3); message authentication is via the cryptographic legs of the cert itself.
- **Quantum adversary** ($\mathcal{A}_Q$): polynomial-time on a quantum Turing machine; runs Shor's algorithm on classical discrete logarithms; has QROM access [3]; subject to Grover speedup against unstructured search. Under $\mathcal{A}_Q$, the BLS and Z-Chain legs break in polynomial time; the PQ legs remain hard (Corollary 5.2).
- **Side-channel adversary**: passive timing / cache / power-analysis observer of validator processes. All threshold primitives are constant-time in their secret-dependent paths (documented per-primitive in `CONSTANT-TIME-REVIEW.md` / `ct/` subdirs).

9.2 Threat surface boundaries

In scope. The QuasarCert verifier is in scope. Specifically:

- Cert wire format and verification predicate (Definition 3.2).
- Profile selection and profile invariants (Section 4).
- Replay protection via round-digest binding (Theorem 3.3).
- Cross-primitive composition soundness (Theorem 5.1).
- Domain-separation contexts (Table 1).
- Per-leg primitive unforgeability (delegated to per-primitive papers; see [20, 28, 33, 22]).

Out of scope (delegated to other layers).

- Network-level partition tolerance: Lux uses a partial-synchrony assumption; full asynchronous safety / liveness is a property of the underlying Snow consensus family [29, 16], not the cert verifier.
- Validator-set rotation: governed by the LSS framework [40] and the per-primitive resharing protocols ([20] for Pulsar, etc.). The cert sees only the post-resharing public keys.
- Trusted setup for Z-Chain Groth16: governed by the Bowe-Gabizon-Miers ceremony [11]; the verifying key is rooted on Z-Chain as `zchain_vk_root`.
- DKG-time security: each primitive runs its own DKG independently; the umbrella spec does not coordinate DKGs. Pulsar uses Pedersen-style DKG over R_q; Corona uses a similar lattice DKG; Magnetar uses byte-wise Shamir VSS over GF(257).
- EVM precompile bindings: each PQ verifier is exposed via an EVM precompile (e.g., P3Q at slot `0x012205`); precompile-layer profile gating uses `contract.RefuseUnderStrictPQ` in `luxfi/evm/precompile/`; cert verification is orthogonal to precompile verification.

9.3 Failure modes

We enumerate the failure modes and their consequences.

Single-primitive break. An adversary breaks one PQ leg's underlying assumption (e.g., a cryptanalytic break against M-LWE specifically). On the Aurora profile, this would compromise the Pulsar leg. The cert remains unforgeable because the Corona leg (Module-LWE) is independent and still valid (Corollary 5.3). The compromised leg's primitive must be deprecated and replaced; the wire format and other legs are unaffected.

Cross-lattice break. An adversary breaks both M-LWE and Module-LWE (some breakthrough sieving / NTT-aware attack). On the Aurora profile, this would compromise both Pulsar and Corona; the profile no longer provides PQ security beyond the failing ZK leg's MLDSA witness (which is also lattice-based). On the Polaris profile, the Magnetar / hash-based leg remains valid. This is the cross-family insurance that motivates Polaris.

Hash break. An adversary breaks SHA-3 / SHAKE preimage or target-collision resistance. On any profile, the round-digest construction would be compromised. This is a catastrophic event that affects all of cryptography, not just QuasarCert; the remediation is to migrate to a different hash function and re-issue all validator keys. The QuasarCert wire format makes this swap non-disruptive in principle (replace H_{round} with a new hash) but the per-primitive keys would need full rotation.

Trusted-setup compromise. If an adversary compromises the Z-Chain Groth16 trusted setup (e.g., the $\geq$ 1024-contributor ceremony [11]), they can forge π^{ZK}. By Theorem 5.1, the cert is not forgeable because the other legs (BLS / Pulsar / Corona / Magnetar) remain unbroken. The Z-Chain vk_Z would be rotated to a fresh ceremony output; the cert format is unchanged.

Aggregator-as-TCB (Pulsar v0.1 / Magnetar v0.1). A malicious aggregator can refuse to combine signatures (liveness fault) but cannot produce a forgery (the aggregator's material is purely derived from honest contributions; forging requires forging at the underlying primitive). The fault is identifiable abort [12]; the consensus engine excludes the faulty aggregator from future rounds.

