Pulsar: Dynamic Lattice Threshold Signatures for Permission-less Validator Sets

Zach Kelling

Lux Industries

2026

Abstract

We present Lux, a hybrid post-quantum consensus architecture for DAG-native chains. Lux separates fast operational finality from durable post-quantum finality. Validators emit Photons over a Nebula DAG substrate; classical BLS aggregates form Beams for low-latency confirmation, while ML-DSA attestations and Pulsar lattice-threshold Pulses provide post-quantum finality over Nebula roots. Prism binds all certificate lanes to the same transcript, and Quasar reaches Horizon finality when the bound lanes verify.

To support dynamic validator sets, Lux introduces LSS, a lifecycle framework for threshold cryptography that manages generations, snapshots, rollback, and activation across independent threshold kernels. Pulsar instantiates LSS with Corona-family lattice threshold signatures under a SHA3 transcript profile; Lens provides the curve-threshold counterpart. Warp 2.0 extends this finality model across chains by carrying Pulse-backed source finality in cross-chain message envelopes. The result is a coherent architecture for sovereign, co-validating, and validity-based chains under a hybrid classical/post-quantum security model.

Honest claim. The crypto primitives in this paper are not new — proactive secret sharing, secret redistribution, verifiable redistribution, two-round lattice threshold signing, and identifiable-abort lattice threshold signing all exist in prior work. *What is new is the composition*: a permissionless-consensus deployment architecture that turns static two-round PQ threshold signatures into a dynamic, leaderless, validator-rotation-tolerant finality layer. The paper specifies that systems layer at a level of detail sufficient to reproduce the Go canonical at `github.com/luxfi/pulsar` byte-for-byte across C++, Metal, CUDA, and WGSL ports.

This paper is the formal companion to LP-073 [37]; the wire-format, lifecycle, and security claims here are normative and supersede earlier Corona-only sketches. The canonical claims/evidence table and the ten architectural commitments live in LP-105 (Lux Stack Lexicon); other Lux documents reference them by section, never duplicate them.

Contents

1 Introduction **5**
- 1.1 What Pulsar is . . . 5
- 1.2 Why MAC-authenticated Round 1 . . . 5
- 1.3 What is novel . . . 5
- 1.4 What this paper specifies . . . 6

2 Parameters and Ring Construction **6**
- 2.1 Rings . . . 6
- 2.2 NTT and Montgomery constants . . . 7
- 2.3 Protocol constants . . . 7
- 2.4 Discrete-Gaussian parameters . . . 7

3 Protocol **8**
- 3.1 Notation . . . 8
- 3.2 Gen (key generation, trusted-dealer or DKG) . . . 8
- 3.3 SignRound1 (offline commitment) . . . 9
- 3.4 SignRound2Preprocess (MAC verify, full-rank check) . . . 10
- 3.5 SignRound2 (message binding, partial signature) . . . 10
- 3.6 SignFinalize (combine into a single signature) . . . 11
- 3.7 Verify . . . 12
- 3.8 Refresh (HJKY97 proactive refresh) . . . 12
- 3.9 Reshare (Desmedt–Jajodia97 redistribution) . . . 13
- 3.10 Activate (the consensus circuit-breaker) . . . 13
- 3.11 Rollback . . . 14
- 3.12 Reanchor (era boundary) . . . 14

4 Wire Format **14**
4.1 Endianness and primitive widths . . . 15
4.2 Polynomial . . . 15
4.3 Vector and Matrix . . . 15
4.4 Signature . . . 15
4.5 Group Key . . . 15
4.6 MAC input . . . 16
4.7 Transcript hashes . . . 16
4.8 Discrete-Gaussian sampler (Ziggurat) . . . 17
4.9 Ternary sampler . . . 17
4.10 Cross-implementation Known-Answer-Test contract . . . 17

5 Security **18**
5.1 Security games . . . 18
5.2 Theorem 5.1: Pulsar SUF-CMA under MLWE / MSIS . . . 18
5.3 Theorem 5.2: Activation-certificate circuit-breaker . . . 19
5.4 Theorem 5.3: Resharing preserves master secret . . . 20
5.5 Theorem 5.4: Lineage unforgeability . . . 20
5.6 Concrete parameter analysis . . . 20
5.7 Side-channel posture . . . 21
5.8 Nonce-uniqueness invariant . . . 21
5.9 Identifiable abort posture . . . 22
5.10 Composition with Quasar Horizon . . . 22

6 Formal Model **22**
6.1 Validator set, key era, and persistent group key . . . 22
6.2 KeyEraID lineage as a chain of tuples . . . 23
6.3 Adversary models . . . 23
6.4 Static vs adaptive corruption . . . 23
6.5 Soundness and liveness goals . . . 24
6.6 What this model omits . . . 25

7 Implementation **25**
7.1 Go canonical . . . 25
7.2 Key-era lifecycle . . . 26
7.3 LSS-Pulsar adapter . . . 26
7.4 Quasar consensus integration . . . 26
7.5 Warp 2.0 cross-chain envelope . . . 27
7.6 C++ port . . . 27
7.7 GPU kernels . . . 27
7.8 Test oracle . . . 28
7.9 Performance . . . 28

8 Related Work **28**
8.1 Lattice-based threshold signatures . . . 28
8.2 Schnorr-family / curve-side threshold signatures . . . 29
8.3 ECDSA and CGGMP21 . . . 30
8.4 BLS aggregate signatures . . . 30
8.5 Hash-based threshold signatures . . . 30
8.6 Classical lifecycle layer . . . 30
8.7 Standards landscape . . . 31
8.8 Comparison table . . . 31

8.9 Where Pulsar's contribution lives . . . 31

9 Pulsar Key-Era Lifecycle **32**
9.1 Three lineage fields, kept distinct . . . 32
9.2 Three layers, one shipping path . . . 33
9.3 Refresh vs Reshare: distinct primitives . . . 33
9.4 Activation certificate: the consensus circuit-breaker . . . 33
9.5 Rollback and the no-slashing failure ladder . . . 34
9.6 Pedersen DKG over R_q: the trusted-dealer-free genesis path . . . 34
9.7 What is preserved, what changes . . . 35

10 Pedersen-Style DKG over R_q (`dkg2`) **35**
10.1 The broken Feldman commit . . . 35
10.2 The Pedersen construction . . . 36
10.3 Hiding (computational, MLWE on $\mathbf{B}$) . . . 37
10.4 Binding (computational, MSIS on $[\mathbf{A} \mid \mathbf{B}]$) . . . 37
10.5 Pedersen identity (correctness across Lagrange recombination) . . . 38
10.6 Comparison to broken Feldman, GKS24, KRT24, Threshold Raccoon . . . 38
10.7 Concrete parameters . . . 39
10.8 Identifiable abort and the complaint workflow . . . 39
10.9 Mapping to Pulsar Sign . . . 40
10.10 Implementation surface . . . 40
10.11 Lean theorem index . . . 41

11 The LSS-Pulsar Adapter **41**
11.1 Bootstrap Dealer vs Signature Coordinator . . . 42
11.2 Adapter contract guarantees . . . 42
11.3 What the adapter does *not* inherit . . . 42
11.4 Acceptance tests . . . 43

12 Grouped Pulsar Certificates **43**
12.1 Construction . . . 43
12.2 What this buys . . . 43
12.3 Open systems-research questions . . . 43
12.4 Comparison to Threshold Raccoon and Hermine . . . 44

13 Pulsar in Quasar Horizon **44**
13.1 The vocabulary stack . . . 44
13.2 Why three lanes (and not one) . . . 45
13.3 The Horizon certificate shape (Prism-bound) . . . 45
13.4 Asynchronous PQ finality . . . 46
13.5 Cert-tier degradation: measured 2026-06-03 . . . 46
13.6 Where the systems contribution lives . . . 47

1 Introduction

Threshold signatures replace a single signing key with a (t, n) secret sharing of that key, requiring t of n parties to cooperate before a signature is produced. For a blockchain consensus pipeline, the relevant question is not only *can* t parties produce a signature, but *how often* the protocol must run, *how many bytes* cross the wire per round, and *what survives* when an attacker captures any subset of $t-1$ key shares or acquires a quantum oracle for ECDSA.

Lux Quasar 3.0 [33] answers all three questions at once: the consensus engine signs every block with three independent threshold signatures in parallel. BLS12-381 [38] occupies the classical fast-path lane (48-byte aggregate, sub-second). ML-DSA-65 [35] provides FIPS-compliant per-validator identity proofs (3.3 KB each). The third lane, Pulsar, is the post-quantum threshold lane: a single ∼33 KB aggregate that is Module-LWE-secure under both classical and quantum adversaries, recovers liveness when a quantum oracle invalidates BLS, and adds no Paillier-style range proofs because shares aggregate *linearly*.

1.1 What Pulsar is

Pulsar is a two-round threshold signature over the cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^\varphi + 1)$ with $\varphi = 256$ and a 48-bit NTT-friendly prime q. Round 1 is offline: each party samples a Discrete-Gaussian masking matrix $\hat{R}$, broadcasts a commitment $D_i = A \cdot \hat{R}_i + \hat{E}_i$, and authenticates the broadcast with pairwise BLAKE3-keyed MACs. Round 2 binds the message: the parties hash a Discrete-Gaussian challenge vector u from the Round 1 transcript, derive a low-norm ternary challenge c, and open partial signatures z_i that aggregate linearly. A verifier reconstructs $Az - bc$ in the rounded-coefficient hint domain R_ν and accepts iff the per-coefficient Δ correction stays below an L_2 bound.

The protocol mirrors the structure of FROST [29] (linear, non-interactive after Round 1) but operates on a lattice rather than an elliptic curve. The closest analog in the Module-LWE family is two-round threshold ML-DSA [19, 1]: Pulsar differs in the rounding/hint construction (a per-coefficient correction Δ supersedes ML-DSA's (α, β)-bounded hint), in the use of pairwise MACs for Round 1 authentication [18], and in the optimized $t = K$ Shamir path [39] that avoids per-coefficient polynomial evaluation.

1.2 Why MAC-authenticated Round 1

Without MAC authentication, an attacker who controls the network can replace a victim's D_i with a malicious D_i' and force the protocol into a transcript that accepts a forged z. Drijvers, Edalatnejad, Ford, Neven, and Stadler [18] showed that any threshold signature with concurrent execution and unauthenticated commitment broadcast is vulnerable to a sub-exponential attack on the underlying secret. The classical defense is non-interactive zero-knowledge proofs of well-formedness on D_i; these are too expensive for a 3-second consensus interval at K=21 validators.

Pulsar uses pairwise symmetric MAC keys (one 32-byte key per ordered pair (i, j), distributed at key generation alongside the Shamir shares) and a 256-bit BLAKE3 [51] MAC over D_i concatenated with the session id and active set. Verification cost is one BLAKE3 hash per pair, dominated by the D_i payload size. The asymmetric "identity in slot 0" construction (the generator embeds its own party id, the verifier embeds the recipient's id) prevents reflection attacks where a captured $\text{MAC}_{i \to j}$ is replayed as $\text{MAC}_{j \to i}$.

1.3 What is novel

Two-round MAC-authenticated lattice threshold signing is well-established [7, 16, 52]. Proactive secret sharing [23] and secret redistribution to new access structures [17] are even older. What this paper specifies, and what `github.com/luxfi/pulsar` ships, is the *composition* that makes these primitives operate in a permissionless consensus setting:

1. A key-era lifecycle (Section 9) that distinguishes the persistent group public key $(\mathbf{A}, \tilde{\mathbf{b}})$ from the rotating share distribution $\{\mathbf{s}_i\}$, so validator-set rotations preserve the master secret structurally without a per-epoch trusted dealer.
2. An activation certificate (§9.4) that gates each generation transition on a threshold signature verifying under the unchanged group public key. Activation proves new signing capability; complaint and slashing-evidence pipelines are independent.
3. An LSS-Pulsar adapter (Section 11) that plugs into Lux's universal MPC lifecycle framework [53, 39], sharing the orchestration with curve-side schemes (Lens / FROST, LP-103 [44]) and ECDSA (CMP).
4. A grouped Pulsar deployment shape (Section 12) for large validator sets, with per-group resharing and a quorum-of-groups Horizon certificate.
5. A Prism-bound Horizon certificate (Section 13) composing BLS Beam, ML-DSA cert set, and Pulsar Pulse against a shared source-chain transcript (LP-105 [45]).
6. A Warp 2.0 cross-chain envelope (§7, `warp/pulsar/`) carrying the same Prism-bound shape over the Lux cross-chain messaging protocol, with forward / backward compatibility against shipping Warp 1.x receivers.

The crypto exists. The systems layer that turns it into permissionless PQ-threshold consensus is the contribution.

1.4 What this paper specifies

This paper is the byte-exact specification of the math kernel plus the systems composition. Section 2 pins every constant to its derivation in the Lattigo `v7` fork [31]. Section 3 documents the two-round protocol plus finalize and verify. Section 4 is the normative wire format: a C++ implementer reading this section alone should produce serialized output that hashes byte-equal to the Go canonical's output without consulting the Go source. Section 5 states the MLWE assumption, the rejection-sampling slack η_ε, and the identifiable-abort posture. Section 9 specifies the key-era lifecycle (Bootstrap, Refresh, Reshare, Reanchor) plus the activation certificate. Section 11 specifies the LSS-Pulsar adapter contract. Section 12 specifies grouped Pulsar certificates. Section 13 composes Pulsar with BLS Beam and ML-DSA cert sets into the Prism-bound Horizon certificate. Section 7 surveys the shipping Go canonical, the C++ port [40], and the GPU-kernel research workstream. Section 8 surveys the post-quantum threshold-signature literature and locates Pulsar within it.

The companion artifacts are LP-073 [37] (this LP) and LP-105 [45] (vocabulary). Where this paper and LP-073 conflict, the Go canonical at commit `HEAD` of `github.com/luxfi/pulsar` is the tiebreaker.

2 Parameters and Ring Construction

2.1 Rings

The protocol uses three rings, all of degree $\varphi = 256$ and Standard NTT [34] (negacyclic $\mathbb{Z}[X]/(X^\varphi + 1)$):

Table 1: Ring parameters. R is the working ring; R_ξ and R_ν are non-NTT auxiliary rings used solely as containers for rounded coefficients.

Ring	Degree	Modulus	Purpose
R	256	$q = \texttt{0x1000000004A01}$ (48-bit)	working ring
R_ξ	256	$q_\xi = \texttt{0x40000} = 2^{18}$	rounded public key $\tilde{b}$
R_ν	256	$q_\nu = \texttt{0x80000} = 2^{19}$	hint $\tilde{h}$, $\tilde{\Delta}$

The construction R requires $q \equiv 1 \pmod{2\varphi}$ for the existence of a 2φ-th primitive root of unity; this holds since $q \mod 512 = 1$. The rings are instantiated as

$$\texttt{ring.NewRing}(2^8,\ \{q\}) \quad \text{etc.}$$

in the Go canonical (`threshold/threshold.go:46`).

2.2 NTT and Montgomery constants

For each modulus q_*, Lattigo [31] pre-computes:

- $\text{MRedConstant}(q) = q^{-1} \bmod 2^{64}$, used in Montgomery reduction (`lattice/ring/modular_reduction`
- $\text{BRedConstant}(q) = (\lfloor 2^{128}/q \rfloor \gg 64,\ \lfloor 2^{128}/q \rfloor \bmod 2^{64})$, used in Barrett reduction (line 99).
- RootsForward and RootsBackward tables: powers of the 2φ-th primitive root in Montgomery form, bit-reversed order (`lattice/ring/subring.go:99`).

All NTT-domain multiplies in the protocol use `MulCoeffsMontgomery` (the lazy variant when output bounds permit). Polynomials are stored in Montgomery form ($a \cdot 2^{64} \bmod q$) inside the NTT domain; `MForm` and `IMForm` interconvert.

2.3 Protocol constants

Constants from `sign/config.go` are reproduced in Table 2.

Table 2: Protocol constants. `LogN` $= 8$ defines the ring degree φ; the symbol `N` in the Go source denotes the secret-vector length, which we render as n_{vec} to avoid confusion.

Symbol	Value	Role
φ	256	ring degree (`1 << LogN`)
m_{pk}	8	public-key matrix rows (M in Go)
n_{vec}	7	secret-vector length (`N` in Go)
$\bar{D}$	48	masking nonce expansion factor
K	n	total parties (set per epoch)
t	threshold	$1 \le t < n$
κ	23	challenge ternary Hamming weight
ξ	30	public-key rounding $\tilde{b} = \lfloor b/2^{30} \rceil$
ν	29	hint rounding $\tilde{h} = \lfloor h/2^{29} \rceil$
$\lvert\text{Key}\rvert$	32 bytes	symmetric / MAC / PRNG seed length
η_ε	2.650104	rejection-sampling slack

2.4 Discrete-Gaussian parameters

Three Discrete-Gaussian distributions appear in the protocol; bound $B = 2\sigma$ is the truncation bound passed to Lattigo's Ziggurat sampler [46, 31].

Table 3: Discrete-Gaussian samplers. All bounds are $B = 2\sigma$.

Sampler	σ	Used for
σ_E	6.108187070284607	LWE error e, per-party E_i, mask R_i
σ_*	1.7285×10^{11} ($\approx 2^{37.33}$)	"star" column r^*, e^*
σ_U	1.6396×10^8	challenge-side u (GaussianHash)

The L_2 acceptance bound is

$$B^2 = 184\,960\,669\,042\,442\,604\,975\,662\,780\,477,$$

equivalent to $B \approx 4.30 \times 10^{14}$. Every coefficient of z and Δ is reduced to its signed representative in $(-q/2, q/2]$ before squaring (`sign.go:303`).

3 Protocol

The protocol is a 2-round distributed signing scheme on top of a (t, n) Shamir setup over R_q, augmented with the Pulsar lifecycle operations (Refresh, Reshare, Activation, Reanchor). We give every algorithm in mathematical pseudocode with explicit input / output types and explicit $\mathbb{Z}_q$ / R_q operations. Line numbers reference the Go canonical at `github.com/luxfi/pulsar` (Pulsar fork) and `github.com/luxfi/corona` (per-signature kernel) [26, 42].

3.1 Notation

- $R = \mathbb{Z}[X]/(X^\varphi + 1)$ with $\varphi = 256$; $R_q = R/qR$ with q the 48-bit prime of §2.
- $R_\xi = \mathbb{Z}[X]/(X^\varphi + 1)$ reduced mod 2^{18}; R_ν similarly mod 2^{19}.
- Polynomial arithmetic in R_q is performed in *NTT-Montgomery* form: every $a(X) \in R_q$ is stored as $\hat{a} = (a \cdot 2^{64} \bmod q)$ with NTT applied. Multiplication is coefficient-wise $(\hat{a}\hat{b}) \bmod q$ with Montgomery reduction.
- $\mathbf{A} \in R_q^{m_{\mathrm{pk}} \times n_{\mathrm{vec}}}$, $m_{\mathrm{pk}} = 8$, $n_{\mathrm{vec}} = 7$, $\bar{D} = 48$.
- $\mathcal{D}(\sigma)$ is the discrete-Gaussian distribution over R with width σ, sampled per-coefficient via Lattigo Ziggurat.
- NTT, INTT, MForm, IMForm denote forward/inverse NTT and Montgomery form / inverse Montgomery form.
- WriteTo is the LP-073 §4 wire serialisation.
- $\odot$ is element-wise polynomial multiplication on a vector.
- $\mathbf{1}$ is the constant polynomial $1 \in R_q$ in NTT-Montgomery form (i.e., MForm(NTT(X^0))).

3.2 Gen (key generation, trusted-dealer or DKG)

Gen (`sign/sign.go:48`) takes a 32-byte trusted-dealer seed k_{TD} and a list of pre-computed Lagrange coefficients λ_i for the active set $T = \{0, \ldots, K-1\}$ (`primitives/shamir.go:122`).

Inputs: $k_{\mathrm{TD}} \in \{0,1\}^{256}$, $T = \{0, \ldots, K-1\}$, threshold $t \le K$, Lagrange coefficients $\boldsymbol{\lambda} = (\lambda_0, \ldots, \lambda_{K-1}) \in R_q^K$.

Outputs: Group key $\mathsf{GK} = (\mathbf{A}, \tilde{\mathbf{b}}) \in R_q^{m_{\mathrm{pk}} \times n_{\mathrm{vec}}} \times R_\xi^{m_{\mathrm{pk}}}$; per-party share $\mathrm{sk}_i \in R_q^{n_{\mathrm{vec}}}$; pairwise PRF seeds $\mathsf{seeds} \in (\{0,1\}^{256})^{K \times K}$; pairwise MAC keys $\mathsf{macKeys} \in (\{0,1\}^{256})^{K \times K}$ symmetric.

Steps:

1. $\mathbf{A} \leftarrow$ UniformPolyMatrix$(R_q, m_{\mathrm{pk}}, n_{\mathrm{vec}})$ in NTT-Montgomery form, sampled coefficient-wise from $\mathbb{Z}_q$ via BLAKE3-XOF over k_{TD}.
2. Pre-compute deterministic random byte budget by BLAKE3-XOF over k_{TD} [51] (`utils/utils.go:444`):

$$\textsc{precomp} = K^2 \cdot 32 + \varphi \cdot n_{\mathrm{vec}} \cdot (K-1) \cdot \lceil \log_2(q)/8 \rceil + K(K-1) \cdot 32.$$

3. Seed BLAKE2 keyed PRNG [2] from k_{TD}.
4. $\mathbf{s} \leftarrow \mathcal{D}(\sigma_E)^{n_{\mathrm{vec}}}$ sampled per-coefficient in standard form.