Byzantine quorum. An adversary controls $\geq 2/3$ of validators. The cert remains valid (because the Byzantine signers sign correctly under the protocol) but the consensus engine's safety property is violated: the adversary can finalize conflicting blocks. This is outside QuasarCert's scope — QuasarCert is a finality certificate, not a Byzantine-quorum gate. Quorum intersection at $2f + 1$ of $3f + 1$ is proved by the Lean `lean/Consensus/BFT.lean` [22].

9.4 Migration risk

The PQ-migration era is the period during which a quantum adversary becomes capable of breaking classical primitives in polynomial time but the legacy classical primitives are still deployed. QuasarCert addresses this risk in two ways:

- The PQ leg is always populated (profile invariant 2). A chain configured with the Pulsar profile is PQ-safe from day one; no migration is required.
- The classical BLS leg is supplementary, not load-bearing. Under quantum attack, the BLS leg becomes trivially forgeable ($\epsilon_{\mathrm{BLS}} = O(1)$); by Corollary 5.2, the cert's unforgeability is bounded by the PQ leg's hardness, which remains intact.

The decision to keep BLS in the cert (rather than removing it entirely; see [21] for the BLS-removal analysis) is a deployment-pragmatic one: BLS aggregate verify is sub-millisecond, much faster than any PQ verifier; on classical hardware, the cert verifies fast because BLS is checked first; on quantum hardware, the cert is still safe because the PQ leg is also verified.

The reverse decision — remove BLS, run only PQ legs — is also implemented in `luxfi/consensus` as the "pure PQ" mode (`pq-finality-no-bls`), gated behind a chain-config flag. This is the migration target for chains that desire to retire all classical cryptography from the consensus path.

10 Production Deployment

10.1 Q-Chain finality on Lux primary network

The current production deployment of QuasarCert is on the Lux primary network's Q-Chain, where it provides PQ-safe finality for the cross-chain Warp messaging layer. The Q-Chain runs the `Pulsar` cert profile (BLS + Pulsar + Z-Chain Groth16), which is the smallest profile that satisfies the PQ-floor requirement.

Q-Chain validators are the Lux primary network validators (the same validator set that secures the P-Chain, C-Chain, and X-Chain). Each validator holds:

- A BLS12-381 keypair with PoP for the QuasarCert BLS leg.
- A Pulsar threshold-secret share for the QuasarCert Pulsar leg.
- A per-validator ML-DSA-65 keypair for the Z-Chain Groth16 leg.

The Pulsar DKG is run once per validator-set epoch by Q-Chain validators; the resulting group public key pk^{Puls} is committed to the validator-set Merkle root for that epoch.

10.2 LP-020 Quasar consensus activation

The Quasar 3.0 protocol was activated on the Lux primary network on 2025-12-25 per LP-020 [29]. The activation enabled the three-lane cert (BLS + Pulsar + Z-Chain Groth16) on Q-Chain blocks from height h_{act} onwards. Pre-activation blocks carry BLS-only certs (legacy); post-activation blocks carry full QuasarCerts under the Pulsar profile.

The activation roadmap (LP-020 §14) committed to:

- Pulsar primitive at Tier A submission readiness (achieved 2026-05-18 per the `CRYPTOGRAPHER-SIGN-OFF.md` in `luxfi/pulsar` commit `443c725`).
- Aurora profile reserved for the deployment phase after Corona's Tier A audit (target: late 2026, gated on Corona's open GATE-1/2/3/4 closure per `luxfi/corona`'s submission status).
- Polaris profile reserved for after Magnetar's Tier A audit (target: 2027 Q3 per `luxfi/magnetar/SUBMISSION`

10.3 Profile gating for Aurora and Polaris

The Aurora and Polaris profiles are operationally ready in the `luxfi/consensus` implementation but are not yet enabled by default on any Lux production chain. The gating is documented in `luxfi/consensus/config/pq_mode.go` and is bound to the maturity tier of the non-Pulsar primitives.

Aurora gating. Aurora requires Corona at Tier A submission readiness. As of 2026-05-18, Corona is at "Tier A documentation shape complete" with four open gates: EasyCrypt theory shells (GATE-1, v0.7.0), Lean↔EasyCrypt bridge (GATE-2, v0.7.0), dudect 10^9 on threshold layer (GATE-3, v0.8.0), and external audit (GATE-4, v0.8.0). The Aurora profile is unblocked once GATE-4 closes.

Polaris gating. Polaris requires both Corona Tier A and Magnetar Tier A. Magnetar is currently at Tier B (v0.1 reveal-and-aggregate implementation landed with internal review; full Tier A artifact bundle is on the roadmap). Polaris activation target is post-Magnetar-Tier-A (2027 Q3).