5. Run optimised Shamir secret-sharing ([39]; the $t = K$ path samples $K - 1$ uniform shares per coefficient and computes the last to satisfy the Lagrange constraint):

$$\mathrm{sk}_i \leftarrow \textsc{ShamirSecretSharing}(R_q, \mathbf{s}, K, \boldsymbol{\lambda}).$$

   For each coefficient slot $k \in [0, \varphi)$, the share polynomial $f_k(X) \in \mathbb{Z}_q[X]$ has degree $\leq t - 1$ with $f_k(0) = s_k$ and $f_k(\alpha_i) = (\mathrm{sk}_i)_k$.
6. Convert sk_i and $\mathbf{s}$ to NTT-Montgomery form.
7. $\mathbf{e} \leftarrow \mathcal{D}(\sigma_E)^{m_{\mathrm{pk}}}$ in NTT-Montgomery form.
8. Compute the LWE relation in $R_q^{m_{\mathrm{pk}}}$:

$$\mathbf{b} \;=\; \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \mod q,$$

   output in NTT-Montgomery form.
9. Convert $\mathbf{b}$ to standard form, then compress to R_ξ by per-coefficient rounding:

$$\tilde{b}_k = \left\lfloor (b_k + 2^{\xi-1})/2^{\xi} \right\rfloor \quad \forall k \in [0, \varphi).$$

   The result $\tilde{\mathbf{b}} \in R_\xi^{m_{\mathrm{pk}}}$ is the rounded public-key vector.
10. For every ordered pair (i, j), sample $\mathsf{seeds}[i][j] \in \{0,1\}^{256}$ uniformly.
11. For every unordered pair $\{i, j\}$, sample $\mathsf{macKeys}[i][j] = \mathsf{macKeys}[j][i]$ uniformly.
12. Erase $\mathbf{e}$ and $\mathbf{s}$ (only shares sk_i survive after dealer state is destroyed).

Public output: $\mathsf{GK} = (\mathbf{A}, \tilde{\mathbf{b}})$.

Per-party output: $(\mathrm{sk}_i, \mathsf{seeds}[i], \mathsf{macKeys}[i], \lambda_i)$ for $i \in T$.

3.3 SignRound1 (offline commitment)

Each party i, on input $(\mathbf{A}, sid, k_{\mathrm{PRF}}, T)$ (`sign.go:99`), produces $D_i \in R_q^{m_{\mathrm{pk}} \times (\bar{D}+1)}$ and $\{\mathrm{MAC}_{i \to j}\}_{j \in T \setminus \{i\}}$.

Inputs: $\mathbf{A}$, session id $sid \in \mathbb{Z}_{2^{64}}$, PRF key $k_{\mathrm{PRF}} \in \{0,1\}^{256}$, active set $T \subseteq \{0, \ldots, n-1\}$ with $|T| = K$.

Outputs: Commitment matrix $D_i \in R_q^{m_{\mathrm{pk}} \times (\bar{D}+1)}$ in NTT-Montgomery form; pairwise MAC values $\mathrm{MAC}_{i \to j} \in \{0,1\}^{256}$ for each $j \in T \setminus \{i\}$.

Steps:
1. Derive PRNG key from share: $k_{\mathrm{sk}} \leftarrow \mathrm{BLAKE3}(\textsc{WriteTo}(\mathrm{sk}_i))[:32]$.
2. Seed BLAKE2 keyed PRNG from k_{sk}.
3. Sample masking nonces (per-party):

$$\begin{aligned} r^* &\leftarrow \mathcal{D}(\sigma_*)^{n_{\mathrm{vec}}}, \\ e^* &\leftarrow \mathcal{D}(\sigma_*)^{m_{\mathrm{pk}}}, \\ R_i &\leftarrow \mathcal{D}(\sigma_E)^{n_{\mathrm{vec}} \times \bar{D}}, \\ E_i &\leftarrow \mathcal{D}(\sigma_E)^{m_{\mathrm{pk}} \times \bar{D}}, \end{aligned}$$

   all in NTT-Montgomery form. The "star" column carries $\eta_\varepsilon \cdot \sigma_*$ slack reservation for the L_2 acceptance bound.

4. Concatenate the star column with R_i to form the augmented mask:

$$\hat{R}_i = [r^* \mid R_i] \in R_q^{n_{\text{vec}} \times (\bar{D}+1)}, \qquad \hat{E}_i = [e^* \mid E_i] \in R_q^{m_{\text{pk}} \times (\bar{D}+1)}.$$

5. Compute the commitment:

$$D_i \;=\; \mathbf{A} \cdot \hat{R}_i + \hat{E}_i \mod q \quad \in \quad R_q^{m_{\text{pk}} \times (\bar{D}+1)},$$

   in NTT-Montgomery form.
6. For each $j \in T \setminus \{i\}$:

$$\text{MAC}_{i \to j} \;=\; \textsc{GenerateMAC}(D_i,\ \mathsf{macKeys}[i][j],\ i,\ \mathit{sid},\ T,\ j,\ \mathit{verify}{=}\perp).$$

   GenerateMAC (`primitives/hash.go:34`) emits BLAKE3 over the byte stream specified in §4 truncated to 256 bits.

Per-party output broadcast: $(D_i, \{\text{MAC}_{i \to j}\}_{j \in T \setminus \{i\}})$.

3.4 SignRound2Preprocess (MAC verify, full-rank check)

After collecting $\{D_j, \text{MAC}_{j \to i}\}_{j \in T}$ from all signers, party i runs (`sign.go:147`):

Inputs: $\{D_j\}_{j \in T}$, $\{\text{MAC}_{j \to i}\}_{j \in T \setminus \{i\}}$, $\mathbf{A}$, $\tilde{\mathbf{b}}$, sid, T.

Outputs: Aggregate commitment $D_\Sigma \in R_q^{m_{\text{pk}} \times (\bar{D}+1)}$ (with column 0 retained for the message-binding step) and transcript hash $h_{\text{tx}} \in \{0,1\}^{256}$. Aborts if any check fails.

Steps:

1. Compute the transcript hash:

$$h_{\text{tx}} \;=\; \text{BLAKE3}\big(\textsc{WriteTo}(\mathbf{A}) \,\|\, \textsc{WriteTo}(\tilde{\mathbf{b}}) \,\|\, \mathit{sid}_{\text{BE}} \,\|\, |T|_{\text{BE}} \,\|\, T_{\text{BE}} \,\|\, \textsc{WriteTo}(D_0) \,\|\, \cdots \,\|\, \textsc{WriteTo}(D$$

2. For each $j \neq i$: recompute $\text{MAC}^{\text{exp}}_{j \to i}$ with $\mathit{verify} = \top$ and reject the round if it differs from the received MAC. The asymmetric "identity in slot 0" (sender on generate, recipient on verify) prevents reflection.
3. Aggregate: $D_\Sigma = \sum_{j \in T} D_j$ in NTT-Montgomery form.
4. Drop column 0 of D_Σ (the r^* aggregate). For each coefficient slot $k \in [0, \varphi)$, run Gaussian elimination on the resulting $m_{\text{pk}} \times \bar{D}$ matrix over $\mathbb{F}_q$ (`sign.go:348`):

$$\text{rank}\big(D_\Sigma[1 : \bar{D}+1]_k\big) = m_{\text{pk}} \quad \forall k \in [0, \varphi).$$

   If any slot is rank-deficient, abort.

3.5 SignRound2 (message binding, partial signature)

Party i now binds the message $\mu \in \{0,1\}^*$ (`sign.go:173`).

Inputs: h_{tx}, μ, D_Σ, sk_i, $\mathsf{seeds}[i]$, k_{PRF}, λ_i, $\mathbf{A}$, $\tilde{\mathbf{b}}$.

Outputs: Partial signature $z_i \in R_q^{n_{\text{vec}}}$ in NTT-Montgomery form.

Steps:

1. Sample challenge-side u':

$$u' \leftarrow \textsc{GaussianHash}(R_q, h_{\text{tx}}, \mu, \sigma_U, B_U, \bar{D}) \quad \in \quad R_q^{\bar{D}},$$

   a length-$\bar{D}$ Discrete-Gaussian vector seeded from BLAKE3($h_{\text{tx}} \,\|\, \mu$).
2. Prepend the constant $\mathbf{1}$: $u = [\mathbf{1} \,|\, u'] \in R_q^{\bar{D}+1}$, NTT-Montgomery.
3. Compute the raw message-binding hint:

$$h_{\text{raw}} \;=\; D_\Sigma \cdot u \quad \in \quad R_q^{m_{\text{pk}}},$$

   NTT-Montgomery, then INTT + IMForm to standard form.
4. Compress to R_ν:

$$\tilde{h}_k \;=\; \left\lfloor (h_{\text{raw},k} + 2^{\nu-1})/2^{\nu} \right\rfloor \quad \forall k.$$

5. Derive the low-norm ternary challenge:

$$c \;=\; \textsc{LowNormHash}(R_q, \mathbf{A}, \tilde{\mathbf{b}}, \tilde{\mathbf{h}}, \mu, \kappa) \quad \in \quad R_q,$$

   a Hamming-weight-κ ternary polynomial in NTT-Montgomery form, seeded from BLAKE3(WriteTo($\mathbf{A}$)
6. Compute the symmetric pairwise mask sums:

$$\textsc{mask} \;=\; \sum_{j \in T} \text{PRF}(R_q, \mathsf{seeds}[i][j], k_{\text{PRF}}, \mu, h_{\text{tx}}, n_{\text{vec}}),$$
$$\textsc{mask}' \;=\; \sum_{j \in T} \text{PRF}(R_q, \mathsf{seeds}[j][i], k_{\text{PRF}}, \mu, h_{\text{tx}}, n_{\text{vec}}),$$

   both in NTT-Montgomery form.
7. Compute the partial signature:

$$z_i \;=\; \hat{R}_i \cdot u + \textsc{mask}' + (\text{sk}_i \odot \lambda_i \odot c) - \textsc{mask} \quad \in \quad R_q^{n_{\text{vec}}},$$

   NTT-Montgomery. ($\odot$ is element-wise polynomial multiplication; the dot product over $\hat{R}_i \cdot u$ is the standard matrix-vector multiply.)

Cancellation invariant. The asymmetric mask term cancels across honest parties because the seed $\mathsf{seeds}[i][j] = \mathsf{seeds}[j][i]$ is symmetric. Hence $\sum_{i \in T}(\textsc{mask}'_i - \textsc{mask}_i) = 0$ over a honest aggregate, and only the $\text{sk}_i \odot \lambda_i \odot c$ contribution survives the sum (Lagrange-recombines to $\mathbf{s} \odot c$ at 0 via the Shamir identity).

3.6 SignFinalize (combine into a single signature)

The designated combiner CombinerID $\in T$ collects $\{z_j\}_{j \in T}$ and produces the final signature $\sigma = (c, z, \Delta)$ (`sign.go:234`).

Inputs: $\{z_j\}_{j \in T}$, $\mathbf{A}$, $\tilde{\mathbf{b}}$, c, $\tilde{\mathbf{h}}$.

Outputs: Signature $\sigma = (c, z, \Delta) \in R_q \times R_q^{n_{\text{vec}}} \times R_\nu^{m_{\text{pk}}}$.

Steps:

1. Aggregate partials: $z = \sum_{j \in T} z_j \in R_q^{n_{\text{vec}}}$, NTT-Montgomery.
2. Decompress public key: $\mathbf{b} \leftarrow \textsc{RestoreVector}(R_q, R_\xi, \tilde{\mathbf{b}}, \xi)$ via per-coefficient $b_k = \tilde{b}_k \cdot 2^\xi$. Convert to NTT-Montgomery form.
3. Compute the verifier reconstruction:

$$Az \;=\; \mathbf{A} \cdot z, \qquad Az\_bc \;=\; Az - \mathbf{b} \cdot c \quad \in \quad R_q^{m_{\text{pk}}},$$

   NTT-Montgomery, then INTT + IMForm to standard form.
4. Compress to R_ν: $\widetilde{Az\_bc} \leftarrow \textsc{RoundVector}(R_q, R_\nu, Az\_bc, \nu)$.
5. Compute the per-coefficient correction $\Delta \in R_\nu^{m_{\text{pk}}}$:

$$\Delta_k \;=\; \tilde{h}_k - \widetilde{Az\_bc}_k \quad \in \quad R_\nu \text{ (raw coefficients, no NTT).}$$

Output: $\sigma = (c, z, \Delta)$ where $c \in R_q$ (NTT-Montgomery), $z \in R_q^{n_{\text{vec}}}$ (NTT-Montgomery), and $\Delta \in R_\nu^{m_{\text{pk}}}$ (raw coefficients).

3.7 Verify

A standalone verifier with $(\mathbf{A}, \tilde{\mathbf{b}})$ accepts $\sigma = (c, z, \Delta)$ for μ iff (`sign.go:268`):

1. Recompute $\mathbf{b} \leftarrow \textsc{RestoreVector}(R_q, R_\xi, \tilde{\mathbf{b}}, \xi)$, NTT-Montgomery.
2. $Az \leftarrow \mathbf{A} \cdot z$ on a copy of z.
3. $Az\_bc \leftarrow Az - \mathbf{b} \cdot c$, NTT-Montgomery, then INTT + IMForm.
4. $\widetilde{Az\_bc} \leftarrow \textsc{RoundVector}(R_q, R_\nu, Az\_bc, \nu)$.
5. $h_{\text{rec}} \leftarrow \widetilde{Az\_bc} + \Delta$ in R_ν.
6. $c' \leftarrow \textsc{LowNormHash}(R_q, \mathbf{A}, \tilde{\mathbf{b}}, h_{\text{rec}}, \mu, \kappa)$. If $c' \neq c$ in NTT-Montgomery representation, reject.
7. $\Delta_{\text{full}} \leftarrow \textsc{RestoreVector}(R_q, R_\nu, \Delta, \nu)$ in R via ν-bit shift.
8. Convert z to standard form and compute the squared L_2 norm over signed coefficients in $(-q/2, q/2]$:

$$\sum_k (\Delta_k^{\text{signed}})^2 + \sum_k (z_k^{\text{signed}})^2 \;\leq\; B^2,$$

   $B^2 = 184\,960\,669\,042\,442\,604\,975\,662\,780\,477$ [19] (`sign.go:303`).

Hint Δ. Rounding $\mathbf{A}z - \mathbf{b}c$ at scale 2^ν does not exactly recover $\tilde{\mathbf{h}}$ because of carry propagation across the rounding boundary. Δ corrects this per-coefficient; the L_2 bound prevents a forger from inflating Δ to satisfy the equation with an arbitrary c.

3.8 Refresh (HJKY97 proactive refresh)

Same committee, same threshold, defends against mobile-adversary share accumulation across time [23].

Inputs: Per-party share $(\text{sk}_i)_{i \in V}$, threshold t, randomness oracle.

Outputs: Refreshed shares $(\text{sk}'_i)_{i \in V}$ with the same constant term $\mathbf{s}$ at slot 0.

Steps (each party i):

1. Sample a degree-$(t-1)$ *zero-polynomial* $z_i(X) \in R_q[X]$ with $z_i(0) = 0$ (per coefficient slot, sample $t-1$ random $\mathbb{Z}_q$ coefficients and constrain slot 0 to zero).
2. For each $j \in V$, compute $z_i(\alpha_j) \in R_q^{n_{\text{vec}}}$ and broadcast a lattice Pedersen commit $\mathbf{C}_{i,j} = \mathbf{A} \cdot \text{NTT}(z_i(\alpha_j)) + \mathbf{B} \cdot \text{NTT}(r_{i,j})$ on the contribution with auxiliary mask $r_{i,j} \leftarrow \mathcal{D}(\sigma_E)$.
3. Each party j aggregates incoming contributions: $\text{sk}'_j = \text{sk}_j + \sum_{i \in V} z_i(\alpha_j)$.
4. Verify each contribution against its commit (binding under MSIS on $[\mathbf{A} \mid \mathbf{B}]$, hiding under MLWE on $\mathbf{B}$ [54]); abort and run complaint protocol if any commit fails to open.

Invariant. The refreshed sharing polynomial $f'(X) = f(X) + \sum_i z_i(X)$ has $f'(0) = f(0) = \mathbf{s}$ because each $z_i(0) = 0$. Hence $\mathbf{A}, \tilde{\mathbf{b}}$ are unchanged (Theorem 3, Refresh case).

3.9 Reshare (Desmedt–Jajodia97 redistribution)

New committee V' with new threshold t', $V \cap V'$ may be empty.

Inputs: Old share set $(\text{sk}_i)_{i \in V}$, old threshold t, new committee $V' \subseteq \mathcal{V}$, new threshold $t' \leq |V'|$, qualified subset $Q \subseteq V$ with $|Q| \geq t$.

Outputs: New per-party shares $(\text{sk}'_j)_{j \in V'}$ that recombine to the same $\mathbf{s}$.

Steps:

1. Each $i \in Q$ samples a degree-$(t'-1)$ polynomial $g_i(X) \in R_q[X]^{n_{\text{vec}}}$ with $g_i(0) = \text{sk}_i$ (per coefficient slot, sample $t'-1$ random $\mathbb{Z}_q$ coefficients and constrain slot 0 to the share).
2. For each $j \in V'$, i computes $g_i(\beta_j) \in R_q^{n_{\text{vec}}}$ where β_j is the new evaluation point for j, and broadcasts a lattice Pedersen commit $\mathbf{C}_{i,j} = \mathbf{A} \cdot \text{NTT}(g_i(\beta_j)) + \mathbf{B} \cdot \text{NTT}(r_{i,j})$ on the contribution.
3. Compute new Lagrange coefficients $\lambda_i^Q(0) \in R_q$ for the qualified subset Q at 0 (`primitives.ComputeLagran`
4. Each new party j aggregates contributions weighted by Lagrange coefficients of Q:

$$\text{sk}'_j \;=\; \sum_{i \in Q} \lambda_i^Q(0) \cdot g_i(\beta_j) \quad \in \quad R_q^{n_{\text{vec}}}.$$

5. Verify each commit (binding/hiding as in Refresh).
6. Regenerate pairwise PRF seeds and MAC keys for V' via authenticated key exchange (X25519 baseline; ML-KEM-768 hybrid [47] for PQ-aware deployments).

Invariant. The new sharing polynomial $G(X) = \sum_{i \in Q} \lambda_i^Q(0) \cdot g_i(X)$ satisfies $G(0) = \sum_{i \in Q} \lambda_i^Q(0) \cdot \text{sk}_i = f(0) = \mathbf{s}$ by Lagrange identity (1). Hence $\mathbf{A}, \tilde{\mathbf{b}}$ are unchanged (Theorem 3, Reshare case).

3.10 Activate (the consensus circuit-breaker)

After Reshare or Refresh produces a candidate post-state Π' at generation $g+1$, the new committee threshold-signs an activation message under the unchanged GroupKey to attest that the new shares can sign.

Inputs: New share set $(\text{sk}'_j)_{j \in V'}$, lifecycle context $(\eta, g{+}1, \text{old\_set\_hash}, \text{new\_set\_hash}, \text{old\_}t, \text{new\_}t, \mathsf{GKH},$

Outputs: Activation signature $\sigma_{\text{act}} \in R_q \times R_q^{n_{\text{vec}}} \times R_\nu^{m_{\text{pk}}}$ that verifies under $(\mathbf{A}, \tilde{\mathbf{b}})$.

Steps:

1. Construct the activation message μ_{act} as the byte concatenation

```
"QUASAR-PULSAR-ACTIVATE-v1" ||
   chain_id || network_id ||
   eta || group_id ||
   old_epoch || new_epoch ||
   old_set_hash || new_set_hash ||
   old_t || new_t ||
   GKH || transcript_hash ||
   pairwise_commit || impl_version
```

2. Run SignRound1 (§3.3), SignRound2Preprocess (§3.4), SignRound2 (§3.5), SignFinalize (§3.6) under the new committee V' on $\mu = \mu_{\text{act}}$.
3. Run Verify (§3.7) under the *unchanged* $(\mathbf{A}, \tilde{\mathbf{b}})$ from the pre-state.
4. If verification succeeds, the chain commits Π' and bumps $g \to g + 1$. If it fails, the chain stays at g and triggers Rollback (§3.11).

What this proves. Theorem 2: VerifyActivation accepts iff the new committee shares Lagrange-recombine to the original $\mathbf{s}$, except with negligible probability under MLWE/MSIS.

3.11 Rollback

Inputs: Target generation $g_{\text{target}} = g - 1$ (or any prior on-chain g_j); current state Π_{cur} at generation g.

Outputs: Restored state Π' at generation $g+1$ with $r = g_{\text{target}}$ and $\mathsf{GKH}(\Pi') = \mathsf{GKH}(\Pi_{g_{\text{target}}})$.