The gating discipline is: a profile is enabled in production only after every primitive on the profile has external audit and mechanized proofs (EasyCrypt / Lean / Jasmin). This is the production-grade bar; primitives at lower tiers are research references, not deployment surfaces.

10.4 Cross-chain finality propagation

The QuasarCert is the finality envelope for the Lux ecosystem's cross-chain messaging [31]. A message from chain c_1 to chain c_2 carries the source chain's QuasarCert; the destination chain verifies the cert against the source chain's known public keys.

The wire-format stability commitment (Section 3) is what makes this work: a Q-Chain QuasarCert, a C-Chain QuasarCert, and a chain-VM EVM QuasarCert all use the same Go struct, the same encoder, and the same verifier. The only variation across chains is the cert profile (`Pulsar` for Q-Chain; potentially `Aurora` or `Polaris` for high-assurance L1 chains in the future) and the validator-set / primitive public keys.

A Lux chain that adopts the Aurora profile can produce QuasarCerts that are interoperable with Pulsar-profile chains via downward compatibility: an Aurora cert verifies on a Pulsar-only verifier that ignores the Corona leg, with the verification rule degraded to profile-Pulsar correctness. (In practice, Lux disallows this downward verification because the profile-match clause of Definition 3.2 would reject the cert as malformed for the wrong profile. Cross-profile interoperability requires both chains to be configured for the same profile or for a profile rotation event.)

10.5 Operational runbooks

Each primitive ships an operational runbook documenting its deployment, key-rotation, and incident-response procedures.

- Pulsar: `luxfi/pulsar/DEPLOYMENT-RUNBOOK.md` covers DKG coordination, group-key publication, resharing across validator-set rotations, identifiable abort handling, and the v0.1 aggregator-TCB caveat with mitigation guidance.
- Corona: `luxfi/corona/DEPLOYMENT-RUNBOOK.md` (Tier A docs landed; equivalent shape to Pulsar's runbook).
- Magnetar: `luxfi/magnetar/DEPLOYMENT-RUNBOOK.md` covers the byte-wise Shamir VSS DKG, the reveal-and-aggregate signing flow, and the aggregator zeroization protocol.
- Quasar umbrella: profile rotation, primitive version bumps, and cross-repo cascade procedures documented at `luxfi/quasar/SPEC.md` §Wire-format stability commitment.

10.6 NIST MPTC submission trajectory

Each PQ primitive in QuasarCert is being submitted to the NIST MPTC project [37] as an independent threshold construction.

- Pulsar: scheduled for NIST MPTC v0.1 submission on 2026-11-16 (cut script dry-run validated; see `luxfi/pulsar/scripts/cut-submission.sh`).
- Corona: scheduled for NIST MPTC v0.2 (target: 2027 Q1).
- Magnetar: scheduled for NIST MPTC v0.3 (target: 2027 Q3).

NIST MPTC submission requires Class N1 (single-party-compatible threshold signing) and Class N4 (public-key preservation across resharing). Pulsar satisfies both N1 and N4 by construction (`TestN1_ByteEquality` and `TestN4_Resharing` in `luxfi/pulsar/threshold`); Corona and Magnetar will follow the same template.

10.7 Audit status

The QuasarCert composition (this paper) has been internally reviewed by the Lux cryptography team. Each individual primitive has its own audit trajectory:

- Pulsar: internal cryptographer sign-off APPROVED WITH GATES (2026-05-18). External audit gated on GATE-4 closure (target: 2026 Q4). EC theory shells 13/13, Lean 5/5, Jasmin 3/3 at `v1.0.7+`.
- Corona: internal cryptographer verdict APPROVED WITH GATES. External audit gated on GATE-4 closure (target: 2027 Q1).
- Magnetar: Tier B; not yet externally audited. The v0.1 construction has been internally red-team reviewed; the `BLOCKERS.md` file lists open issues.
- BLS12-381: existing literature audit [9, 5, 7]; IETF draft stabilization in progress.
- Z-Chain Groth16: Bowe-Gabizon-Miers ceremony audit [11]; circuit audit included in the `luxfi/chains/quantumvm` repo audit trajectory.

The umbrella composition proof (Theorem 5.1) is machine-checked in Lean at `luxfi/proofs/lean/Consensu` [22]. The Lean proof tree mirrors the LaTeX theorem numbers 1-to-1.