Steps:

1. Retrieve snapshot of $\Pi_{g_{\text{target}}}$ from `PulsarSnapshotManager`.
2. Reinstate per-party shares from the snapshot.
3. Bump generation to $g + 1$, set $r = g_{\text{target}}$, preserve η and GKH.
4. Emit a rollback evidence transaction with the failed activation cert (if any) for forensic review.

Invariant. By Lemma 13 of [27] ("Rollback-lineage forward reconstruction"), the on-chain (η, g, r) tuple sequence uniquely reconstructs the canonical forward path the rollback restored to. The chain stays signing-live throughout.

3.12 Reanchor (era boundary)

A governance event executes a fresh ceremony or Pedersen DKG (§9.6) producing $(\mathbf{A}_{\eta+1}, \tilde{\mathbf{b}}_{\eta+1})$ with $\mathsf{GKH}_{\eta+1} \neq \mathsf{GKH}_{\eta}$. The new era starts at $g = 0$, $r = 0$. Reanchor is rare (annual at most in production); it is the only transition that breaks GroupKey continuity.

4 Wire Format

This section is normative. The byte stream produced by the Go canonical's WriteTo methods is the reference; every C++, Metal, CUDA, and WGSL port [40] MUST emit byte-identical output for cross-implementation Known-Answer-Test acceptance.

4.1 Endianness and primitive widths

All multi-byte integers in the polynomial / vector / matrix payloads are encoded little-endian via `encoding/binary.LittleEndian` [31] (`lattice/utils/buffer/writer.go:325`). Coefficients are stored as `uint64` regardless of the modulus' bit-width: the 18-bit and 19-bit moduli of R_ξ and R_ν are zero-padded to 8 bytes per coefficient.

The MAC and transcript inputs use *big-endian* encodings via `binary.Write(BigEndian, ...)` for integer fields (*sid*, $|T|$, T_k, party ids). This is a deliberate split: ring-domain payloads are little-endian (matches `x86_64` memory layout for fast `memcpy`-style serialization); transcript metadata is big-endian (matches network-byte-order convention for cross-host transcripts).

4.2 Polynomial

A single `ring.Poly` at one RNS level has shape $1\times\varphi$. Its WriteTo output (`lattice/ring/poly.go:117`, `utils/structs/matrix.go:82`, `utils/structs/vector.go:82`) is:

Listing 1: Polynomial wire format ($\varphi = 256$).

```
uint64_LE  row_count    = 1
uint64_LE  col_count    = 256
uint64_LE  coeff[0..255]              # row-major
# total: 8 + 8 + 8*256 = 2064 bytes
```

A zero polynomial therefore serializes to `01 00 00 00 00 00 00 00 | 00 01 00 00 00 00 00 00 | (256 zero u64)`.

4.3 Vector and Matrix

Listing 2: Vector and Matrix wire formats.

```
Vector[Poly] of length n:
    uint64_LE  length = n
    Poly * n                  # each as above

Matrix[Poly] of shape m * n:
    uint64_LE  row_count = m
    Vector[Poly] * m          # each as above
```

A 1×1 matrix containing a zero polynomial is $8 + 8 + 2064 = 2080$ bytes.

4.4 Signature

Listing 3: Signature $\sigma = (c, z, \Delta)$ with $n_{\text{vec}} = 7$, $m_{\text{pk}} = 8$.

```
sigma = c || z || Delta
c     : Poly                          # 2064 bytes
z     : Vector[Poly] length 7         # 8 + 7*2064 = 14456 bytes
Delta : Vector[Poly] length 8         # 8 + 8*2064 = 16520 bytes
                                      # total: 33040 bytes
```

4.5 Group Key

Listing 4: Group key $(A, \tilde{b})$.

```
A     : Matrix[Poly] shape 8 x 7      # 8 + 8*(8 + 7*2064) = 116232 bytes
b_til : Vector[Poly] length 8         # 8 + 8*2064 = 16520 bytes
```

4.6 MAC input

GenerateMAC (`primitives/hash.go:34`) hashes the following stream with BLAKE3 and truncates to 32 bytes:

Listing 5: MAC input layout.

```
mac_input =
    int64_BE(prover_or_verifier_id)       # partyID when generating;
                                          # otherPartyID when verifying
 || MACKey[32]
 || Matrix[Poly].WriteTo(D_i)
 || int64_BE(sid)
 || int32_BE(|T|)
 || int32_BE(T_k) for k = 0..|T|-1        # in T's order
MAC = BLAKE3(mac_input)[:32]
```

The asymmetric "identity in slot 0" (*partyID* on generate, *otherPartyID* on verify) is intentional and prevents reflection of $\mathrm{MAC}_{i\to j}$ as $\mathrm{MAC}_{j\to i}$.

4.7 Transcript hashes

Hash (`primitives/hash.go:128`). Round 2 transcript hash:

```
Hash_input =
    Matrix[Poly].WriteTo(A)
 || Vector[Poly].WriteTo(b)
 || int64_BE(sid)
 || int32_BE(|T|)
 || int32_BE(T_k) for k = 0..|T|-1
 || Matrix[Poly].WriteTo(D_j) for j = 0..|T|-1   # ascending j
hash = BLAKE3(Hash_input)[:32]
```

This requires every implementation to populate the D map with contiguous integer keys $\{0, \ldots, |T| - 1\}$ before hashing. Non-contiguous keys produce a different transcript and the round will not converge.

LowNormHash (line 167). Challenge derivation:

```
LNH_seed = BLAKE3(
    Matrix[Poly].WriteTo(A)
 || Vector[Poly].WriteTo(b_til)
 || Vector[Poly].WriteTo(h_til)
 || []byte(mu)
)[:32]
prng = BLAKE2-XOF(LNH_seed)
c = TernarySampler(prng, R, Ternary{H: kappa}).Read()
c = NTT(c); c = MForm(c)
```

GaussianHash (line 75). Challenge-side u' vector:

```
GH_seed = BLAKE3(hash || []byte(mu))[:32]
prng = BLAKE2-XOF(GH_seed)
u' = SamplePolyVector(prng, R, DiscreteGaussian{sigma_U, B_U},
                      length=Dbar, NTT=true, Mont=true)
```

PRF (line 99). Pairwise mask:

```
PRF_seed = BLAKE3(k_PRF || sd_ij || hash || []byte(mu))[:32]
prng = BLAKE2-XOF(PRF_seed)
mask = SamplePolyVector(prng, R, UniformSampler,
                        length=n_vec, NTT=true, Mont=true)
```

4.8 Discrete-Gaussian sampler (Ziggurat)

The Discrete-Gaussian sampler is Marsaglia's Ziggurat [46, 31] with 128 strata over a BLAKE2-XOF byte stream (`lattice/ring/sampler_gaussian.go:48`). The pre-computed tables KN, WN, FN in the Lattigo source MUST be reproduced bit-exactly by every port.

Per coefficient: one 8-byte read for the strata index and sign (only the low 4 bytes are used; the next 4 bytes are skipped to maintain 8-byte buffer alignment, `sampler_gaussian.go:212`). On the rare strata-0 tail or inter-strata reject path, additional 8-byte reads draw a uniform $\mathbb{R} \in (0, 1)$ via `randF64`, masking 53 bits of the lower 8 bytes [31]. A constant-bound C++/GPU port MUST treat the 1024-byte internal buffer as a sliding window with the same refill discipline (one BLAKE2-XOF `Read(buf[0:1024])` when the buffer is exhausted).

4.9 Ternary sampler

For LowNormHash challenges with Ternary$\{H : \kappa\}$, TernarySampler.sampleSparse (`lattice/ring/sampler_ternary.go:198`) implements rejection sampling of κ distinct positions plus pairwise sign bits:

1. Initialize idx $= [0, 1, \ldots, \varphi - 1]$.
2. Read $\lceil \kappa/8 \rceil$ random bytes for sign bits.
3. For $k = 0, \ldots, \kappa - 1$:
   (a) Choose the smallest power-of-two mask $\geq \varphi - k$ (one byte at $\varphi = 256$).
   (b) Rejection-sample a position $p \in [0, \varphi - k)$, consuming bytes from the PRNG until a candidate $\leq \varphi - k - 1$ appears.
   (c) Set coeff[idx[p]] $\leftarrow$ $sign_k \,?\, q - 1 : 1$.
   (d) Swap idx[p] with idx[$\varphi - k - 1$] to prevent re-selection.
4. All other positions remain zero.

The output is in standard form; the protocol then applies NTT+MForm. C++/GPU ports MUST mirror the byte-stripe ordering of sign bits (MSB-first within byte) and the rejection-sampling loop.

4.10 Cross-implementation Known-Answer-Test contract

The reference KAT manifest is `corona/test/kat/manifest.sha256` in the Go canonical [42]. The contract is:

1. Fixed $k_{\text{TD}} \in \{$`0x00...00`, `0x01...01`, $\ldots$, `0x0F...0F`$\}$ (16 vectors).
2. Active set sizes $(t, n) \in \{(2, 3), (3, 5), (5, 7), (7, 10)\}$.
3. For each (k_{TD}, t, n) produce: A, $\tilde{b}$, every sk_i, every D_i, every $\text{MAC}_{i \to j}$, the transcript h_{tx}, every z_i, the final $\sigma = (c, z, \Delta)$, and the Verify accept bit.
4. The SHA-256 of every payload (each individually serialized via WriteTo) MUST match across the Go canonical, the C++ port, and any GPU kernel claiming Pulsar compatibility.

A C++ implementer reading §2–4 alone should produce a byte stream that hashes byte-equal to the Go canonical's WriteTo output without consulting any Go source.

5 Security

This section states the four production-grade security claims of Pulsar and points to the canonical proof artefact for each. Proof bodies live exactly once in `proofs/quasar-cert-soundness.tex` [27] and the Lean tree under `proofs/lean/Crypto/` [43]; this paper cites them verbatim and does not duplicate the bodies.

5.1 Security games

We adopt the standard *existential unforgeability under chosen message attack* (EUF-CMA) and *strong unforgeability under chosen message attack* (SUF-CMA) games for threshold signatures. Throughout, λ is the security parameter, $\mathcal{A}$ is a non-uniform PPT adversary (classical, $\mathcal{A}_C$, or quantum, $\mathcal{A}_Q$), $\mathcal{O}_{\mathrm{Sign}}$ is the threshold signing oracle, and $\mathcal{O}_H$ is the random oracle (programmable in QROM hybrids, following App. C of [27]). The full validator set is $\mathcal{V}$ with $|\mathcal{V}| = n$, threshold t, and stake distribution $\mathrm{stk} : \mathcal{V} \to \mathbb{N}$.

Definition 1 (Threshold EUF-CMA / SUF-CMA games)**.** *The game* $\mathsf{Game}_{\Pi}^{\mathrm{tEUF-CMA}}(\mathcal{A}, \lambda)$ *proceeds as follows.*

1. ***Setup.*** *The challenger runs* $\mathsf{Gen}(1^\lambda) \to (\mathsf{GK}, \{\mathbf{s}_v\}_{v \in \mathcal{V}})$*, where* $\mathsf{GK} = (\mathbf{A}, \tilde{\mathbf{b}})$ *is the public group key and* $\mathbf{s}_v$ *the share for* v*.* GK *is given to* $\mathcal{A}$*.*
2. ***Corruption.*** $\mathcal{A}$ *commits to a corrupt set* $F \subseteq \mathcal{V}$ *with stake* $\mathrm{stk}(F) < t$ *and receives* $\{\mathbf{s}_v : v \in F\}$*. (Static model. Adaptive is treated by App. C of [27] via Fischlin's transform with* $O(n \cdot Q_H)$ *loss.)*
3. ***Queries.*** $\mathcal{A}$ *may invoke* $\mathcal{O}_{\mathrm{Sign}}(\mu_i)$ *for messages* $\mu_1, \ldots, \mu_{Q_S}$ *chosen adaptively; each query returns a Pulsar signature* σ_i *produced by an honest threshold protocol over a quorum the challenger selects.* $\mathcal{A}$ *may invoke* $\mathcal{O}_H$ *up to* Q_H *times.*
4. ***Forgery.*** $\mathcal{A}$ *outputs* $(\mu^\star, \sigma^\star)$*.* $\mathcal{A}$ wins *the EUF-CMA game iff* $\mathsf{Verify}(\mathsf{GK}, \mu^\star, \sigma^\star) = 1$ *and* $\mu^\star \notin \{\mu_1, \ldots, \mu_{Q_S}\}$*.* $\mathcal{A}$ *wins the SUF-CMA game iff* $\mathsf{Verify}(\mathsf{GK}, \mu^\star, \sigma^\star) = 1$ *and* $(\mu^\star, \sigma^\star) \notin \{(\mu_i, \sigma_i)\}_{i=1}^{Q_S}$*.*

The advantages are

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathrm{tEUF-CMA}}(\lambda) = \Pr\big[\mathcal{A} \text{ wins EUF-CMA}\big], \qquad \mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathrm{tSUF-CMA}}(\lambda) = \Pr\big[\mathcal{A} \text{ wins SUF-CMA}\big].$$

Π *is* $(\mathcal{Q}, \epsilon)$*-tEUF-CMA secure if* $\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathrm{tEUF-CMA}} \le \epsilon$ *for every* $\mathcal{A}$ *running within budget* $\mathcal{Q} = (Q_S, Q_H, T)$*.*

For the lifecycle layer we additionally consider the *key-era unforgeability* game, in which $\mathcal{A}$ may also invoke $\mathcal{O}_{\mathrm{Reshare}}, \mathcal{O}_{\mathrm{Refresh}}, \mathcal{O}_{\mathrm{Reanchor}}$ subject to corruption-budget bounds at each transition.

Definition 2 (Key-era unforgeability)**.** *The game* $\mathsf{Game}_{\Pi}^{\mathrm{KE-UF}}(\mathcal{A}, \lambda)$ *extends Definition 1 with three lifecycle oracles. At each transition,* $\mathcal{A}$ *commits to the corrupt subset of the post-state validator set with* $\mathrm{stk}(F) < t$*.* $\mathcal{A}$ wins *iff it produces a forgery* $(\mu^\star, \sigma^\star)$ *on* any *live key era's GroupKey for an unsigned* $\mu^\star$*.*

5.2 Theorem 5.1: Pulsar SUF-CMA under MLWE / MSIS

Theorem 1 (Pulsar threshold SUF-CMA, MAC-bound Round 1)**.** *Let* Π_{Pulsar} *be the LP-073 protocol of §3 at the parameter set of §2. Under the Module-LWE assumption* $\mathsf{MLWE}_{q,m_{\mathrm{pk}},n_{\mathrm{vec}},\chi_E}$ *on* $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ *with* $q \approx 2^{48}$ *and secret/error distribution* $\chi_E = \mathcal{D}(\sigma_E)$*,* $\sigma_E \approx 6.108$*, and the Module-SIS assumption* $\mathsf{MSIS}_{q,m_{\mathrm{pk}},\beta}$ *at norm bound* $\beta = B \approx 2^{48.61}$*, in the random-oracle model on* BLAKE3 *and the PRF assumption on the BLAKE3-keyed MAC,*

$$\mathsf{Adv}_{\Pi_{\mathrm{Pulsar}},\mathcal{A}}^{\mathrm{tSUF-CMA}}(\lambda) \;\le\; \epsilon_{\mathrm{MLWE}}(\lambda) + \epsilon_{\mathrm{MSIS}}(\lambda) + \frac{Q_H + Q_S}{2^{256}} + Q_S \cdot 2^{-128} + \frac{Q_M}{2^{256}},$$

where Q_H, Q_S, Q_M are random-oracle, signing, and MAC queries respectively. The first two terms are negligible at the LP-073 §2 parameters (App. E of [27]); the third bounds challenge collisions on the 256-bit transcript hash; the fourth bounds the rejection-sampling slack at $\eta_\varepsilon = 2.650104$ over Q_S signing transcripts at the 80-bit statistical-distance margin [19]; the fifth bounds MAC forgery on Round 1 broadcasts.

Proof reference. The Pulsar kernel SUF-CMA reduction is the standard Fiat–Shamir- with-aborts proof of [7], instantiated at the LP-073 parameter set. The Pulsar wrapper adds (i) the Round 1 MAC layer and (ii) the lifecycle compatibility (Lemmas 10–13 of [27], "Pulsar Key-Era Lifecycle Lemmas" section). The Lean specification `Crypto.Pulsar.corona_pq_unforgeability` in `proofs/lean/Crypto/Corona.lean` [43] encodes the unforgeability axiom against quantum adversaries with compromised $< t$. The detailed bound derivation, including App. E parameter analysis and the BDGL/Grover cost equations, is in [27] Theorem 5 (QuasarCert classical soundness, Lane 2 case) and App. E. □

Round 1 MAC subgame. A network attacker that observes Round 1 broadcasts cannot replace any party's D_i with a malicious D_i' unless it forges the corresponding $\mathrm{MAC}_{i\to j}$. The MAC is BLAKE3 [51] truncated to 256 bits over a payload that includes the symmetric pairwise key $\mathrm{MACKEYS}[i][j]$. Under the PRF assumption on BLAKE3, the attacker's MAC-forgery advantage is $Q_M/2^{256}$ where Q_M is the number of MAC queries observed; at $Q_M \le 2^{30}$ over an epoch lifetime, this is below 2^{-220}. The attack class of [18] (sub-exponential forgery on unauthenticated concurrent commitment broadcast) is closed by the combination of MAC authentication and per-party PRNG seeding from BLAKE3(WRITETO(sk_i)) (`sign.go:103`).

Full-rank subgame. SIGNROUND2PREPROCESS performs a per-coefficient full-rank check on D_Σ after dropping column 0 (`sign.go:348`). This rules out residual malicious commitments that pass the MAC layer (e.g., $D_i' = D_i + V$ where V has rank-deficient slot k) but admit a re-opening attack. The check costs φ Gaussian eliminations on $m_{\mathrm{pk}} \times \bar{D} = 8 \times 48$ matrices over $\mathbb{F}_q$.

L_2 acceptance. The verifier accepts iff the squared L_2 norm of (Δ, z) over signed coefficients in $(-q/2, q/2]$ is at most $B^2 = 184\,960\,669\,042\,442\,604\,975\,662\,780\,477$. The slack reservation $\eta_\varepsilon \cdot \sigma_*$ in r^* ensures honest signers pass with probability $> 1 - 2^{-128}$ at the canonical parameters; a forger cannot inflate Δ to compensate for an arbitrary c without exceeding the bound, by SIS hardness on $[\mathbf{A} \mid \mathbf{I}]$ at norm $\beta = B$.

5.3 Theorem 5.2: Activation-certificate circuit-breaker

Theorem 2 (Activation-certificate soundness)**.** *Let Π be a pre-activation Pulsar state at (η, g) and Π' the proposed post-activation state at $(\eta, g+1)$. Define* $\mathsf{VerifyActivation}(\Pi, \Pi', \sigma) = 1$ *iff σ is a valid Pulsar threshold signature on the activation message* $\mathsf{ActivationMsg}_{\eta,g+1}$ *of §9.4 that verifies under the* unchanged *GroupKey* $\mathsf{GK}(\Pi) = (\mathbf{A}, \tilde{\mathbf{b}})$*. Then under MLWE/MSIS at the LP-073 §2 parameters,*

$$\Pr\big[\mathsf{VerifyActivation}(\Pi, \Pi', \sigma) = 1 \wedge \Pi' \text{ does not Lagrange-recombine to } \mathbf{s}\big] \le \mathsf{Adv}^{\mathrm{tSUF-CMA}}_{\Pi_{\mathrm{Pulsar}},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda).$$

Equivalently: VerifyActivation accepts iff the new committee shares can sign under the unchanged GroupKey, except with negligible probability.

Proof reference. This is Theorem 14 of [27] ("Activation- certificate soundness"). The proof reduces acceptance under the unchanged GroupKey to a Pulsar SUF-CMA challenge: a malformed Π' that nonetheless produces an accepting σ would yield a fresh MLWE/MSIS instance breaking Theorem 1 above. The forward direction follows from the Lagrange-recombination identity (see Theorem 5.3 below). □

5.4 Theorem 5.3: Resharing preserves master secret

Theorem 3 (Lagrange-recombination identity)**.** *Let $f(x) \in R_q[X]$ be the old-set sharing polynomial of degree $t_{\text{old}} - 1$ with $f(0) = \mathbf{s}$ and $f(\alpha_v) = \mathbf{s}_v$ for $v \in V$. After* Reshare(V', t') *with qualified subset $Q \subseteq V$, $|Q| \geq t_{\text{old}}$, where each $i \in Q$ samples $g_i(x)$ of degree $t'-1$ with $g_i(0) = \mathbf{s}_i$, the new sharing polynomial*

$$G(x) = \sum_{i \in Q} \lambda_i^Q(0) \cdot g_i(x)$$

satisfies

$$\begin{aligned} G(0) \;&=\; \sum_{i \in Q} \lambda_i^Q(0) \cdot g_i(0) \;=\; \sum_{i \in Q} \lambda_i^Q(0) \cdot \mathbf{s}_i \\ &=\; \sum_{i \in Q} \lambda_i^Q(0) \cdot f(\alpha_i) \;=\; f(0) \;=\; \mathbf{s}, \end{aligned} \tag{1}$$

where the penultimate equality is the standard Lagrange interpolation identity over R_q. Consequently, $(\mathbf{A}, \mathbf{s}, \mathbf{e})$ is preserved across Reshare, $\tilde{\mathbf{b}} = \lfloor \mathbf{A}\mathbf{s} + \mathbf{e} \rceil_\xi$ is byte-preserved, and $\mathsf{GKH}(\Pi) = \mathsf{GKH}(\Pi')$.

Proof reference. This is Lemma 12 of [27] ("GroupKey byte-identity across Refresh and Reshare"), proved via equation (1). The Lean encoding is `Crypto.Threshold.Reshare.reshare_preserves_publi` in `proofs/lean/Crypto/Threshold/Reshare.lean` [43], which derives $\mathsf{pubkey\_from\_shares}(\mathsf{newShares}, t') = \mathsf{pubkey}(\mathsf{key})$ from the underlying axiom `reshare_same_secret`. □

For Refresh (HJKY97), an analogous identity applies with $z_i(x)$ a zero-polynomial of degree $t-1$ [23]: the constant term contributes nothing to $G(0)$, so $\mathbf{s}, \mathbf{e}, \tilde{\mathbf{b}}$ are byte-preserved. Verifiability of each contribution under lattice Pedersen commits is per Wong–Wang–Wing [54] (LP-073 §9.6); the binding/hiding properties hold under MSIS on $[\mathbf{A} \,|\, \mathbf{B}]$ and MLWE on $\mathbf{B}$ respectively.

5.5 Theorem 5.4: Lineage unforgeability

Theorem 4 (KeyEraID and Generation lineage unforgeability)**.** *Let $\mathcal{C} = (\eta_i, g_i, r_i)_{i=0}^N$ be the on-chain lineage record across an arbitrary execution of Pulsar lifecycle transitions. Under the activation-cert soundness theorem (Theorem 2 above) and the collision resistance of* BLAKE3*, no PPT adversary $\mathcal{A}$ can produce $\mathcal{C}^\star \neq \mathcal{C}$ that the chain accepts as a valid lineage on the same epoch sequence, except with probability* $\mathsf{Adv}^{\text{tSUF-CMA}} + Q_H/2^{256}$.

Proof reference. The lifecycle record is enforced by chain-state transitions that each require a valid activation cert (Theorem 2 above). By Lemmas 10–13 of [27] (Generation monotonicity, KeyEraID stability, GroupKey byte-identity, Rollback- lineage forward reconstruction), the triple (η, g, r) uniquely determines the lineage chain up to permutation of forward-then-rolled-back branches; these are unambiguously labelled by $r > 0$. Forging a different valid $\mathcal{C}^\star$ requires either (a) forging an activation cert (reduces to Theorem 2) or (b) finding a BLAKE3 collision on $\mathsf{ActivationMsg}$ ($Q_H/2^{256}$). The detailed reverse-walk construction that recovers the canonical forward path appears in the proof of Lemma 13 of [27]. □

5.6 Concrete parameter analysis

The classical and post-quantum cost analysis for the Pulsar parameter set lives in App. E of [27] and is summarised here for the bound substitutions in Theorems 5.1–5.4.

Lattice dimension. Module-LWE underlying Pulsar has $d = m_{\text{pk}} \cdot \varphi = 8 \cdot 256 = 2048$ over $\mathbb{Z}_q$ with $\log_2 q \approx 48$.

Classical cost (BDGL primal sieving). For $d = 2048, q \approx 2^{48}, \sigma_E \approx 6.108$, the optimal BKZ block size is $\beta \approx 430$. Under the Becker–Ducas–Gama– Laarhoven exponent 0.292 [4],

$$T_{\text{classical}} \;=\; 2^{0.292\beta + c_0} \;\approx\; 2^{0.292 \cdot 430 + 16.4} \;=\; 2^{142}.$$

Quantum cost (Grover-amplified sieving). Under the Laarhoven–BDGL exponent 0.265 for Grover-amplified sieving [30, 4],

$$T_{\text{quantum}} \;=\; 2^{0.265\beta + c_0} \;\approx\; 2^{0.265 \cdot 430 + 16.4} \;=\; 2^{130.35}.$$

This is the substitution in Theorem 1's $\epsilon_{\text{MLWE}}, \epsilon_{\text{MSIS}}$ at $\lambda = 128$: the bound saturates the NIST Cat. 3 quantum target with $\geq 2^{130}$ headroom.

MSIS bound. The verifier accepts at L_2-norm bound $\beta = B \approx 2^{48.61}$. SIS at $(\mathbf{A}, \beta)$ on the 2048-dimensional module is hard at the same BKZ block size; the honest forger must find (Δ, z) within β that satisfies the $Az - bc$ identity, which by linear-algebra reduces to a short-vector search on a coset of the SIS lattice. The cost matches the MLWE estimate above to within $O(1)$ in the exponent.

Substitution into Theorem 5.1. At $\lambda = 128$ and $Q_S, Q_H \leq 2^{60}$ (a generous epoch-lifetime bound):

$$\begin{aligned}
\mathsf{Adv}^{\text{tSUF-CMA}}_{\Pi_{\text{Pulsar}}, \mathcal{A}_C} &\leq 2^{-142} + 2^{-142} + 2^{-196} + 2^{-68} + 2^{-196} \;\leq\; 2^{-67}, \\
\mathsf{Adv}^{\text{tSUF-CMA}}_{\Pi_{\text{Pulsar}}, \mathcal{A}_Q} &\leq 2^{-130} + 2^{-130} + 2^{-196} + 2^{-68} + 2^{-196} \;\leq\; 2^{-67}.
\end{aligned}$$

The dominant term is the rejection-sampling slack $Q_S \cdot 2^{-128}$; the lattice term is far below the budget. To shrink the slack one shrinks Q_S by limiting per-epoch signing or by adopting the tighter slack of 2^{-256} at the cost of one extra σ_E-deviation in the Gaussian sampler. Lux production accepts 2^{-67} as the operative bound at the LP-073 parameters; raising the slack tier is tracked as a live parameter knob.

5.7 Side-channel posture

Lattigo's `MRed` and `MRedLazy` [31] are constant-time on a 64-bit `bits.Mul64` substrate (Go and C++ on x86/ARM 64). The Ziggurat [46] and ternary samplers contain rejection-sampling loops whose iteration count depends on the random bytes consumed; the PRNG seed is independent of the secret in every call site except $\text{PRNGKey}(\text{sk}_i)$ (`primitives/hash.go:19`), where the BLAKE3 hashing is constant-time and the resulting k_{sk} is used only as a seed for the per-session PRNG. The dependence between secret-derived seed and observable PRNG output is bounded by BLAKE3 collision resistance (PRF-secure for $Q_H \leq 2^{128}$).

5.8 Nonce-uniqueness invariant

The per-session PRNG is seeded deterministically from $\text{BLAKE3}(\text{WriteTo}(\text{sk}_i))$, which means the same $(\text{sk}_i, \mathit{sid}, \mu)$ triple produces the same R_i sequence. This is safe iff sid is unique per signing session. In Lux Quasar deployment (LP-020 [33]), sid is the (chain id, block height) pair, whose monotonicity is enforced by P-Chain consensus; this provides session-id uniqueness as a chain invariant rather than a runtime obligation. Under Theorem 1 a session-id collision would manifest as a transcript-hash collision and reduce to BLAKE3 collision-resistance, but the chain rules out the collision before the PRNG is reseeded.

5.9 Identifiable abort posture

Pulsar identifies misbehaviour at two layers:

- **Round 1 MAC layer.** A failed MAC verification names the sender (the verifier can identify which i's broadcast was tampered with). This is per-party identifiable abort by construction.
- **Round 2 share layer.** VERIFY can run per-share: z_j verifies independently against $(D_j, \lambda_j, c, \tilde{b})$ via the same $Az_j - b \cdot \lambda_j c$ identity restricted to share j. The first failing index identifies the corrupt z_j and is recorded as slashing-evidence material on chain. The chain's safety and liveness do not depend on this attribution; per §9.5, a failed activation triggers rollback rather than slashing.

5.10 Composition with Quasar Horizon

Pulsar is one of three lanes in QuasarCert [27]. The composed soundness theorem (Theorem 5 of [27]) bounds the adversary advantage by the minimum of three per-lane advantages (BLS, Pulsar, ML-DSA-via-Groth16); under quantum adversary the BLS lane is broken by Shor and the standalone-PQ predicate $\mathsf{VerifyPQ} = \mathsf{RT.Verify} \wedge \mathsf{ZK.Verify}$ remains sound, with Pulsar's contribution bounded by Theorem 1 above (Theorem 8 of [27], "post-quantum safety"). Subject-binding replay protection across epochs is per Theorem 9 of [27] ("subject-binding replay protection"): every Pulsar pulse hashes the certificate subject including $(\mathsf{chain\_id}, \mathsf{epoch}, \mathsf{round}, \mathsf{KeyEraID}, \mathsf{Generation})$, so cross-epoch replay reduces to BLAKE3 collision resistance.

6 Formal Model

This section formalises the validator-set model, adversary model, corruption model, and lineage chain on top of which the security theorems of §5 are stated. The model is shared with [27] App. C and is reproduced here in the form needed by Pulsar's lifecycle theorems.

6.1 Validator set, key era, and persistent group key

Definition 3 (Validator set at epoch τ)**.** *The Lux P-Chain anchors a validator-set sequence $(\mathcal{V}_\tau)_{\tau \geq 0}$ where $\mathcal{V}_\tau \subseteq \mathcal{V}^*$ is the validator subset active at epoch τ with stake distribution $\mathrm{stk}_\tau : \mathcal{V}_\tau \to \mathbb{N}$. The total stake at epoch τ is $S_\tau = \sum_{v \in \mathcal{V}_\tau} \mathrm{stk}_\tau(v)$. The Byzantine bound is $f_\tau < S_\tau/3$ (i.e., total stake controlled by an adversary at epoch τ is strictly less than one third of S_τ); the threshold t_τ is the smallest integer with $t_\tau \geq \lceil 2S_\tau/3 \rceil$. Membership in $\mathcal{V}_\tau$ is determined by P-Chain governance and stake delegation; transitions between $\mathcal{V}_\tau$ and $\mathcal{V}_{\tau+1}$ are permitted at every epoch boundary.*

Definition 4 (Key era)**.** *A* key era E_η *is the persistent state*

$$\mathsf{E}_\eta = (\eta,\ \mathbf{A}_\eta,\ \tilde{\mathbf{b}}_\eta,\ \mathsf{GKH}_\eta,\ \mathrm{birth}(\eta),\ \mathrm{death}(\eta))$$

where $\eta \in \mathbb{N}$ is the era identifier; $(\mathbf{A}_\eta, \tilde{\mathbf{b}}_\eta)$ is the persistent group public key with $\mathbf{A}_\eta \in R_q^{m_{\mathrm{pk}} \times n_{\mathrm{vec}}}$, $\tilde{\mathbf{b}}_\eta \in R_\xi^{m_{\mathrm{pk}}}$; $\mathsf{GKH}_\eta = \mathsf{BLAKE3}(\mathsf{WriteTo}(\mathbf{A}_\eta) \,\|\, \mathsf{WriteTo}(\tilde{\mathbf{b}}_\eta))[:32]$ the canonical GroupKey hash; and $\mathrm{birth}(\eta), \mathrm{death}(\eta)$ are the P-Chain block heights at which E_η is anchored and superseded. An era is live *during $[\mathrm{birth}(\eta), \mathrm{death}(\eta))$. The triple $(\mathbf{A}_\eta, \tilde{\mathbf{b}}_\eta, \mathsf{GKH}_\eta)$ is byte-immutable across the era's lifetime; this is Lemma 12 of [27] ("GroupKey byte-identity across Refresh and Reshare") and Theorem 3 of LP-073 §5.*

Definition 5 (Generation lineage within an era)**.** *Within E_η, the chain records a strict-monotone generation sequence*

$$\mathcal{C}_\eta = \big((g_0, V_0, t_0, r_0),\ (g_1, V_1, t_1, r_1),\ \ldots,\ (g_{k_\eta}, V_{k_\eta}, t_{k_\eta}, r_{k_\eta})\big)$$

with $g_0 = 0$, $g_{i+1} = g_i + 1$ (strict-monotone bump on every transition, including rollback). $V_i \subseteq \mathcal{V}_{\tau_i}$ is the active threshold committee at generation i, t_i the threshold, r_i the RollbackFrom counter ($r_i = 0$ on forward; $r_i = j > 0$ for a rollback to generation j). The lineage is sealed at $\mathrm{death}(\eta)$ *when a Reanchor produces* $\mathsf{E}_{\eta+1}$.

6.2 KeyEraID lineage as a chain of tuples

The full lifecycle of Pulsar is a concatenation of per-era lineages:

$$\mathsf{LIN} = (\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots).$$

Every entry in every $\mathcal{C}_\eta$ is a tuple (η, g, V, t, r) committed on chain via the activation cert of LP-073 §9.4. We treat LIN as a labelled graph: nodes are tuples; edges are activations (forward, $r = 0$) or rollbacks ($r > 0$). The Reanchor edge $\eta \to \eta + 1$ is labelled with a fresh ceremony or DKG transcript and carries the proof-of-knowledge that $(\mathbf{A}_{\eta+1}, \tilde{\mathbf{b}}_{\eta+1})$ was sampled honestly per LP-073 §9.6. The Reanchor edge breaks GroupKey continuity by definition: $\mathsf{GKH}_\eta \neq \mathsf{GKH}_{\eta+1}$ except with probability 2^{-256} (BLAKE3 collision).

Remark 1 (Multi-group lineage)**.** *For grouped Pulsar deployments (§12), the lineage chain generalises to per-group lineages* $\mathsf{LIN}^{(j)}$, $j = 1, \ldots, G$, *with each group running an independent lifecycle. Group-level rotations do not affect other groups; cross-group consistency is enforced only at the Horizon certificate (§13) where the G Pulsar pulses are combined into a Horizon-cert quorum.*

6.3 Adversary models

We adopt the standard cryptographic adversary classes, refined for the threshold-blockchain setting per Garay–Kiayias–Leonardos [21].

Definition 6 (Honest-but-curious)**.** *A party v is* honest-but-curious *(HBC) iff v executes the protocol exactly as specified but logs every received message, intermediate computation, and outgoing message for offline analysis. HBC parties do not deviate from the prescribed steps; they merely attempt to extract as much information as possible from the protocol transcript.*

Definition 7 (Malicious)**.** *A party v is* malicious *(or Byzantine) iff v may deviate arbitrarily from the protocol: omit messages, send malformed messages, collude with other malicious parties, refuse to participate, or participate with crafted inputs designed to bias the protocol output.*

Definition 8 (Adversary classes)**.** *We consider:*

- *$\mathcal{A}_C$: classical PPT, controls a corrupt set $F_\tau \subseteq \mathcal{V}_\tau$ at each epoch with $\mathrm{stk}_\tau(F_\tau) < t_\tau$, has access to all corrupt parties' state and signing oracles, polynomially bounded random-oracle queries.*
- *$\mathcal{A}_Q$: quantum PPT, same corrupt-set bound, full quantum random-oracle access (QROM), can run Shor and Grover in polynomial quantum time.*

6.4 Static vs adaptive corruption

Definition 9 (Static corruption [10])**.** *The adversary $\mathcal{A}$ commits to the corrupt set $F = \bigcup_\tau F_\tau$ before the protocol begins. Within an epoch, F_τ is fixed at protocol start; new validators in $\mathcal{V}_{\tau+1} \setminus \mathcal{V}_\tau$ are added to $F_{\tau+1}$ only if $\mathcal{A}$ committed to them at the global start.*

Definition 10 (Adaptive corruption [10])**.** *$\mathcal{A}$ may corrupt parties at any point during execution. Upon corrupting v, $\mathcal{A}$ learns v's current state (keys, randomness, in-flight transcripts). The aggregate stake of corrupted parties remains bounded by t_τ at every epoch τ.*

Definition 11 (Mildly adaptive (blockchain model)). *$\mathcal{A}$ corrupts parties* between rounds *but not within a round. This is the canonical model for partially synchronous blockchain consensus per [21] and matches the assumption in LP-020 [33].*

Definition 12 (Erasure / Fischlin hybrid). *A party v* erases *a value x at time τ if any subsequent state inspection at $\tau' > \tau$ does not reveal x. The* erasure model *assumes honest parties erase per-round randomness immediately after the round completes; the* Fischlin hybrid *[20] replaces this with a programmable random oracle that pays a $O(n \cdot Q_H)$ multiplicative loss in the security reduction.*

Pulsar adopts both. The Go canonical performs explicit memory zeroing of r^*, e^*, R_i, E_i after Round 2 completes (`sign/sign.go` signing-finish blocks; `primitives/hash.go` PRNG buffer scrub). HSM-backed signing on the Lux T-Chain (LP-020 [33] §5) provides hardware-enforced erasure. By Cor. 16 ("Adaptive soundness, erasure model") of [27], Theorem 1 extends to mildly adaptive corruption with no additional security loss.

6.5 Soundness and liveness goals

We state the formal goals that Pulsar must satisfy as a permissionless- consensus threshold-signature layer.

Definition 13 (Soundness goal: SUF-CMA across eras). *For every era E_η and every adversary $\mathcal{A} \in \{\mathcal{A}_C, \mathcal{A}_Q\}$ within (Q_S, Q_H, Q_M, T) budget,*

$$\mathsf{Adv}^{\mathrm{tSUF-CMA}}_{\Pi_{\mathrm{Pulsar}},\eta,\mathcal{A}}(\lambda) \;\le\; \mathsf{negl}(\lambda)$$

under MLWE/MSIS at the LP-073 §2 parameters and the random-oracle / PRF assumptions on BLAKE3.

Definition 14 (Lifecycle soundness goal: lineage unforgeability). *For every adversary $\mathcal{A}$, the probability that $\mathcal{A}$ produces an alternate valid lineage $\mathsf{LIN}^\star \neq \mathsf{LIN}$ at the same epoch sequence is bounded by $\mathsf{Adv}^{\mathrm{tSUF-CMA}} + Q_H/2^{256}$ (Theorem 4).*

Definition 15 (Liveness goal: progress under partial synchrony). *Under the partial-synchrony model of [21] with GST Δ, every epoch with $\mathrm{stk}(\mathcal{V}_\tau \setminus F_\tau) \ge t_\tau$ produces at least one accepting Pulsar pulse within wall-clock time $T_{\mathrm{Pulsar}} + \Delta$ where T_{Pulsar} is the Round 1 + Round 2 + Combine wall-clock at $|V_\tau| = K$ on the production hardware (LP-073 §7, ~ 200 ms at $K = 5$ on Apple M2 Ultra, under 1 s at $K = 21$).*

Definition 16 (Activation soundness). *Every accepted lifecycle transition $\mathcal{C}_\eta[i] \to \mathcal{C}_\eta[i+1]$ within an era satisfies the activation cert soundness of Theorem 2: under unchanged $(\mathbf{A}_\eta, \tilde{\mathbf{b}}_\eta)$, the new committee's shares Lagrange-recombine to $\mathbf{s}_\eta$ except with probability $\le \mathsf{Adv}^{\mathrm{tSUF-CMA}}$.*

Definition 17 (Rollback soundness). *Every accepted Rollback transition restores a state that was committed on chain at a prior generation; the restored state's GroupKey hash matches by Lemma 12 of [27] ("GroupKey byte-identity"), and the lineage audit of Lemma 13 of [27] ("Rollback-lineage forward reconstruction") reconstructs the canonical forward path uniquely.*

The four goals (Definitions 13–17) together formalise the security target of LP-073: a permissionless- consensus PQ threshold signature with verifiable lifecycle, per-era SUF-CMA, and progress under partial synchrony with mildly adaptive adversaries up to $f < n/3$ stake.

6.6 What this model omits

Three model assumptions are made explicit so that downstream proofs inherit them cleanly:

- **P-Chain consensus.** The validator-set sequence $(\mathcal{V}_\tau)$ is delivered to Pulsar by the P-Chain. We assume P-Chain consensus is sound and live; Pulsar inherits epoch-boundary commitments by reference (LP-020 [33] §6). A P-Chain-level safety violation that produced two conflicting $\mathcal{V}_\tau$ sequences would invalidate Pulsar's epoch-bound state, but this is out-of-scope: it is treated by Theorem 7.5 of LP-020 and is the responsibility of QuasarCert as a whole.
- **Network model.** We assume partially synchronous delivery with GST Δ post-stabilisation. Asynchronous adversaries can delay messages indefinitely pre-GST; this affects liveness only, not soundness, and is matched by every BFT consensus engine in production.
- **Setup ceremony.** Genesis of E_0 is one of (a) foundation MPC ceremony with toxic-waste assumption confined to the first era, or (b) Pedersen DKG over R_q per §9.6. Subsequent Reanchors use the same two paths. Both paths are formalised in App. D of [27] (multi-party SRS ceremony, security per Bowe–Gabizon–Miers [9]).

7 Implementation

The shipping implementation spans four repositories: `github.com/luxfi/pulsar` (math kernel + key-era lifecycle), `github.com/luxfi/threshold` (LSS framework + Pulsar adapter), `github.com/luxfi/consensus` (Quasar integration), and `github.com/luxfi/warp` (cross-chain envelope). All layers are byte-equal across the Go canonical and the in-flight C++ port; KAT vectors gate every release.