11 Conclusion

11.1 What QuasarCert is

QuasarCert is the wire-format finality certificate for the Lux Network's PQ consensus suite. It composes up to five independent cryptographic legs — a BLS12-381 aggregate, a Pulsar threshold ML-DSA-65 signature, an optional Corona threshold M-LWE signature, an optional Magnetar threshold SLH-DSA signature, and a Z-Chain Groth16 rollup of per-validator ML-DSA-65 identity signatures — under a strict conjunctive verifier. Three cert profiles (`Pulsar`, `Aurora`, `Polaris`) select which legs are populated, with profile-specific hardness diversification:

- `Pulsar` (PQ floor, $\sim$ 3.5 KB): co-CDH $\wedge$ M-LWE $\wedge$ KEA. Forging the cert under quantum attack requires breaking Pulsar.
- `Aurora` (intra-lattice, $\sim$ 37 KB): adds a second Module-LWE leg (Corona) at a distinct parameter set and codebase. Forging requires breaking both legs — parameter and implementation diversity within M-LWE, not a second hardness family.
- `Polaris` (cross-family, $\sim$ 53 KB): adds SHA-3 one-wayness. Forging requires breaking lattice and hash assumptions both.

11.2 What QuasarCert delivers

Hardness diversification, not redundancy. Each populated leg invokes an independent hardness assumption; the conjunctive soundness theorem (Theorem 5.1) bounds forgery by the minimum advantage over legs. A cryptanalytic break against one family does not transfer to the others (Corollaries 5.2, 5.3).

Wire-format stability. The QuasarCert byte layout is specified at `luxfi/quasar/SPEC.md` and is byte-compatible with the Go canonical at `luxfi/consensus/protocol/quasar/types.go`. Cross-chain finality propagation uses the same byte string regardless of consensus mode (linear-chain or DAG); cross-chain Warp messaging carries QuasarCerts as the universal finality envelope.

Composable primitives. Each primitive lives in its own repository with its own audit, version pin, and submission trajectory. The umbrella does not re-implement primitives; the primitives do not depend on the umbrella. The only shared contract is the wire format. This is the only sustainable PQ-stack architecture for the migration era: primitives must be swappable without rebuilding the engine.

Per-chain profile selection. A single Lux chain is configured with one profile via the `FinalitySchemeID` field on its consensus config. Different chains may run different profiles; cross-chain interoperability is via the universal QuasarCert envelope. Profile rotation is a coordinated multi-window event documented in the deployment runbooks.

11.3 Tier status by primitive

- Pulsar: **Tier A** (cut-ready for NIST MPTC v0.1 submission on 2026-11-16). Cryptographer sign-off APPROVED WITH GATES [20].
- Corona: **Tier A documentation shape complete**. Four open gates to full Tier A (EC / Lean↔EC / dudect / external audit).
- Magnetar: **Tier B** (v0.1 reveal-and-aggregate impl landed). Tier A roadmap targets 2027 Q3.
- BLS / Z-Chain Groth16: production-deployed; existing literature audit.

The Pulsar profile is the current production deployment on Lux primary network's Q-Chain. The Aurora profile is operationally ready and gated on Corona Tier A. The Polaris profile is operationally ready and gated on Magnetar Tier A.

11.4 Path to Polaris

The path from current production (`Pulsar` profile) to maximum-assurance deployment (`Polaris` profile) is:

1. **Corona Tier A closure**. Land EC theory shells, the Lean↔EC bridge, dudect 10^9 samples, and external audit. Target: 2027 Q1.
2. **Aurora profile activation**. Coordinated multi-window rotation from Pulsar to Aurora on a target chain (initial candidate: Z-Chain itself, which has the highest assurance requirements).
3. **Magnetar Tier A closure**. Land the full Tier A artifact bundle for Magnetar; complete the Tier B→Tier A gap analysis in `luxfi/magnetar/BLOCKERS.md`. Target: 2027 Q3.
4. **Polaris profile activation**. Coordinated rotation from Aurora to Polaris on the target chain.

At each step, the cert wire format is stable; only the populated-leg set changes. This is what the umbrella spec was designed to make possible.

11.5 Open questions

Hash agility. The QuasarCert binds all legs to round-digest μ, which is currently SHA3-256. A hash break would require migrating μ to a different hash function. The umbrella spec does not currently specify a hash-agility upgrade path; this is a candidate for a future LP.