7.1 Go canonical

The production reference is `github.com/luxfi/pulsar` [26] (the Sign math is forked byte-equal from the upstream academic reference [7]). The relevant files are:

- `sign/config.go` (28 lines): every numeric constant, listed in Table 2.
- `sign/sign.go` (374 lines): Gen, SignRound1, SignRound2Preprocess, SignRound2, SignFinalize, Verify, CheckL2Norm, FullRankCheck.
- `threshold/threshold.go` (294 lines): higher-level wrapper around `sign` that exposes GenerateKeys(t, n, rand), the Signer object, and the Round1Data/Round2Data/Signature wire types.
- `primitives/hash.go` (209 lines): Hash, LowNormHash, GaussianHash, PRF, GenerateMAC, PRNGKey.
- `primitives/shamir.go` (147 lines): the optimized $t = K$ Shamir path and ComputeLagrangeCoefficients.
- `utils/utils.go` (524 lines): NTT-aware matrix-vector multiplies (MatrixVectorMul, MatrixMatrixMul, VectorPolyMul), rounding helpers (RoundVector, RestoreVector, RoundCoefficients), Gaussian elimination over $\mathbb{F}_q$ (GaussianEliminationModQ), and the BLAKE3-XOF random-byte budget (PrecomputeRandomness).

The canonical depends on `github.com/luxfi/lattice/v7` [31] for ring arithmetic, NTT, samplers, and serialization. The version pin is normative: bumping to `v8` or upstream Lattigo would introduce SIMD-NTT path differences that break byte equality on the WriteTo stream.

7.2 Key-era lifecycle

The lifecycle additions to upstream Pulsar (Section 9) live alongside the math kernel:

- `pulsar/keyera/keyera.go`: Bootstrap, Reshare, Reanchor entrypoints. Three lineage scalars (KeyEraID, Generation, RollbackFrom) per Section 9.1; KeyEra carries a GroupKey pointer that is byte-identical across every reshare within one era.
- `pulsar/reshare/`: VSR primitives. Two distinct callable shapes: Refresh (HJKY97 same-set zero-poly [23]) and ReshareToNewSet (Desmedt–Jajodia old-qualified-subset [17]). Plus `commit.go` for lattice Pedersen verification, `transcript.go` for canonical transcript bytes under TupleHash256, and `activation.go` for the `QUASAR-PULSAR-ACTIVATE-v1` message.
- `pulsar/dkg2/`: the Pedersen-DKG-over-R_q trusted-dealer-free path (§9.6). Hiding under MLWE on $\mathbf{B}$; binding under MSIS on $[\mathbf{A} \mid \mathbf{B}]$. Ships as a parallel-track production option to the Bootstrap MPC ceremony.
- `pulsar/hash/`: the canonical `Pulsar-SHA3` hash profile (TupleHash256 / cSHAKE256 over FIPS 202 / SP 800-185). The default suite identifier `"Pulsar-SHA3"` pins the transcript-binding profile.

7.3 LSS-Pulsar adapter

The orchestration layer is the LSS framework [53, 39] at `github.com/luxfi/threshold`. The Pulsar adapter at `threshold/protocols/lss/lss_pulsar.go` is the analogue of `lss_cmp.go` (ECDSA) and `lss_lens.go` (curve, LP-103 [44]). Section 11 elaborates the adapter contract; this implementation note records the file layout:

- DynamicResharePulsar validates KeyEraID, Generation, GroupKey-pointer agreement; delegates lattice math to `pulsar/keyera`; bumps Generation; preserves KeyEraID; sets RollbackFrom = 0 on forward transitions.
- PulsarSnapshotManager retains a configurable rolling history of share states; Rollback(*targetGeneration*) restores a prior snapshot under the no-slashing failure ladder (§9.5).
- BuildActivationTranscript produces the canonical `reshare.TranscriptInputs` binding chain id, network id, key-era id, generations, epochs, set hashes, thresholds, group-pk hash, Nebula root, hash-suite id, implementation version, and variant.

7.4 Quasar consensus integration

`github.com/luxfi/consensus/protocol/quasar` consumes `pulsar/keyera` and the LSS-Pulsar adapter:

- `epoch.go`: InitializeEpoch delegates to `keyera.Bootstrap`; RotateEpoch delegates to lss.DynamicResharePulsar gated by activation-cert verification under the unchanged GroupKey.
- `reshare_epoch.go`: production wiring under the `QUASAR-PULSAR-ACTIVATE-v1` domain prefix; ReshareEpoch (set rotation) and RefreshEpoch (same-set proactive refresh).
- `types.go`: `EpochKeys` carries `KeyEraID`, `Generation`, `RollbackFrom`; BundleSigner re-derives Lambda for the participating subset.

The 258-test Quasar suite exercises the integrated path; the consensus build passes `GOWORK=off go test ./protocol/quasar/...` cleanly.

7.5 Warp 2.0 cross-chain envelope

The cross-chain emission of a Pulse on a Warp message envelope lives at `github.com/luxfi/warp` (root) plus `github.com/luxfi/warp/pulsar` (Pulse path). The split exists so the root warp package does not import the Pulsar kernel directly; the dispatch surface is the small `PulseVerifier` interface, with the concrete kernel-driven verifier in the subpackage:

- `warp/envelope.go`: `EnvelopeV2` carries the v1 Beam (`Message`) plus the four transcript-binding fields (`SourceNebulaRoot`, `SourceKeyEraID`, `SourceGeneration`, `HashSuiteID`) and two optional PQ lanes (`PulsarPulse`, `MLDSACertSet`). The leading `0x02` byte discriminates v2 from v1 wire bytes.
- `warp/pulsar/pulsar.go`: KernelVerifier drives the lattice verification path; BuildSigningBytes produces the canonical transcript under the `WARP-PULSAR-ENVELOPE-v1` domain prefix; HorizonFromEnvelope lifts a verified envelope into the `HorizonCertificate` shape (§13).
- Forward compatibility: a v2 receiver decodes v1 bytes via `ParseEnvelope` (PQ lanes empty). Backward compatibility: v1 receivers reject v2 bytes (the leading `0x02` is not a valid RLP-list prefix); senders that need to reach v1-only verifiers emit Warp 1.x bytes on the v1 channel.

The `WARP-PULSAR-ENVELOPE-v1` prefix is distinct from any consensus-side prefix (`QUASAR-PULSAR-BUNDLE-v1` `QUASAR-PULSAR-ACTIVATE-v1`, etc.) so a Pulse over a bundle root cannot be replayed as a Pulse over a cross-chain envelope.

7.6 C++ port

The C++ port lives at `luxcpp/crypto/corona/cpp/corona.{hpp,cpp}` [40] (691 LOC at LP-137-ACTUAL-STATE 2026-04-28). The C-ABI shim exposes `corona_setup`, `corona_sign`, `corona_verify` per LP-137 §3. The C++ body mirrors the Go canonical layer for layer:

- `corona/cpp/ring.hpp` replicates Lattigo's MRed, BRed, NTTStandard, INTTStandard on `uint64_t` with `__int128` extended multiplication.
- `corona/cpp/serialize.hpp` implements WriteTo in the wire format of §4, byte-for-byte.
- `corona/cpp/sampler.hpp` ports the Ziggurat tables (kn, wn, fn) verbatim and replicates the BLAKE2-XOF buffer-refill discipline.
- `corona/cpp/protocol.hpp` contains the protocol pseudocode of §3.

The cross-implementation Known-Answer-Test contract (§4) gates every release.

7.7 GPU kernels

GPU kernels live under `corona/gpu/{metal,cuda,wgsl}/` [40]. The dominant cost is $A \cdot \hat{R}$ in Round 1 ($m_{\text{pk}} \times n_{\text{vec}} \cdot n_{\text{vec}} \times (\bar{D}+1) = 8 \times 7 \cdot 7 \times 49$). The Lattigo SIMD-NTT [31] delivers 6–9 Mops/s of NTT throughput on Apple M2 Ultra at $N = 2^{20}$; per LP-164 [41] the GPU crossover threshold is $N^* = 1$ for FHE-class NTT, so Pulsar's per-poly NTT at $\varphi = 256$ crosses over at small but nontrivial K. Per-platform kernel sources, crossover-threshold sweeps, and benchmarks are tracked in LP-137 [40].

7.8 Test oracle

The Go canonical test suite is the byte oracle:

- `sign/sign_test.go`: per-component unit tests.
- `sign/local.go`: LocalRun, the end-to-end multi-party orchestration that the threshold-package [26] wraps.
- `threshold/threshold_test.go`: $K = 5$ end-to-end at $\mu =$ "Message".
- `utils/utils_test.go`: matrix/vector primitive coverage.

The KAT manifest (§4.7) is the cross-implementation gate.

7.9 Performance

Indicative timings on a single Apple Silicon M2 Ultra core, Go canonical:

Table 4: Phase timings, $K = 5$, $\mu =$ "Message". Numbers from `LocalRun(10)` median (`sign/local.go:21`).

Phase	Median (ms)
Gen	95
SignRound1 (per party)	185
SignRound2Preprocess (per party)	35
SignRound2 (per party)	22
SignFinalize	8
Verify	9

Round 1 is the dominant per-party cost; the C++ port at LP-137 closes the bulk of the matrix-multiply gap, and GPU acceleration is targeted to bring Round 1 below 50 ms at $K = 21$. The 3-second consensus interval of Lux mainnet (LP-020 [33]) accommodates the current Go canonical at $K \leq 21$ with an order-of-magnitude headroom.

8 Related Work

This section locates Pulsar within the cryptographic and systems literature on threshold signatures. We compare against (i) the contemporary lattice threshold-signature schemes (Threshold Raccoon, Hermine, two-round threshold ML-DSA), (ii) the curve-side dual (FROST / RFC 9591), (iii) the classical lifecycle layer (HJKY97 proactive refresh, Desmedt–Jajodia97 redistribution, Wong–Wang–Wing02 verifiable redistribution), and (iv) the standards landscape (NIST IR 8214C). The comparison criteria in Table 5 are: signature size at $K = 21$ participants, communication per signing session per participant, threshold support, native DKG support, native refresh / reshare support, post-quantum security level.

8.1 Lattice-based threshold signatures

Ringtail (S&P 2025) [7]. The academic ancestor. Two-round MAC-authenticated lattice threshold over $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q \approx 2^{48}$. The Lux Pulsar fork (`github.com/luxfi/pulsar`) inherits the Sign1/Sign2/Combine math byte-equal and adds the key-era lifecycle (LSS-managed generations, verifiable resharing, activation certificate, rollback). Pulsar replaces the upstream Feldman-style DKG (which is pseudoinverse-recoverable when commits omit the noise term) with two complementary genesis paths: a one-time foundation MPC ceremony or a Pedersen-style DKG over R_q with binding under MSIS on $[\mathbf{A} \mid \mathbf{B}]$ and hiding under MLWE on $\mathbf{B}$.

Threshold Raccoon (Eurocrypt 2024) [16]. Practical lattice threshold signatures from standard MLWE/MSIS at $\sim$ 13 KiB signature size and $\sim$ 40 KiB per-user communication. Targets up to 1024 signers. Provides a structured construction with DKG, identifiable abort, and non-interactive aggregation. Differences versus Pulsar: (i) Raccoon does not specify a permissionless-validator lifecycle (its paper assumes a static signing committee fixed at setup); (ii) Raccoon's signature size is $\sim$ 13 KiB per pulse versus Pulsar's $\sim$ 33 KB per pulse, but Raccoon's communication is higher per session due to its non-interactive structure requiring more broadcast material per signer; (iii) Raccoon's correctness proof and KAT vectors are at an earlier stage of standardisation than Pulsar's production fork. We treat Raccoon as the closest peer at the primitive layer; Pulsar's contribution is the deployment shape, not a displacement of Raccoon.

Hermine (NIST MPTC 2025) [52]. Threshold lattice signatures with DKG, identifiable aborts, and proactive refresh, presented at the NIST Multi-Party Threshold Cryptography project. Provides a complete primitive-layer package similar to Threshold Raccoon but with explicit refresh semantics. Differences versus Pulsar: Hermine targets the cryptographic-primitive standardisation track at NIST IR 8214C [50]; Pulsar targets a deployment shape that composes the primitive (any of Pulsar, Threshold Raccoon, or Hermine) with a permissionless-consensus lifecycle. The Pulsar adapter contract (§11) is primitive-agnostic at its boundary; in principle a Hermine-backed Pulsar is a drop-in alternative once Hermine's KAT vectors and constant-time verifier are pinned.

Two-round threshold ML-DSA [6, 15, 1]. Boneh, Eskandarian, Kim, Shih sketched a generic threshold construction over MLWE [6]; Damgård, Orlandi, Takahashi, Tibouchi produced the first concrete two-round threshold ML-DSA with rejection-sampling correctness [15]; [1] extended this with algebraic one-more LWE assumptions. Differences versus Pulsar: (i) the rounded-coefficient hint Δ in R_ν replaces ML-DSA's (α, β)-bounded h, simplifying the verifier's reconstruction; (ii) pairwise BLAKE3-keyed MACs replace the more expensive non-interactive zero-knowledge proofs of well-formedness on Round 1; (iii) the optimised $t = K$ Shamir path [39] avoids per-coefficient polynomial evaluation when the active set equals the threshold.

Threshold ML-DSA [12] (App. A [27]). The Celi–del Pino–Espitau–Niot–Prest construction targets threshold ML-DSA with masked shares and amortised rejection sampling, but the rejection-sampling circularity (App. A [27]) and the AGM/trapdoor assumptions in the proof keep the construction outside the QuasarCert 3.0 scope. QuasarCert 3.0 instead uses individual ML-DSA-65 signatures plus a Z-Chain Groth16 rollup for accountability, complementary to Pulsar's threshold lane.

Drijvers et al. on concurrent multi-signatures [18]. Showed that any threshold signature with concurrent execution and unauthenticated commitment broadcast admits a sub-exponential forgery on the underlying secret. Pulsar's MAC-authenticated Round 1 is the direct response. The MAC layer provides identifiable abort for Round 1 corruption (the failing MAC names its sender) and bounds the forgery advantage at $Q_M/2^{256}$ per Theorem 1.

8.2 Schnorr-family / curve-side threshold signatures

FROST [29, 28] and RFC 9591 [13]. Two-round threshold Schnorr with linear share aggregation, the elliptic-curve analog of Pulsar. RFC 9591 standardises the wire format for FROST over secp256k1, P-256, and ristretto255. Structural resemblance to Pulsar is intentional: both schemes use a Round 1 commitment broadcast plus a Round 2 message-bound open. Differences: (i) FROST uses a binding factor ρ_i derived from the commitment; Pulsar uses pairwise BLAKE3

MACs (256-bit symmetric) which provide stronger security against rogue-key style attacks; (ii) FROST's signature is 64 bytes regardless of K; Pulsar's is $\sim$ 33 KB per pulse; (iii) FROST is classical-only (broken by Shor), Pulsar is post-quantum.

The Lux production stack runs FROST on the secp256k1 control-plane signing for the bridge MPC (LP-019 [32]); Pulsar runs on the consensus-plane post-quantum lane. The sister kernel *Lens* (LP-103 [44]) extends FROST with the same LSS lifecycle as Pulsar, demonstrating that the LSS adapter contract is primitive-agnostic across curve and lattice.

8.3 ECDSA and CGGMP21

CGGMP21 [11]. The production ECDSA threshold standard. Requires Paillier encryption for the multiplicative-to- additive conversion in the per-party masked product computation and ships an extensive zero-knowledge proof suite to handle malicious aborts. The complexity translates into roughly an order of magnitude more bytes on the wire than FROST or Pulsar for the same (t, n). Pulsar's linear share aggregation eliminates the entire MtA layer; this is the fundamental advantage of moving from a multiplicative group $\mathbb{F}_p^*$ to an additive ring R_q. Pulsar and CGGMP21 are not redundant in Lux: CGGMP21 signs ECDSA-format bridge transactions for Ethereum-compatible counterparties; Pulsar signs PQ-format consensus blocks for the Quasar pipeline; the two coexist by jurisdiction.

8.4 BLS aggregate signatures

BLS12-381 [8, 38] aggregates non-interactively. The aggregate is 48 bytes regardless of K. BLS is faster, smaller, and simpler than Pulsar; it is the classical fast-path lane of QuasarCert 3.0. BLS is not post-quantum: a quantum oracle for the discrete log in $\mathbb{G}_1$ inverts the scheme. Pulsar is the PQ insurance against that adversary; the two lanes ship in parallel [33]. Quasar Horizon's hedging argument (App. E [27]) isolates the assumptions one per lane: DL/co-CDH on BLS12-381 (Lane 1, classical), MLWE/MSIS on Pulsar (Lane 2, post-quantum), MLWE/MSIS on ML-DSA-65 (Lane 3 inner, post-quantum). No known reduction transfers a break of one to a break of another.

8.5 Hash-based threshold signatures

SLH-DSA / SLH-DSA (FIPS 205, formerly SPHINCS+) [49, 5] provides hash-only post-quantum signatures. A threshold variant exists [14] but is not competitive with Pulsar on signing rate (SLH-DSA per-signature cost is dominated by Merkle path generation, hard to amortise across signers). Lux uses SLH-DSA [36] for cold- storage / archival signing where signing rate is irrelevant, and Pulsar for hot consensus signing where the $\sim$ 200 ms aggregate Round 1 cost dominates.

8.6 Classical lifecycle layer

HJKY97 proactive secret sharing [23]. Herzberg, Jarecki, Krawczyk, Yung introduced proactive secret sharing as a defence against mobile-adversary share accumulation: each epoch, the parties redistribute shares of 0 that sum into a fresh sharing of the same secret. Pulsar's Refresh operation (§3.8) is the lattice port of HJKY97 with lattice Pedersen commits replacing the original Feldman commits.

Desmedt–Jajodia97 redistribution [17]. Desmedt and Jajodia generalised proactive refresh to redistribution across distinct access structures: the old qualified subset Q samples polynomials with constant terms set to the old shares, the new committee aggregates Lagrange-weighted contributions, and the master secret is preserved. Pulsar's Reshare operation (§3.9) is

the lattice port of this construction. The Lagrange-recombination identity is equation (1) and is mechanised in `Crypto.Threshold.Reshare.reshare_preserves_public_key`.

Wong–Wang–Wing02 verifiable redistribution [54]. Wong, Wang, Wing introduced verifiable redistribution for archive systems: each contribution comes with a commit that proves the contribution is consistent with the polynomial without revealing the secret. Pulsar uses lattice Pedersen commits $\mathbf{C}_{i,j} = \mathbf{A} \cdot \mathrm{NTT}(g_i(\beta_j)) + \mathbf{B} \cdot \mathrm{NTT}(r_{i,j})$, with binding under MSIS on $[\mathbf{A} \mid \mathbf{B}]$ and hiding under MLWE on $\mathbf{B}$ (LP-073 §9.6).

8.7 Standards landscape

NIST IR 8214C [50]. The NIST Multi-Party Threshold Cryptography project surveys threshold ECDSA, threshold Schnorr, threshold ML-DSA, and threshold ML-KEM. Pulsar is positioned as a deployment-shape contribution rather than a primitive-layer candidate for IR 8214C: the underlying primitive (Pulsar / Threshold Raccoon / Hermine) is interchangeable, and Pulsar's contribution is the lifecycle that turns any of these primitives into a permissionless- consensus threshold signature. The Lux Core Team contributes the Pulsar deployment specification to inform IR 8214C's deployment-pattern documentation.

8.8 Comparison table

Table 5: Comparison of threshold signature schemes at $K = 21$ participants. "Sig size" = signature wire size; "Comm" = total communication per session per participant; "DKG" = native distributed key-generation support; "Refresh" = native HJKY-style proactive refresh; "Reshare" = native Desmedt–Jajodia redistribution to a new committee; "PQ" = post-quantum security under standard MLWE/MSIS or SHA-3 assumptions.

Scheme	Sig size	Comm	Threshold	DKG	Refresh	PQ
Pulsar (this paper)	$\sim$ 33 KB	$\sim$ 50 KB	$t \in [1, K]$	yes	yes	yes
Ringtail (academic) [7]	$\sim$ 33 KB	$\sim$ 50 KB	$t \in [1, K]$	Feldman	no	yes
Threshold Raccoon [16]	$\sim$ 13 KiB	$\sim$ 40 KiB	$t \in [1, K]$	yes	no	yes
Hermine [52]	$\sim$ 13 KiB	$\sim$ 40 KiB	$t \in [1, K]$	yes	yes	yes
2R-Th-Dilithium [15]	$\sim$ 30 KB	$\sim$ 50 KB	$t \in [1, K]$	no	no	yes
Threshold ML-DSA [12]	$\sim$ 5 KB	$\sim$ 8 KB	$t \in [1, K]$	no	no	yes*
FROST / RFC 9591 [13]	64 B	$\sim$ 200 B	$t \in [1, K]$	yes	no†	no
CGGMP21 ECDSA [11]	64 B	$\sim$ 50 KB	$t \in [1, K]$	yes	yes	no
BLS aggregate [8]	48 B	96 B	K-of-K	no	no	no
SLH-DSA-thr. [14]	$\sim$ 30 KB	$\sim$ 50 KB	$t \in [1, K]$	no	no	yes

* Threshold ML-DSA's PQ guarantee is conditional on the rejection-sampling resolution; see App. A [27].
† FROST refresh is provided by Lens (LP-103 [44]) via the LSS adapter contract.

8.9 Where Pulsar's contribution lives

The cryptographic primitives in this paper are not new: two-round MAC-authenticated lattice threshold signing [7, 16, 52], proactive secret sharing [23], secret redistribution [17], verifiable redistribution [54], and standardised PQ signatures [48, 49] all exist in prior work. What is new is the *composition*: a permissionless-consensus deployment architecture that

- operates the lattice threshold primitive over rotating validator sets without a per-epoch trusted dealer;
- gates each generation transition on an activation cert verifying under the unchanged group public key (Theorem 2);

- recovers from failed activation via a non-punitive rollback ladder (Lemma 13 of [27], "Rollback- lineage forward reconstruction") that does not require slashing-based attribution;
- scales to large validator sets via grouped Pulsar certificates (§12);
- composes BLS, ML-DSA, and Pulsar into a single Horizon finality artifact with explicit lane-by-lane verification (Theorem 5 of [27], "QuasarCert classical soundness");
- plugs into a universal MPC lifecycle framework (LSS) shared with curve-side schemes (Lens / FROST) and ECDSA (CMP).

The crypto exists. The systems layer that turns it into permissionless PQ-threshold consensus is Pulsar's contribution.

Acknowledgments

The Lux Core Team acknowledges the Ringtail authors for the protocol design [7], the Lattigo authors for the ring arithmetic and NTT primitives [31], the Threshold Raccoon [16] and Hermine [52] authors for the contemporary peer constructions whose comparison sharpened the Pulsar deployment scope, and the BLAKE2 / BLAKE3 [2, 51] authors for the hash and PRNG primitives that underlie every transcript construction.

9 Pulsar Key-Era Lifecycle

The static Pulsar signing primitive presupposes a one-shot key generation followed by a fixed signing committee. Permissionless chains do not have that luxury: validator sets rotate every epoch. Pulsar extends Pulsar with a *key-era lifecycle* that supports validator rotation without reconstructing the master secret and without requiring a trusted dealer after genesis.

The metaphor is load-bearing, not flavor. A pulsar (the cosmic object) is a rotating neutron star: the body persists, the beam direction rotates, observed pulses appear as the beam sweeps. In Pulsar:

- **Persistent body**: the group public key $(\mathbf{A}, \tilde{\mathbf{b}})$ and the hidden master secret $\mathbf{s}$ persist within a key era.
- **Rotating beam**: the share distribution $\{\mathbf{s}_i\}_{i \in V}$ rotates each epoch with validator-set changes.
- **Observed pulse**: each Quasar bundle emits one threshold certificate (a Pulsar *pulse*) signed under the unchanged group public key.

9.1 Three lineage fields, kept distinct

Every Pulsar share state carries three lineage scalars that must not be collapsed:

- **KeyEraID** (η). The group-key lineage. Bumps *only* at Reanchor, when a new group public key is established.
- **Generation** (g). The LSS resharing version within a key era. Bumps every Refresh or Reshare under the same $(\mathbf{A}, \tilde{\mathbf{b}})$.
- **RollbackFrom** (r). Audit lineage. $r = 0$ on ordinary forward transitions; $r > 0$ records the prior generation reverted from when the chain rolls back.

A trajectory across one key era and one Reanchor:

```
KeyEraID=7, Gen=0, Rb=0     genesis state of era 7
KeyEraID=7, Gen=1, Rb=0     reshared to a new validator set
KeyEraID=7, Gen=2, Rb=0     refreshed (same set; HJKY zero-poly)
KeyEraID=7, Gen=3, Rb=0     reshared again
KeyEraID=7, Gen=4, Rb=2     activation @gen 3 failed; reverted to gen 2
KeyEraID=8, Gen=0, Rb=0     reanchor: fresh group key, new era
```

9.2 Three layers, one shipping path

Layer	Operation	Trust
Bootstrap	Foundation MPC ceremony at chain launch, OR Pedersen DKG over R_q (§9.6)	One-time, observable; confined to one key era's launch.
Reshare	Old qualified subset $Q \subseteq O$, $\|Q\| \geq t_{\text{old}}$, runs $g_i(x)$ with $g_i(0) = s_i$; new party j computes $s'_j = \sum_{i \in Q} \lambda_i^Q \cdot g_i(\beta_j)$ [17]	No new trust. Master secret preserved structurally.
Reanchor	Fresh ceremony with new η	Same as Bootstrap; rare governance event.

9.3 Refresh vs Reshare: distinct primitives

Two primitives, never collapsed:

Refresh (HJKY97). Same committee, same threshold. Each party i samples a degree-$(t-1)$ zero-polynomial $z_i(x)$ with $z_i(0) = 0$, distributes $z_i(\alpha_j)$, and updates $s'_j = s_j + \sum_i z_i(\alpha_j)$. Defends against mobile-adversary share accumulation across time [23].

Reshare (Desmedt–Jajodia). New committee. Old qualified subset Q samples $g_i(x)$ of degree $t_{\text{new}} - 1$ with $g_i(0) = s_i$. Each new party j aggregates contributions weighted by Lagrange coefficients evaluated at 0 for Q. New polynomial $G(x) = \sum_{i \in Q} \lambda_i^Q \cdot g_i(x)$ satisfies $G(0) = \sum_{i \in Q} \lambda_i^Q \cdot s_i = s$ by Lagrange identity [17]. Verifiability per Wong–Wang–Wing [54].

Both primitives leave the public key unchanged: the lattice Pedersen-style commits $\mathbf{C}_k = \mathbf{A} \cdot \text{NTT}(c_k) + \mathbf{B} \cdot \text{NTT}(r_k)$ bind shares to the polynomial without revealing $\mathbf{s}$.

9.4 Activation certificate: the consensus circuit-breaker

After the resharing math finishes, the chain does *not* accept the new generation on faith. The new committee threshold-signs an activation message under the *unchanged* group public key:

```
QUASAR-PULSAR-ACTIVATE-v1 ||
    chain_id || network_id ||
    key_era_id || group_id ||
    old_epoch || new_epoch ||
    old_validator_set_hash || new_validator_set_hash ||
    old_threshold || new_threshold ||
    group_public_key_hash ||
    reshare_transcript_hash ||
    pairwise_material_commitment_hash ||
    implementation_version
```

If the threshold-aggregated signature verifies under $(\mathbf{A}, \tilde{\mathbf{b}})$, the new shares actually reconstruct $\mathbf{s}$ and the chain commits the new state. If verification fails, the chain stays at the old generation; the LSS rollback mechanism (§9.5) restores the prior snapshot.

What activation proves. Successful new signing capability under the unchanged GroupKey. That is all.

What activation does *not* prove. Which old party sent a bad contribution; which new party lied; whether all contributions were polynomial-consistent; whether a malicious old subset caused a failed activation. Those are responsibilities of the complaint and disqualification pipeline (commit / share verification under lattice Pedersen) and the slashing-evidence pipeline. Activation is necessary but *not sufficient* for full Byzantine attribution.

9.5 Rollback and the no-slashing failure ladder

Pulsar's failure response is graded and *non-punitive*. There is no slashing dependency:

1. Coordinator timeout / unavailable → next coordinator picked deterministically.
2. Resharing round fails (commit mismatch, bad share, etc.) → emit signed evidence; retry under fresh transcript binding next epoch.
3. Activation cert fails to verify under unchanged GroupKey → `Rollback(targetGeneration = current - 1)`; chain stays at the previous generation; signing continues.
4. Multiple consecutive activation failures → governance Reanchor: new η, fresh ceremony.

This ladder works for chains, bridges, enterprise custody, and validator committees alike, because none of the steps require identifying a malicious actor. Slashing evidence is *collected* during steps 2–3 (the framework persists signed messages from the failed run for forensic review), but the chain's liveness and safety do not depend on attribution.

9.6 Pedersen DKG over R_q: the trusted-dealer-free genesis path

The default Bootstrap path is a one-time foundation MPC ceremony, with toxic-waste assumption confined to genesis of one key era. As a parallel-track production option, Pulsar offers a Pedersen-style DKG over R_q (`pulsar/dkg2/`):

- Two uniform matrices $\mathbf{A}, \mathbf{B}$ derived deterministically from public seeds.
- Each party samples Gaussian polynomials f_i, g_i and broadcasts commits $\mathbf{C}_{i,k} = \mathbf{A}\cdot\mathrm{NTT}(c_{i,k}) + \mathbf{B}\cdot\mathrm{NTT}(r_{i,k})$.
- Hiding under MLWE on $\mathbf{B}$; binding under MSIS on $[\mathbf{A} \mid \mathbf{B}]$.

This avoids the upstream Pulsar Feldman-style DKG, which is pseudoinverse-recoverable: $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \cdot \mathbf{C}_k = c_k$ when commits omit the noise term. The Pedersen construction restores hiding by adding the $\mathbf{B}\cdot\mathrm{NTT}(r_k)$ mask.

The Pedersen-DKG path solves an open distributed-construction problem: jointly sampling $\tilde{\mathbf{b}} = \lfloor \mathbf{A}\cdot\mathbf{s} + \mathbf{e} \rceil_\Xi$ with the correct small-noise distribution, without any party learning $\mathbf{s}$ or $\mathbf{e}$, and producing a publicly verifiable proof of public-key well-formedness. Threshold Raccoon [16] and Hermine [52] are the closest related constructions; Pulsar's two production paths — foundation MPC ceremony and Pedersen DKG — are both viable, ship in parallel, and a chain operator chooses based on ceremony tolerance and validator-set provenance. Genesis ceremony is the lower-complexity path and ships first; Pedersen DKG eliminates the toxic-waste assumption entirely once its KAT vectors and constant-time verifier are pinned.

9.7 What is preserved, what changes

Quantity	Status across resharing within an era
$\mathbf{A}$ (public matrix)	PRESERVED
$\mathbf{s}$ (master secret)	PRESERVED structurally; share distribution rotates
$\tilde{\mathbf{b}}$ (rounded public key)	PRESERVED
GroupKey pointer (byte-identical)	PRESERVED
$\mathbf{e}$ (LWE error)	NOT preserved; used at genesis to form $\tilde{\mathbf{b}}$, then erased with dealer state. Signers only need shares of $\mathbf{s}$.
Share distribution $\{\mathbf{s}_i\}$	ROTATED
Pairwise PRF/MAC material	REGENERATED for the new committee via authenticated KEX (X25519 baseline; ML-KEM-768 hybrid recommended for PQ-aware deployments).
η (KeyEraID)	PRESERVED until Reanchor
g (Generation)	INCREMENTED
r (RollbackFrom)	0 on forward; > 0 on rollback

10 Pedersen-Style DKG over R_q (dkg2)

This section specifies the Pedersen-style distributed key generation protocol shipped at `github.com/luxfi/pulsa` the trusted-dealer-free genesis path that runs in parallel with the foundation MPC ceremony of §9.6. The construction replaces upstream Pulsar's Feldman commit $\mathbf{C}_k = \mathbf{A} \cdot \mathrm{NTT}(c_k)$ with a Pedersen commit hiding under MLWE on a second uniform matrix $\mathbf{B}$ and binding under MSIS on the wide concatenation $[\mathbf{A} \mid \mathbf{B}]$. Formal hiding and binding statements are given as Theorem 5 and Proposition 6; the Lean theorem statements referenced are `Crypto.Pulsar.DKG2.pedersen_hiding`, `Crypto.Pulsar.DKG2.pedersen_binding`, and `Crypto.Pulsar.DKG2.pedersen_identity` at `proofs/lean/Cry`

10.1 The broken Feldman commit

The upstream Pulsar DKG at `corona/dkg/dkg.go` uses Feldman commits

$$\mathbf{C}_k \;=\; \mathbf{A} \cdot \mathrm{NTT}(c_k) \;\in\; R_q^{m_{\mathrm{pk}}}, \qquad k = 0, \ldots, t-1, \tag{2}$$

where $\mathbf{A} \in R_q^{m_{\mathrm{pk}} \times n_{\mathrm{vec}}}$ is the Pulsar public matrix ($m_{\mathrm{pk}} = 8$, $n_{\mathrm{vec}} = 7$, $q = \texttt{0x1000000004A01}$, $\sigma_E = 6.108$), $c_k \in R_q^{n_{\mathrm{vec}}}$ is the Gaussian secret coefficient sampled from $\chi_E = \mathcal{D}(\sigma_E)$, and NTT is the number-theoretic transform on the cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$.

Because $\mathbf{A}$ is overdetermined ($m_{\mathrm{pk}} > n_{\mathrm{vec}}$) and NTT is unitary, the linear map $c_k \mapsto \mathbf{C}_k$ is injective with overwhelming probability ($1 - O(q^{-2})$ per slot, $1 - O(N \cdot q^{-2}) \le 1 - 2^{-89}$ overall). A passive observer of the public broadcast recovers c_k in closed form via the Moore-Penrose pseudoinverse, slot-by-slot:

$$\mathrm{NTT}(c_k)[s] \;=\; \big(\mathbf{A}[s]^\top \mathbf{A}[s]\big)^{-1} \mathbf{A}[s]^\top \cdot \mathbf{C}_k[s] \qquad s \in [0, 256). \tag{3}$$

Total cost: $\Theta(\varphi \cdot m_{\mathrm{pk}}^3)$ ring multiplications, a few milliseconds on a laptop. This is the headline finding of the red review at `luxcpp/crypto/corona/RED-DKG-REVIEW.md` (Finding 1) [24]. Public-key rounding $\tilde{\mathbf{b}} = \lfloor \sum_i \mathbf{C}_{i,0} \rceil_\Xi$ does not save the construction because the attacker has full-precision $\mathbf{C}_{i,k}$ for every i, k on the broadcast channel.

The empirical verification at `pulsar/dkg2/dkg2_test.go::TestDKG2_PseudoinverseAttack` runs this exact slot-wise pseudoinverse against the dkg2 commits and reports $1792/1792 = 100\%$ slot disagreement (against the broken-Feldman baseline of 0% disagreement, i.e. exact recovery).

10.2 The Pedersen construction

We define `dkg2` as the t-of-n protocol whose Round 1 broadcasts are

$$\mathbf{C}_{i,k} \;=\; \mathbf{A}\cdot \mathrm{NTT}(c_{i,k}) \;+\; \mathbf{B}\cdot \mathrm{NTT}(r_{i,k}) \qquad k = 0,\dots,t-1, \tag{4}$$

where $\mathbf{A},\mathbf{B} \in R_q^{m_{\mathrm{pk}}\times n_{\mathrm{vec}}}$ are *independent* public uniform matrices and $r_{i,k} \in R_q^{n_{\mathrm{vec}}}$ is a fresh Gaussian *blinding* vector sampled from the same χ_E as $c_{i,k}$. The matrices are derived deterministically from nothing-up-my-sleeve domain-separation tags via BLAKE3 of `"pulsar.dkg2.A.v1"` and `"pulsar.dkg2.B.v1"` respectively, so every party agrees on $\mathbf{A},\mathbf{B}$ without coordination (`pulsar/dkg2/dkg2.go` `DeriveA`/`DeriveB`).

The matrix derivation is deliberately HashSuite-independent — it uses a dedicated BLAKE3 path with its own version tag so KAT bytes stay stable across the Pulsar-SHA3 cutover. The Round 1.5 commit digest, by contrast, is bound to the active HashSuite (Pulsar-SHA3 in production; Pulsar-BLAKE3 retained for byte-equality with pre-cutover KATs). The suite ID is bound into the digest input so two suites can never collide on the same commit body (`pulsar/dkg2/dkg2.go` `CommitDigest`).

Each dealer i broadcasts $\{\mathbf{C}_{i,k}\}_{k=0}^{t-1}$ publicly and sends the share-pair

$$\big(f_i(j+1),\; g_i(j+1)\big) \;\in\; R_q^{n_{\mathrm{vec}}} \times R_q^{n_{\mathrm{vec}}} \tag{5}$$

to recipient j over the authenticated point-to-point channel, where $f_i(x) = \sum_k c_{i,k}x^k$ and $g_i(x) = \sum_k r_{i,k}x^k$ are the Shamir polynomials in the secret and the blinding respectively.

Round 1.5 (cross-party consistency). Before opening shares, each party broadcasts the digest

$$h_i \;=\; \mathrm{HashSuite.TranscriptHash}\big(\texttt{tagCommitDigest} \,\|\, \texttt{suite.ID()} \,\|\, \texttt{serialize}(\mathbf{C}_{i,0}) \,\|\, \cdots \,\|\, \texttt{serialize}(\mathbf{C}_{i,t-1})\big)$$

Recipients abort if any h_i they receive differs from the digest computed by other parties on the same broadcast. This closes the "different commits to different recipients" attack flagged as Finding 2 in [24]. Implementation: `Round1Output.CommitDigest()` (`pulsar/dkg2/dkg2.go`). The legacy BLAKE3-only digest path `CommitDigestBLAKE3()` is retained for byte-equal replay against the canonical KAT at `luxcpp/crypto/pulsar/dkg2/test/kat/dkg2_kat.json`.

Round 2 (verification + aggregation). Recipient j checks every sender $i \in [n]$:

$$\mathbf{A}\cdot \mathrm{NTT}\big(f_i(j+1)\big) \;+\; \mathbf{B}\cdot \mathrm{NTT}\big(g_i(j+1)\big) \;\stackrel{?}{=}\; \sum_{k=0}^{t-1} (j+1)^k \cdot \mathbf{C}_{i,k} \qquad (\mathrm{mod}\ q). \tag{6}$$

Equality is *exact*: both sides are $\mathbb{Z}_q$-linear and NTT is bijective. No norm tolerance is needed (contrast with naive "add fresh noise to every commit" fixes which require σ-dependent slack). Concretely the verification is one matrix-vector multiply over each of $\mathbf{A},\mathbf{B}$ plus a Horner accumulation in the NTT domain ($\Theta(t\cdot m_{\mathrm{pk}}\cdot n_{\mathrm{vec}})$ ring multiplications).

The verifier is constant-time: comparison runs an AND across all m_{pk} slots' coefficient blobs via `subtle.ConstantTimeCompare`; runtime is independent of how many slots differ. Implementation: `constTimePolyEqual` in `pulsar/dkg2/dkg2.go`; Lean reference `Crypto.Pulsar.DKG2.verifier_consta`

On success party j aggregates

$$s_j = \sum_{i=1}^{n} f_i(j+1) \;\in\; R_q^{n_{\mathrm{vec}}}, \tag{7}$$

$$u_j = \sum_{i=1}^{n} g_i(j+1) \;\in\; R_q^{n_{\mathrm{vec}}}, \tag{8}$$

which are valid (t, n)-Shamir shares of the master secret $s = \sum_i c_{i,0}$ and the master blinding $t_{\text{master}} = \sum_i r_{i,0}$ respectively. The Pedersen-shaped public key is

$$b_{\text{ped}} \;=\; \Big\lfloor \text{INTT}\Big(\sum_{i=1}^{n} \mathbf{C}_{i,0}\Big) \Big\rceil_{\Xi} \;=\; \big\lfloor \mathbf{A} \cdot s + \mathbf{B} \cdot t_{\text{master}} \big\rceil_{\Xi}, \tag{9}$$

rounded to $\Xi = 30$ bits as in the trusted-dealer Sign Gen path (`pulsar/sign/sign.go`).

10.3 Hiding (computational, MLWE on B)

Theorem 5 (Hiding)**.** *Assume the decisional Module Learning-With-Errors problem* $\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}$ *on the matrix* $\mathbf{B} \in R_q^{m_{pk} \times n_{vec}}$ *is hard. For any pair of secret distributions* $\mathcal{C}, \mathcal{C}'$ *over* $R_q^{n_{vec}}$*, the views*

$$\text{View}_{\mathcal{C}} = \big\{\mathbf{A}, \mathbf{B}, \{\mathbf{C}_{i,k}\}_{i\in[n],k\in[t]} \,:\, c_{i,k} \leftarrow \mathcal{C}\big\} \quad \textit{and} \quad \text{View}_{\mathcal{C}'} = \big\{\cdots \,:\, c_{i,k} \leftarrow \mathcal{C}'\big\}$$

are computationally indistinguishable, with adversary advantage at most $n \cdot t \cdot \text{Adv}_{\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}}$.

Game-based proof outline. We define a sequence of hybrid games $\text{H}_0, \text{H}_1, \ldots, \text{H}_{n\cdot t}$ and show consecutive hybrids are computationally indistinguishable under $\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}$.

- $\text{H}_0 = \text{View}_{\mathcal{C}}$ — the real protocol view.
- H_ℓ $(\ell = 1, \ldots, n \cdot t)$: for the first ℓ Pedersen summands $\mathbf{B} \cdot \text{NTT}(r_{i,k})$ in canonical (i, k) order, replace the summand with a fresh uniform $\mathbf{u}_{i,k} \in R_q^{m_{\text{pk}}}$. The remaining $n \cdot t - \ell$ summands are unchanged.
- $\text{H}_{n\cdot t}$: every $\mathbf{C}_{i,k}$ is now $\mathbf{A} \cdot \text{NTT}(c_{i,k}) + \mathbf{u}_{i,k}$ with $\mathbf{u}_{i,k}$ uniform.