Cert size at large n. The Pulsar and Corona threshold signatures are $O(1)$ in n (the validator count) after DKG, but the Z-Chain Groth16 prover time scales linearly in n ($\sim 2^{20}$ constraints per validator). At $n \geq 1000$, the Z-Chain leg's prover cost becomes the bottleneck. The `pq-finality-no-bls.tex` analysis [21] proposes fixed-committee signing ($k = 128$) as a scale-invariant alternative; this is a candidate for the next iteration of the Z-Chain leg.

PQ VRF for committee sampling. The current Photon committee selection uses a VRF based on classical primitives. A fully PQ-safe committee sampler would require a lattice-based VRF (see Boneh-Dai 2024 [6]); this is an open research area.

Strict-PQ profile. A profile that omits the BLS leg entirely is implemented as `pq-finality-no-bls` [21] and is available behind a chain-config flag. The wire format supports an empty σ^{BLS} slot, so the strict-PQ profile is a valid QuasarCert variant. The profile-match clause would need to allow this variant; this is a candidate for a fourth registered profile (working name: "Quanta").

11.6 Closing

QuasarCert is the cryptographic conscience of Lux consensus. Each primitive does one thing, in one place; each profile composes them under one verifier; each cert carries one byte string; each chain selects one profile. The composition is conjunctive — one leg breaks, the cert still stands — and the assumptions are diversified across discrete-log, two lattice families, and hash-based primitives. The wire format is stable; the primitives are swappable; the profiles are independently selectable.

This is what hardness diversification looks like when it is designed orthogonally from day one. No redundant computation, no braided concerns, no implicit cross-leg dependencies. Each primitive is independently complete; the umbrella composes them without owning them. As Hickey would put it: simple, not easy.

The PQ migration is not a single event; it is a multi-year trajectory across multiple primitives, audits, standards, and deployments. QuasarCert is the wire-format contract that makes the trajectory coherent: a Pulsar-profile cert today, an Aurora-profile cert tomorrow, a Polaris-profile cert when Magnetar matures, all carrying the same Go struct, the same verifier, the same Warp envelope, the same Z-Chain proof structure.

The primitives change; the contract is forever.

References

[1] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Journal of Mathematical Cryptology, 2015.

[2] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, 2016.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*. ACM, 1993.

[4] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. SPHINCS+: Submission to the NIST post-quantum project, 2022.

[5] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*. Springer, 2003.

[6] Dan Boneh and Quang Dai. Lattice-based verifiable random functions. IACR ePrint 2024 (working draft), 2024.

[7] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*. Springer, 2018.

[8] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. BLS signatures (`draft-irtf-cfrg-bls-signature-05`). IRTF Internet Draft, 2022.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*. Springer, 2001.

[10] Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. Ringtail: Practical two-round threshold signatures from LWE. Cryptology ePrint Archive, Paper 2024/1113; IEEE Symposium on Security and Privacy (S&P) 2025, 2024. Academic two-round lattice threshold (IACR ePrint 2024/1113, IEEE S&P 2025). Not a Lux artifact; do not relabel as "Corona" or "Pulsar".

[11] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. IACR ePrint 2017/1050, 2017.

[12] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. IACR ePrint 2021/060, 2021.

[13] Andrea Celi, Rafaël del Pino, Thomas Espitau, Guilhem Niot, and Thomas Prest. Efficient threshold ML-DSA. In *USENIX Security*, 2026.

[14] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, volume 2018, pages 238–268, 2018.

[15] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*. Springer, 2005.

[16] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*. Springer, 2015.

[17] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*. Springer, 1999.

[18] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security*, 2021.

[19] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*. Springer, 2016.

[20] Zach Kelling. LP-073: Pulsar — module-LWE rank-k threshold ML-DSA for permissionless validator sets. Technical Report LP-073, Lux Industries, Inc., 2026. Lux's own threshold ML-DSA (FIPS 204) over Module-LWE at general rank k; canonical LP-073. Distinct from Corona (rank 1) and from Ringtail ([10]).

[21] Zach Kelling. Post-quantum finality for lux quasar: Replacing bls aggregation with ml-dsa + corona. Lux Industries; `lux/proofs/pq-finality-no-bls.tex`, 2026.

[22] Zach Kelling. QuasarCert soundness: Canonical formal specification. Lux Industries; `lux/proofs/quasar-cert-soundness.tex`, 2026.

[23] Lux Core Team. luxfi/chains: Z-chain groth16 identity rollup vm. https://github.com/luxfi/chains, 2026. `quantumvm/` subdir; v1.2.3+.

[24] Lux Core Team. luxfi/consensus: Quasar consensus engine. https://github.com/luxfi/consensus, 2026. v1.22.85; `protocol/quasar/` subdir; QuasarCert wire format and verification.