Hybrid distance. For each $\ell \to \ell + 1$, the only change is one summand $\mathbf{B} \cdot \text{NTT}(r_{i,k})$ replaced with uniform $\mathbf{u}_{i,k}$. This is precisely a single decisional $\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}$ challenge: the secret is $r_{i,k} \sim \chi_E$, the public matrix is $\mathbf{B}$, the target is $\mathbf{B} \cdot \text{NTT}(r_{i,k})$ vs uniform. So $|\Pr[\mathcal{A} \text{ wins } \text{H}_\ell] - \Pr[\mathcal{A} \text{ wins } \text{H}_{\ell+1}]| \le \text{Adv}_{\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}}$.
Endpoint distinguishability. In $\text{H}_{n\cdot t}$ each $\mathbf{C}_{i,k}$ is $\mathbf{A} \cdot \text{NTT}(c_{i,k}) + \mathbf{u}_{i,k}$ with $\mathbf{u}_{i,k}$ uniform; this is uniformly distributed over $R_q^{m_{\text{pk}}}$ and independent of $c_{i,k}$. Thus $\text{H}_{n\cdot t}$ is identical for $\mathcal{C}$ and $\mathcal{C}'$.
Telescoping. Adversary advantage telescopes to $n \cdot t \cdot \text{Adv}_{\mathsf{MLWE}^{\mathbf{B}}_{q,\chi_E}}$, which is negligible under the standing MLWE hardness assumption at the Pulsar parameter set. □

The pseudoinverse attack of (3) fails as a direct corollary: applying $\mathbf{A}^+ := (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top$ to $\mathbf{C}_{i,k}$ yields

$$\mathbf{A}^+ \mathbf{C}_{i,k} \;=\; \text{NTT}(c_{i,k}) \;+\; \mathbf{A}^+ \mathbf{B} \cdot \text{NTT}(r_{i,k}),$$

where the second term is a $\mathbf{B}$-driven LWE mask. Without the corresponding $r_{i,k}$ the attacker recovers a uniform shift of $\text{NTT}(c_{i,k})$ in $\mathbb{Z}_q$. The empirical test `TestDKG2_PseudoinverseAttack` in `pulsar/dkg2/dkg2_test.go` confirms $1792/1792 = 100\%$ slot disagreement against $\text{NTT}(c_0)$ for a randomly sampled run (threshold for "hiding intact" is 99% slots disagreeing; observed margin is 100%). Lean reference: `Crypto.Pulsar.DKG2.pseudoinverse_attack_fails`.

10.4 Binding (computational, MSIS on $[\mathbf{A} \,|\, \mathbf{B}]$)

Proposition 6 (Binding)**.** *Suppose an adversary outputs two distinct openings* $(c_k, r_k) \neq (c'_k, r'_k)$ *of the same commitment* $\mathbf{C}_k$ *with* $\|c_k - c'_k\|_\infty$, $\|r_k - r'_k\|_\infty \le 2\beta_E$*. Then* $(c_k - c'_k,\, r_k - r'_k)$ *is a non-trivial bounded-norm element of the kernel of the wide matrix* $[\mathbf{A} \,|\, \mathbf{B}] \in R_q^{m_{pk} \times 2n_{vec}}$*, violating* $\mathsf{MSIS}_{q,[\mathbf{A}\,|\,\mathbf{B}],2\beta_E}$.

Reduction outline. Suppose $(c_k, r_k) \neq (c'_k, r'_k)$ both open $\mathbf{C}_k$. Then

$$\mathbf{C}_k = \mathbf{A} \cdot \mathrm{NTT}(c_k) + \mathbf{B} \cdot \mathrm{NTT}(r_k)$$
$$\mathbf{C}_k = \mathbf{A} \cdot \mathrm{NTT}(c'_k) + \mathbf{B} \cdot \mathrm{NTT}(r'_k)$$

Subtracting, $\mathbf{A} \cdot \mathrm{NTT}(c_k - c'_k) + \mathbf{B} \cdot \mathrm{NTT}(r_k - r'_k) = \mathbf{0}$, i.e., $[\mathbf{A} \,|\, \mathbf{B}] \cdot \binom{\mathrm{NTT}(c_k - c'_k)}{\mathrm{NTT}(r_k - r'_k)} = \mathbf{0}$. Since $(c_k, r_k) \neq (c'_k, r'_k)$, the difference vector is non-zero. The triangle inequality on the bound assumption gives $\| c_k - c'_k \|_\infty, \| r_k - r'_k \|_\infty \leq 2\beta_E$. Combining horizontally yields a non-trivial vector in $\ker_{\mathrm{small}}([\mathbf{A} \,|\, \mathbf{B}])$ of ℓ_∞-norm $\leq 2\beta_E$, which is exactly an $\mathsf{MSIS}_{q,[\mathbf{A} \,|\, \mathbf{B}],2\beta_E}$ solution. □

The matrix $[\mathbf{A} \,|\, \mathbf{B}]$ is fat ($m_{\mathrm{pk}} = 8 < 2n_{\mathrm{vec}} = 14$) so its $\mathbb{Z}_q$-kernel is generically non-trivial in unbounded norm; binding is therefore statistical only within the small-norm cone $\|c\|, \|r\| \leq \beta_E$ and computational otherwise. This matches the binding regime of standard BDLOP commitments [3] and recent lattice DKG schemes [22, 25]. Lean reference: `Crypto.Pulsar.DKG2.pedersen_binding`.

10.5 Pedersen identity (correctness across Lagrange recombination)

Theorem 7 (Pedersen identity)**.** *For honestly generated* $\{\mathbf{C}_{i,0}, s_j, u_j\}$ *and any* t*-subset* $T \subseteq [n]$ *with Lagrange weights* $\lambda_j = \prod_{k \in T, k \neq j} \frac{-(k+1)}{(j-k)} \pmod q$*, let* $s = \sum_{j \in T} \lambda_j s_j$ *and* $t_{\mathrm{master}} = \sum_{j \in T} \lambda_j u_j$*. Then*

$$\mathbf{A} \cdot \mathrm{NTT}(s) \;+\; \mathbf{B} \cdot \mathrm{NTT}(t_{\mathrm{master}}) \;\equiv\; \sum_{i=1}^{n} \mathbf{C}_{i,0} \pmod q. \tag{10}$$

Verification. Each commit expands as $\mathbf{C}_{i,0} = \mathbf{A} \cdot \mathrm{NTT}(c_{i,0}) + \mathbf{B} \cdot \mathrm{NTT}(r_{i,0})$. Summing over i, $\sum_i \mathbf{C}_{i,0} = \mathbf{A} \cdot \mathrm{NTT} \sum_i c_{i,0} + \mathbf{B} \cdot \mathrm{NTT} \sum_i r_{i,0}$. The Lagrange recombination identity $\sum_{j \in T} \lambda_j \cdot f_i(j+1) = f_i(0) = c_{i,0}$ for any degree-$(t-1)$ polynomial f_i (and similarly for g_i) gives $s = \sum_i c_{i,0}$ and $t_{\mathrm{master}} = \sum_i r_{i,0}$. Substituting yields (10). □

The identity is validated end-to-end at $t = 2, n = 3$ in `pulsar/dkg2/dkg2_test.go::TestDKG2_PedersenIde` every m_{pk}-slot of $\mathbf{A} \cdot \mathrm{NTT}(s) + \mathbf{B} \cdot \mathrm{NTT}(t_{\mathrm{master}})$ matches the corresponding slot of $\sum_i \mathbf{C}_{i,0}$ exactly. Lean reference: `Crypto.Pulsar.DKG2.pedersen_identity`.

10.6 Comparison to broken Feldman, GKS24, KRT24, Threshold Raccoon

Scheme	Hiding	Binding	
Broken `corona/dkg/` (Feldman, no noise)	*none* (linear)	statistical	
Naive "add Gaussian to commit"	MLWE on $\mathbf{A}$	statistical (norm bound)	1
GKS24 [22] (lattice GJKR)	MLWE on $\mathbf{A}$	MSIS on $\mathbf{A}$ (NIZK)	
KRT24 [25] (lattice JF-DKG)	MLWE on $\mathbf{A}$, Pedersen-aux	MSIS, structured	$2\times$
Threshold Raccoon [16] (single-round, noise-flooding)	MLWE on $\mathbf{A}$	statistical (flooded noise)	
This work (`dkg2`)	**MLWE on B**	**MSIS on $[\mathbf{A} \,|\, \mathbf{B}]$**	

Table 6: Pedersen `dkg2` versus alternatives. All four lattice DKGs share the same hardness regime; `dkg2` prioritises round count and verification simplicity over zero-knowledge identifiable-abort, and gets identifiable-abort by *sender index* via the signed complaint workflow at `pulsar/dkg2/complaint.go`.

`dkg2` pays a $2\times$ cost relative to the broken Feldman scheme ($\mathbf{A} \cdot c$ becomes $\mathbf{A} \cdot c + \mathbf{B} \cdot r$) but eliminates the key-recovery attack and avoids the noise-tolerance accounting that "add Gaussian to commit" fixes require. It is strictly weaker than GKS24/KRT24 in identifiability (the protocol

identifies a corrupt *sender* via Pedersen verification mismatch but does not produce a zero-knowledge proof of correct dealing); the Round 1.5 digest exchange covers the cross-party-inconsistency vector, and the signed complaint workflow turns every detection into slashable evidence. Threshold Raccoon [16] achieves a single round but at the cost of a noise-flooding accounting that grows with the validator-set size and a separate share-distribution channel; `dkg2` keeps the share distribution inline at 2× matvec cost.

10.7 Concrete parameters

The following parameters are pinned in `pulsar/sign/config.go` and inherited by `dkg2`:

φ	$2^8 = 256$
q	$\texttt{0x1000000004A01} = 281\,474\,976\,729\,601$ (48-bit prime)
m_{pk}	8
n_{vec}	7
σ_E	6.108187070284607
β_E	$2\sigma_E = 12.216374140569214$
Ξ	30 bits

These are the same lattice-attack-estimator parameters as upstream Ringtail [7]: at least 128-bit classical and at least 120-bit quantum security per the Albrecht primal/dual estimator. The matrix $\mathbf{B}$ inherits the same regime as $\mathbf{A}$ (uniform over R_q, dimension 8×7); the additional MLWE instance $(\mathbf{B}, \mathbf{B}\cdot\text{NTT}(r) + \text{noise})$ reuses the same σ_E, so the security-level analysis carries over verbatim. The fat-matrix MSIS on $[\mathbf{A}\,|\,\mathbf{B}] \in R_q^{8\times14}$ has dimension and norm bound parameters that the same estimator covers; binding holds within the small-norm cone $\|c\|, \|r\| \le \beta_E$ at the Pulsar hardness level.

10.8 Identifiable abort and the complaint workflow

Every detected misbehaviour during `dkg2` produces a signed complaint identifying the offending sender. The four reasons mirror `reshare.ComplaintReason` (`pulsar/reshare/complaint.go`) to keep one adjudication path for both protocols:

- **ComplaintBadDelivery** — sender i shipped a $(\text{share}_{i\to j}, \text{blind}_{i\to j})$ pair that fails (6). Evidence: the (share, blind, commits) triple. Re-checked via `VerifyShareAgainstCommits`.
- **ComplaintEquivocation** — sender i broadcast disagreeing commit-digest values to different recipients (Round 1.5 cross-check). Evidence: two signed digest broadcasts under sender i's wire identity that disagree.
- **ComplaintMissing** — sender i failed Round 1 by the cohort deadline. Evidence is the absence; a separate liveness round timestamps the deadline.
- **ComplaintMalformedCommit** — sender i's commit vector has wrong length / wrong dimension / nil entries. Evidence: the malformed commit vector.

Disqualification rule: a sender is disqualified iff at least $\max(1, t-1)$ distinct complainers signed valid complaints against it (`DisqualificationThreshold` in `complaint.go`). The $t-1$ cap is the corruption bound, so no honest sender can be disqualified by a maximally adversarial complainer set. After the Round 2 deadline, every honest party computes the same disqualified set D and the same surviving quorum $Q' = Q \setminus D$ via `FilterQualifiedQuorum`. If $|Q'| < t$ the protocol returns `ErrInsufficientQuorum` and the activation cert binds the abort transcript so the chain stays at the previous epoch.

Lean reference: `Crypto.Pulsar.DKG2.disqualification_below_corruption_bound` proves that `disqualificationThreshold(t) = t-1` for $t \ge 2$, matching the implementation.

10.9 Mapping to Pulsar Sign

Pulsar Sign (LP-073, §3) expects a public key of the form $\tilde{b} = \lfloor \mathbf{A} \cdot s + e \rceil_\Xi$ with e a fresh Gaussian. The Pedersen-shaped key $b_{\text{ped}} = \lfloor \mathbf{A} \cdot s + \mathbf{B} \cdot t_{\text{master}} \rceil_\Xi$ of (9) is structurally different — the noise term has the algebraic shape $\mathbf{B} \cdot t_{\text{master}}$ rather than free Gaussian. Three integration paths exist; we document the trade-offs and the recommended path.

Path (a) — Reflood public key, keep Sign unchanged. After Round 2 each party j broadcasts $\beta_j = \mathbf{A} \cdot \text{NTT}(\lambda_j s_j) + e'_j$ where λ_j are Lagrange weights for any t-subset $T \subseteq [n]$ and $e'_j \sim \mathcal{D}(\sigma'')$ is a fresh Gaussian with $\sigma'' = \kappa \sigma_E \sqrt{n}$ (we pin $\sigma'' = 23 \cdot 6.108 \cdot \sqrt{n}$ here, matching the η_ε slack reservation of LP-073 §5). Final key $b = \sum_{j \in T} \beta_j = \mathbf{A} \cdot s + e''$ with $e'' = \sum_{j \in T} e'_j$ is Pulsar-shaped. The L_2 slack budget for verification is then $\|e''\|_2 \le \sigma'' \sqrt{n \cdot m_{\text{pk}} \varphi} \approx 2^{42}$, well within Pulsar's $B^2 = 184\,960\,669\,042\,442\,604\,975\,662\,780\,477$.

Path (b) — Modify Sign verification to accept Pedersen pk. Replace Pulsar's verification identity $z = r^* + \sum_{j \in T} \lambda_j s_j c$ with the joint identity

$$z + z' \;=\; r^* + \sum_{j \in T} \lambda_j s_j c \;+\; \mathbf{B} \cdot \sum_{j \in T} \lambda_j u_j c,$$

i.e., promote the protocol to a 2-secret instance carrying both s_j and u_j. Every signing round transmits a z' companion to z and the verifier's public input is $(\mathbf{A}, \mathbf{B}, b_{\text{ped}})$. This is a protocol change to LP-073 that does not exist in any published canon; it preserves the elegance of the Pedersen DKG but invalidates the existing Pulsar Sign KAT contract.

Path (c) — Hybrid: Pedersen DKG, separate noise-flooding refresh. Run `dkg2` as specified. Discard u_j permanently after Round 2. Add a separate, single-round noise-flooding sub-protocol that takes the shares $\{s_j\}$ and produces a Pulsar-shaped $\tilde{b}$ via path (a)'s mechanism. This minimises the change to Pulsar Sign (no protocol edits) at the cost of one extra round.

Recommended path. Path (c) is the canonical production deployment for LP-073 because it

1. leaves Pulsar Sign and its KAT contract unchanged,
2. separates the cryptographic DKG from the noise-flooding refresh (orthogonality), and
3. allows the Pedersen DKG to be re-used by other Pulsar-class signers that may have different public-key conventions.

The integration test in this section exercises path (b) — the modified-verifier form — because it exposes the Pedersen identity (10) directly. The empirical validation at `pulsar/dkg2/dkg2_test.go::TestDKG2` closes the loop: dealer commits, distributed share aggregation, and Lagrange recombination all close end-to-end.

10.10 Implementation surface

- **Go canonical** (`pulsar/dkg2/dkg2.go`, `pulsar/dkg2/complaint.go`) — production-shipping. Round1WithSeed pins every byte of the protocol output for byte-equal C++ porting.
- **KAT oracle** (`pulsar/cmd/dkg2_oracle/main.go`) — emits 4 byte-stable entries (2-of-3, 3-of-5, 5-of-7, 7-of-11) at `luxcpp/crypto/pulsar/dkg2/test/kat/dkg2_kat.json`.
- **C++ surface** (`luxcpp/crypto/pulsar/cpp/dkg2/dkg2.{hpp,cpp}`) — production-stable header surface; algorithmic body via cgo to the Go canonical until the pulsar-namespaced lattice-ring extraction lands. Surface tests at `luxcpp/crypto/pulsar/test/dkg2/dkg2_surface_test.cpp` pin constructor validation, FFI-routing semantics, and the sentinel commit-digest contract.

- **Hash suite** (`pulsar/hash/`) — production digests use Pulsar-SHA3 (`cSHAKE256`/`KMAC256`/`TupleHash256`); legacy Pulsar-BLAKE3 retained for byte-equal KAT replay.
- **Negative tests**
  - `TestDKG2_PseudoinverseAttack` — 100% of NTT slots disagree with broken-Feldman recovery (target 99%).
  - `TestDKG2_TamperedShareRejected` — verification rejects bit-flipped shares.
  - `TestDKG2_Round2Identify` — identifiable abort names the offending sender.
  - `TestDKG2_MalformedCommitRejected` — wrong-length commit vector rejected with sender ID.
  - `TestDKG2_MissingDataRejected` — absent inputs rejected with `ErrMissingData`.
  - `TestDKG2_CommitDigestEquivocation` — Round 1.5 cross-check detects different commits to different recipients.
  - `TestDKG2_BadDeliveryComplaint` — re-checkable evidence packaged for slashing pipeline.
  - `TestDKG2_HashSuiteCrossProfile` — SHA3 and BLAKE3 digests cannot collide on the same commits.
  - `TestDKG2_DisqualificationThreshold`, `TestDKG2_FilterQualifiedQuorum` — disqualification arithmetic and quorum filtering.
- **Integration test** `TestDKG2_PedersenIdentity` verifies (10) end-to-end for the 2-of-3 protocol.

10.11 Lean theorem index

Theorem statements live in `proofs/lean/Crypto/Pulsar/dkg2.lean` and are referenced verbatim from this section without duplicating the proof bodies:

This section	Lean reference
Theorem 5	`Crypto.Pulsar.DKG2.pedersen_hiding`
Proposition 6	`Crypto.Pulsar.DKG2.pedersen_binding`
Theorem 7	`Crypto.Pulsar.DKG2.pedersen_identity`
(3) corollary	`Crypto.Pulsar.DKG2.pseudoinverse_attack_fails`
§10.2 (CT verifier)	`Crypto.Pulsar.DKG2.verifier_constant_time`
§10.8 (disqualification cap)	`Crypto.Pulsar.DKG2.disqualification_below_corruption_bound`
§10.9 (path-b)	`Crypto.Pulsar.DKG2.path_b_sign_verification`

11 The LSS-Pulsar Adapter

Pulsar's lifecycle (§9) is concrete; the *orchestration* of generations, rollback, snapshots, and dealer/coordinator role separation is provided by an external universal-MPC framework. We use **LSS** [53] — Lux's Linear Shamir Secret Sharing dynamic-resharing layer [39], originally specified for ECDSA-CMP and FROST and explicitly designed as a protocol-agnostic lifecycle extension.

The contract:

> *LSS owns generations, snapshots, rollback, and dealer/coordinator roles. Pulsar owns lattice math: R_q shares, lattice Pedersen commits, Λ/Seeds/MAC regeneration, and the activation certificate's lattice signature bytes.*

The wiring lives at `threshold/protocols/lss/lss_pulsar.go` — the analogue of `lss_cmp.go` (ECDSA) and `lss_lens.go` (curve, see LP-103 [44]). The adapter exposes:

```
func DynamicResharePulsar(
    oldConfigs map[party.ID]*PulsarConfig,
```

```
    newPartyIDs []party.ID,
    newThreshold int,
    pool *pool.Pool,
    rand io.Reader,
) (map[party.ID]*PulsarConfig, error)
```

It (1) validates that all old configs agree on KeyEraID, Generation, and GroupKey pointer; (2) reconstructs an in-process `KeyEra` from the union of old shares; (3) delegates all lattice math to the Pulsar kernel; (4) bumps Generation by one, preserves KeyEraID, sets RollbackFrom = 0 on forward transitions; (5) persists the resulting state through `PulsarSnapshotManager` for potential rollback.

11.1 Bootstrap Dealer vs Signature Coordinator

The LSS paper distinguishes two operational roles, both inherited by Pulsar:

Role	LSS responsibility	Pulsar / Quasar mapping
Bootstrap Dealer	Network membership manager. Orchestrates resharing. Never holds unencrypted secrets.	Foundation MPC node at chain genesis (Bootstrap). P-Chain governance authority for Reanchor. Holds no shares after ceremony close.
Signature Coordinator	Operational workhorse. Collects partial signatures. Combines certificates.	Quasar block proposer / per-epoch consensus runtime. Coordinates Sign1/Sign2/Combine for each Pulsar pulse over a bundle.

Crucially, the two roles are *distinct authorities*: a compromised Signature Coordinator cannot reshare or change the validator set (it has no membership-management authority), and a compromised Bootstrap Dealer at non-Bootstrap times cannot sign (it holds no shares). The role split is the operational basis for Pulsar's no-slashing-dependency failure model (§9.5).