[25] Lux Core Team. luxfi/crypto: Bls / ml-dsa / ml-kem / slh-dsa primitives. https://github.com/luxfi/crypto, 2026.

[26] Lux Core Team. luxfi/threshold: Threshold mpc protocol library. https://github.com/luxfi/threshold, 2026. Includes `protocols/bls`, `protocols/corona`, `protocols/cggmp21`, `protocols/frost`.

[27] Lux Core Team. Quasar: Umbrella specification, profile registry, and composition proofs. https://github.com/luxfi/quasar, 2026.

[28] Lux Industries. Corona: Two-round Ring-LWE threshold signatures. https://github.com/luxfi/corona, 2026. Lux's own threshold scheme: Ring-LWE (= Module-LWE at rank 1), two-round. Builds on the Ringtail line ([10]); cited via this Lux artifact, never via the academic paper. Q-Chain Feldman/Pedersen DKG.

[29] Lux Industries. LP-020: Quasar consensus. https://github.com/luxfi/papers/tree/main/lp-020-quasar-consensus, 2026.

[30] Lux Industries. LP-073: Pulsar dynamic lattice threshold signatures. https://github.com/luxfi/papers/tree/main/lp-073-pulsar, 2026.

[31] Lux Industries. LP-075: Bls aggregate signatures for warp messaging. https://github.com/luxfi/lps/blob/main/LP-075-bls-aggregate.md, 2026.

[32] Lux Industries. LP-137: Gpu-native crypto stack. https://github.com/luxfi/lps/blob/main/LP-137.md, 2026.

[33] Lux Industries. LP-181: Magnetar — threshold SLH-DSA via reveal-and-aggregate. Technical Report LP-181, Lux Industries, Inc., 2026. Lux's threshold SLH-DSA (FIPS 205, hash-based) leg; canonical LP-181.

[34] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6):1–35, 2013.

[35] NIST. FIPS 204: Module-lattice-based digital signature standard. https://csrc.nist.gov/pubs/fips/204/final, 2024.

[36] NIST. FIPS 205: Stateless hash-based digital signature standard. https://csrc.nist.gov/pubs/fips/205/final, 2024.

[37] NIST. NIST IR 8214C: Multi-party threshold cryptography schemes. NIST Internal Report, 2024.

[38] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*. Springer, 1991.

[39] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Journal of the ACM*, volume 56, pages 1–40. ACM, 2009.

[40] Vishnu J. Seesahai and Zach Kelling. LSS: A pragmatic framework for dynamic and resilient threshold signatures with live secret resharing. Lux Industries, Inc., 2026.

[41] Adi Shamir. How to share a secret. In *Communications of the ACM*, volume 22, pages 612–613, 1979.

[42] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*. IEEE, 1994.

Reproducibility

Source code. `github.com/luxfi/quasar` (umbrella spec, profile registry, composition proofs); `github.com/luxfi/consensus` (consensus engine, QuasarCert wire implementation, `protocol/quasar/types.g` canonical); `github.com/luxfi/pulsar` (threshold ML-DSA-65 primitive, v1.0.9); `github.com/luxfi/corona` (threshold Module-LWE primitive, v0.6.0); `github.com/luxfi/magnetar` (threshold SLH-DSA primitive, v0.2.0); `github.com/luxfi/crypto` (BLS / ML-DSA / SLH-DSA primitives); `github.com/luxfi/thr` (threshold MPC protocol library); `github.com/luxfi/chains/quantumvm` (Z-Chain Groth16 rollup VM).
Build. `GOWORK=off go build ./...` and `GOWORK=off go test ./...` in `luxfi/consensus`; per-primitive build and KAT vectors are documented in each repo's `README.md` and `SPEC.md`.
Hardware tested. Apple M1 Max (10 cores, darwin/arm64) for the measurements in Section 8; commodity x86_64 Linux validators for production deployment.
Standards referenced. FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA), IETF `draft-irtf-cfrg-bls-signature`, RFC 9591 (FROST), NIST IR 8214C (MPTC).
Mechanized proofs. Lean proofs at `luxfi/proofs/lean/Consensus/Quasar.lean` (triple-mode composition, BFT quorum intersection, single-round uniqueness). LaTeX proof at `luxfi/proofs/quasar-` (Theorems 7.5–7.8). TLA+ specifications at `luxfi/proofs/tla/`. Tamarin protocol verification at `luxfi/proofs/tamarin/`.
Contact. `security@lux.network`.