11.2 Adapter contract guarantees

Preserved across the call. KeyEraID, GroupKey pointer, master secret $\mathbf{s}$ (held only as shares; never reconstructed in any process), and the persistent $(\mathbf{A}, \tilde{\mathbf{b}})$ public key.

Changed across the call. Generation (incremented by 1), participant set, threshold, share values, Λ_i Lagrange coefficients, pairwise PRF Seeds, MAC keys, transcript hash.

Failure behavior. Returns an error. The caller (LSS Rollback Manager or Quasar consensus) decides whether to retry, rollback, or reanchor. The adapter does not silently fall back.

11.3 What the adapter does *not* inherit

A clean adapter must reject ECDSA-specific assumptions baked into LSS-CMP and the curve-side LSS-FROST adapter (LP-103 [44]):

- ECDSA auxiliary secrets w, q for nonce blinding (LSS Section 4 protocols I, II). Corona-style signing has Gaussian blinding built into Sign1/Sign2; no w, q needed.
- Curve-point Pedersen commits $g^s \cdot h^r$. Pulsar uses lattice commits $\mathbf{A} \cdot \mathrm{NTT}(s) + \mathbf{B} \cdot \mathrm{NTT}(r)$ over R_q.
- ECDSA scalar serialization. Pulsar uses Lattigo `v7` polynomial serialization.
- Curve-specific Lagrange interpolation. Pulsar uses `primitives.ComputeLagrangeCoefficients` over $\mathbb{Z}_q$.

11.4 Acceptance tests

Ten acceptance tests gate the adapter (collapsed into six `Test*` functions):

1. GroupKey pointer preserved across resharing.
2. KeyEraID preserved across resharing.
3. Generation increments by exactly one on forward transition.
4. RollbackFrom $= 0$ on forward transition.
5. $t_{\text{old}} \neq t_{\text{new}}$ works.
6. Old set $\neq$ new set works (disjoint rotation supported).
7. Regenerated shares produce a valid Pulsar threshold signature.
8. Signature verifies under *unchanged* GroupKey, proving old shares are not needed after activation.
9. Pairwise PRF / MAC material is regenerated for exactly the new party set, with bytes that differ from the old committee's.
10. Rollback restores prior generation snapshot; restored state carries Generation $=$ current $+1$ and RollbackFrom $=$ current, preserving monotonicity while marking lineage.

All ten pass at the time of writing (`threshold/protocols/lss/lss_pulsar_test.go`).

12 Grouped Pulsar Certificates

A single global threshold key over thousands of validators is operationally brittle: every rotation requires every validator to participate in JVSS broadcasts, Pedersen-commit verification, and activation. Pulsar supports *grouped* certificates as the deployment shape for large validator sets.

12.1 Construction

Validators are partitioned into G groups of (approximate) size $k = N/G$. Each group has its own GroupKey lineage, its own KeyEraID, its own resharing schedule. A Pulsar bundle certificate at the consensus layer is a quorum of group certificates:

$$\mathsf{HorizonCert.Pulsar} = \left\{ (\mathsf{group}_j, \sigma_{\text{Pulsar},j}) \;:\; j \in \mathsf{quorum}(G) \right\}$$

with $|\mathsf{quorum}(G)| \geq \lfloor 2G/3 \rfloor + 1$ for the standard Byzantine bound, or a tighter quorum if individual groups are known to be more reliable.

12.2 What this buys

Local resharing. A group rotates its shares only when its own validators rotate; cross- group churn does not force every group to reshare. JVSS bandwidth scales as $\mathcal{O}(k^2)$ per rotation rather than $\mathcal{O}(N^2)$.

Independent failure domains. A failed activation in group j does not block other groups. The chain emits a Pulsar quorum-of-groups certificate from the surviving groups; group j rolls back independently.

Compact bundle pulses. Each group emits one Pulsar pulse (size on the order of the 13.4 KB Pulsar signature [26]). G groups produce G pulses; aggregating across groups via Merkle is straightforward. Total bundle-cert overhead is $\mathcal{O}(G)$, not $\mathcal{O}(N)$.

12.3 Open systems-research questions

Grouped Pulsar is *the* place where dynamic-PQ-threshold consensus becomes a research contribution rather than an engineering exercise. The open questions:

1. Given validator corruption fraction f, and group assignment process $\mathcal{G}$, what is the probability that the adversary controls enough *groups* to forge conflicting Horizon certificates? (Hypergeometric vs. binomial; balanced vs. adversarial assignment.)
2. How should group assignment respond to validator churn while preserving in-group share integrity?
3. How does group resharing interact with epoch transitions? (Per-group epochs vs. chain-global epochs; leader staleness.)
4. How many group certificates are required for a Horizon finality boundary at a given safety target?
5. What is the bandwidth/latency curve of Pulsar pulses as a function of group size k and group count G, under wide-area churn?

These questions connect cryptographic threshold assumptions to consensus-level quorum assumptions, and are the natural follow-on formal-systems paper to this one.

12.4 Comparison to Threshold Raccoon and Hermine

Threshold Raccoon [16] reports practical thresholds up to 1024 signers with ~ 13 KiB signatures and ~ 40 KiB communication per user. Hermine [52] provides DKG, identifiable aborts, and proactive refresh in a single package. Both target the cryptographic primitive layer.

Grouped Pulsar is orthogonal: it composes the underlying lattice threshold scheme into a deployment shape that scales across *permissionless validator sets* where committee membership is not fixed at protocol design time. The contribution is the *system-level* composition, not the lattice cryptography itself.

13 Pulsar in Quasar Horizon

Pulsar does not stand alone. It is the post-quantum threshold finality lane of *Quasar Horizon*, the Lux consensus stack (LP-020 [33]). Quasar reaches a finality boundary by composing three independent verification lanes plus optional Lens (LP-103 [44]) classical-threshold control-plane certificates.

13.1 The vocabulary stack

Name	Role	Per-validator
Photon	Individual validator vote / attestation	Own credential
Lumen	PQ encrypted+authenticated stream carrying Photons (forthcoming Lumen LP)	Own hybrid-KEM/ML-DSA + OTS
Beam	BLS aggregate certificate (LP-075 [38])	Own BLS keypair
Pulsar	PQ lattice threshold finality (this paper)	**Share** of one group key
Pulse	One Pulsar threshold certificate over a Nebula root	—
ML-DSA	PQ accountability cert set (LP-070 [35])	Own ML-DSA keypair
Lens	Classical curve threshold (LP-103 [44])	**Share** of one group key
Prism	Certificate-composition verifier (LP-105)	—
Horizon	Quasar finality boundary; Prism-bound hybrid finality	—
Quasar	Full leaderless consensus protocol (LP-020)	—

The canonical narrative (LP-105 [45]): *Nebula provides the DAG/GPU-native substrate. Photons are validator signals carried through Lumen streams and cohered into Beams. LSS moves threshold shares across generations. Lens emits classical threshold certificates; Pulsar*

emits post-quantum Pulses. Prism verifies that every certificate lane refracts from the same Nebula transcript. Quasar reaches the Horizon when finality is sealed.

13.2 Why three lanes (and not one)

The lanes have different security and operational profiles:

BLS Beam (independent keypairs, classical fast path). $\mathcal{O}(1)$-size aggregate over N validators. Sub-millisecond verify. Broken in polynomial time by Shor's algorithm; provides only a classical speed-up to a quantum adversary [18].[1] Used as the fast classical finality layer.

ML-DSA cert set (independent keypairs, PQ accountability). Each validator signs individually with FIPS-204 ML-DSA-65. The per- validator signatures may optionally be *compressed* via a Z-Chain Groth16 rollup (LP-307 [48, 35]); the per-block cert is then a $\sim$ 192-byte SNARK plus a public-inputs hash. Provides per-validator PQ *accountability* (i.e., who signed what). Cannot be aggregated into a single threshold signature without an honest dealer or formal threshold-ML-DSA construction; current work in active production-research has not standardized.

Important: Groth16 is not post-quantum. The optional Groth16 wrapper around the ML-DSA cert set is a *classical succinct proof of post-quantum signature verification*, not a PQ proof system. Pairing-based SNARKs rely on elliptic-curve discrete-log assumptions and break under Shor's algorithm. Pulsar's PQ finality flows from ML-DSA's Module-LWE/Module-SIS reduction and from Pulsar's Module-LWE reduction (Theorem 1); the Groth16 wrapper is a compatibility / compression / privacy adapter for EVM verifiers, not the PQ root of trust. See proof bucket `proofs/quasar/horizon-soundness.tex` Remark **??**.

Pulsar pulse (shared group key, PQ threshold finality). $\mathcal{O}(1)$-size compact PQ threshold certificate per bundle. Provides *post-quantum threshold* finality under the active key era's group public key. The right shape for asynchronous finality checkpoints over already-finalized BLS bundles.

Together. A Quasar Horizon certificate combines BLS Beam (immediate classical finality), ML-DSA cert set (per-validator PQ accountability + slashing-evidence material), and Pulsar pulse (compact PQ threshold finality over the bundle root). No single lane suffices; the combination is the contribution.

13.3 The Horizon certificate shape (Prism-bound)

```
type HorizonCertificate struct {
    Beam        bls.AggregateCertificate
    MLDSA       mldsa.CertificateSet      // signer bitmap + sigs (or
                                          //   Z-Chain Groth16 rollup)
    Pulsar      pulsar.Certificate        // the Pulse
    NebulaRoot  ids.ID                    // bound by all lanes
    KeyEraID    pulsar.KeyEraID
    GroupID     pulsar.GroupID            // partitioned-set deployments
    Generation  uint64
    HashSuiteID string                    // e.g., "Pulsar-SHA3"
```

[1] See LP-021 / pq-finality-no-bls [33] for the formal argument that BLS contributes no PQ security to a multi-lane cert.

```
}
```

Prism (LP-105 [45]) is the certificate-composition verifier. Its acceptance order:

1. Verify domain separation and transcript binding (every lane references the same Nebula root, hash-suite, KeyEraID, and Generation).
2. Verify BLS Beam aggregate.
3. Verify ML-DSA attestation set against signer bitmap (or Groth16 rollup against public-inputs hash).
4. Verify Pulsar Pulse under the active KeyEra/GroupKey.
5. Verify signer-set and validator-set hashes match epoch state.
6. Verify Nebula root / epoch / round linkage.

Horizon is not "BLS OR Pulsar." It is Prism-bound hybrid finality: every required lane refracts from the same transcript. A Beam by itself, or a Pulse by itself, may exist as an intermediate confirmation; only Prism's composite acceptance crosses the Horizon.

13.4 Asynchronous PQ finality

Pulsar pulses do not need to land in the same round as the BLS Beam. Quasar uses BLS for fast finality (sub-second), and Pulsar for delayed PQ finality over already-finalized bundles. This avoids pushing heavy two-round PQ threshold signing into every hot-path vote — the right design for permissionless deployment, where the WAN-distributed validator set cannot guarantee sub-second collective lattice signing.

A Quasar bundle aggregates several BLS-finalized blocks (e.g., ~ 6 blocks in ~ 3 seconds). The bundle root is what the Pulsar pulse signs. Even with substantial preprocessing, the online phase of the lattice 2-round signing comfortably fits within the 3-second bundle interval at $K = 21$ active validators per group.

13.5 Cert-tier degradation: measured 2026-06-03

Section 13.4 above motivates the asynchronous-finality design philosophically: BLS fast-path finality lands at one timescale, the Pulsar PQ tier lands at a second timescale, and a SLH-DSA Magnetar tier may land at a third. The 2026-06-03 distributed bench (`/tmp/multi-validator/`; harness source preserved at `harness.go` v0.2) directly measures this degradation schedule across the three host families that mirror a production Lux validator cluster (Apple-Silicon coordinator, NVIDIA Blackwell GPU server, x86_64 Linux). Five rounds per cell on a 3-host cluster (ra/spark/evo), 67% threshold ratio.

Scale	t_{BLS}	t_{Pulsar}	$t_{\text{Pulsar+Corona}}$	t_{Polaris}	Hybrid p50 finality
10v	159 ms	203 ms	1 135 ms	1 135 ms	1 135 ms
100v	162 ms	298 ms	2 593 ms	2 593 ms	2 593 ms
500v	187 ms	408 ms	2 655 ms	2 655 ms	2 655 ms

Table 7: Cert-tier arrival schedule in Hybrid-1s profile (BLS fast-path + Pulsar + Corona + Magnetar). t_{BLS} is when the BLS preliminary-finality cert is composable. t_{Pulsar} adds the Pulsar leg; $t_{\text{Pulsar+Corona}}$ adds the Corona leg; t_{Polaris} adds Magnetar. All values are p_{50} end-to-end (per-host sign + measured one-way propagation + compose) in milliseconds. The Polaris and full-Hybrid times are $\geq$ the Corona-leg time because Corona is the dominant leg in this CPU baseline; once Corona GPU acceleration lands (parity work in flight) Polaris-tier finality is expected to drop to ≤ 500 ms at the 100v scale.

Reading the schedule. A client may take preliminary classical finality at $t_{\text{BLS}} \approx 160{-}190\,\text{ms}$ (one RTT of BLS aggregate + propagation) and treat that block as mainline-confirmed for fast user paths. The PQ confirmation arrives at $t_{\text{Pulsar}} \approx 200{-}410\,\text{ms}$ across scales, upgrading the cert to PQ-finality on the ML-DSA-compatible lattice lane. The Aurora tier (+Corona) and Polaris tier (+Magnetar) follow after the Corona leg's contribution; in the CPU baseline these are 1.1–2.7 s. The degradation is monotone: a client that accepts the BLS tier and later sees the Aurora/Polaris tier never sees a contradiction (the cert composer is pure).

Cluster topology. ra: Apple M1 Max (substitute for dbc/M4 Max during this measurement window). spark: Ubuntu 24 NVIDIA Blackwell arm64. evo: AMD Ryzen AI MAX+ 395 x86_64 under WSL2. Coordinator on ra; partial-signature legs collected via HTTP/JSON-RPC over SSH tunnels (a conservative substitute for QUIC 0-RTT; production deployment is expected to be one network RTT faster). Full numbers and caveats in LP-020 Section **??**.

13.6 Where the systems contribution lives

The crypto primitives in this paper are well-established:

- Two-round MAC-authenticated lattice threshold signing [7, 16, 52].
- Proactive secret sharing [23].
- Secret redistribution to new access structures [17].
- Verifiable redistribution [54].
- Standardized PQ signature for individual accountability [48].
- BLS aggregate over independent keypairs [8].

What is novel in this work is the *composition*: a permissionless- consensus deployment architecture that

1. operates the lattice threshold primitive over rotating validator sets without a per-epoch trusted dealer;
2. gates each generation transition on an activation certificate verifying under the unchanged group public key;
3. recovers from failed activation via a non-punitive rollback ladder that does not require slashing-based attribution;
4. scales to large validator sets via grouped Pulsar certificates;
5. composes BLS, ML-DSA, and Pulsar into a single Horizon finality artifact with explicit lane-by-lane verification;
6. plugs into a universal MPC lifecycle framework (LSS) shared with curve-side schemes (Lens / FROST) and ECDSA (CMP).

The crypto exists. The systems layer that turns it into permissionless PQ-threshold consensus is the contribution.

References

[1] Anonymous. Two-round threshold signature from algebraic one-more learning with errors. In *EUROCRYPT*. Springer, 2024. Concrete two-round threshold Dilithium analogue cited for comparison.

[2] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. RFC 7693, 2015.

[3] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In *SCN*, 2018.

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

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

[6] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, pages 565–596. Springer, 2018.

[7] 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).

[8] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, 2017.

[9] 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.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*. IEEE, 2001.

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

[12] Andrea Celi, Rafaël del Pino, Thomas Espitau, Guilhem Niot, and Thomas Prest. Efficient threshold ML-DSA. In *USENIX Security*, 2025.

[13] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. RFC 9591: The flexible round-optimized schnorr threshold (frost) protocol for two-round schnorr signatures. RFC Editor, 2024.

[14] Andrei Cosmin and Dmitry Khovratovich. Threshold SPHINCS+: A hash-based threshold signature. In *SAC*. Springer, 2021.

[15] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *PKC*. Springer, 2021.

[16] Rafael del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Mark-Henry Servera. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. Eurocrypt, 2024.

[17] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. In *Technical Report ISSE TR-97-01, George Mason University*, 1997.

[18] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Gregory Neven, and Igors Stadler. On the security of two-round multi-signatures. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1084–1101. IEEE, 2019.

[19] 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.

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

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

[22] Rosario Gennaro, Hugo Krawczyk, and Amit Sahai. Lattice-based gjkr distributed key generation with nizk. In *ACNS*, 2024.

[23] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, 1995. Updated 1997.

[24] Lux Industries Inc. Red-dkg-review: Internal red review of the corona feldman dkg. luxcpp/crypto/corona/RED-DKG-REVIEW.md, 2026.

[25] Shuichi Katsumata, Markus Reichle, and Kaoru Takemure. Lattice-based joint-feldman distributed key generation. In *ASIACRYPT*, 2024.

[26] 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 ([7]).

[27] Zach Kelling. QuasarCert soundness: Canonical formal specification, Quasar 3.0. Lux Industries; `lux/proofs/quasar-cert-soundness.tex`, 2026.

[28] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In *SAC*, 2020.

[29] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. RFC 9591, 2024.

[30] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, 2015.

[31] Lux Core Team. Lattigo v7: A lattice-based cryptography library (lux fork). https://github.com/luxfi/lattice, 2026. Fork of `tuneinsight/lattigo`; pinned at v7 for byte-exact reproducibility.

[32] Lux Core Team. Lp-019: Threshold mpc for bridge signing. https://github.com/luxfi/lps/blob/main/LP-019-threshold-mpc.md, 2026.

[33] Lux Core Team. Lp-020: Quasar consensus 3.0. https://github.com/luxfi/lps/blob/main/LP-020-quasar-consensus.md, 2026.

[34] Lux Core Team. Lp-029: Ntt transform. https://github.com/luxfi/lps/blob/main/LP-029-ntt-transform.md, 2026.

[35] Lux Core Team. Lp-070: Fips 204 ml-dsa signatures. https://github.com/luxfi/lps/blob/main/LP-070-mldsa.md, 2026.

[36] Lux Core Team. Lp-071: Fips 205 slh-dsa hash-based signatures. https://github.com/luxfi/lps/blob/main/LP-071-slhdsa.md, 2026.

[37] Lux Core Team. Lp-073: Corona lattice-based threshold signatures. https://github.com/luxfi/lps/blob/main/LP-073-corona.md, 2026.

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

[39] Lux Core Team. Lp-077: Linear shamir's secret sharing. https://github.com/luxfi/lps/blob/main/LP-077-linear-shamir.md, 2026.

[40] Lux Core Team. Lp-137: Gpu-native crypto stack. https://github.com/luxfi/lps/blob/main/LP-137.md, 2026.

[41] Lux Core Team. Lp-164: Six-step ntt and crossover thresholds. https://github.com/luxfi/lps/blob/main/LP-164-six-step-ntt.md, 2026.

[42] 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 ([7]); cited via this Lux artifact, never via the academic paper. Q-Chain Feldman/Pedersen DKG.

[43] Lux Industries. Lean4 cryptographic proof tree. https://github.com/luxfi/proofs/tree/main/lean/Crypto, 2026. Mechanised companion to `quasar-cert-soundness.tex`; canonical theorem names cited verbatim from this tree.

[44] Lux Industries. LP-103: Lens — curve-based threshold signatures with dynamic resharing. Lux Improvement Proposal, 2026.

[45] Lux Industries. LP-105: The lux finality stack — public vocabulary and internal names. Lux Improvement Proposal, 2026.

[46] George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

[47] NIST. Fips 203: Module-lattice-based key-encapsulation mechanism standard. https://csrc.nist.gov/pubs/fips/203/final, 2024.

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

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

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

[51] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. https://github.com/BLAKE3-team/BLAKE3, 2020.

[52] NIST Multi-Party Threshold Cryptography Project. Hermine: Threshold lattice signatures with dkg and proactive refresh. NIST MPTC presentation, 2025.

[53] Vishnu J. Seesahai and Zach Kelling. LSS: A pragmatic framework for dynamic and resilient threshold signatures with live secret resharing. Lux Industries, Inc., 2025.

[54] Theodore M. Wong, Chenxi Wang, and Jeannette M. Wing. Verifiable secret redistribution for archive systems. In *SISW*, 2002.

Reproducibility

Source code. `github.com/luxfi/pulsar` at tag `v1.0.23`; LSS adapter at `github.com/luxfi/threshold`; Quasar composition at `github.com/luxfi/consensus`.
Build. `go build ./...` and `go test ./...` at `v1.0.23`.
Production threshold path. The production threshold-signing path is the v0.3 algebraic-aggregate construction: party shares contribute polynomial vectors directly to a Lagrange-combined ML-DSA witness, producing signatures byte-identical to single-party FIPS 204 ML-DSA. The v1.0.20 release closed the last refinement axiom by repairing the public-matrix expansion in `deriveKeyMaterial` so **A** is taken directly in NTT domain, eliminating the double-NTT that diverged threshold output from single-party FIPS 204. The v1.0.23 release introduced the canonical wire codec (`PULS` signatures, `PULG` group keys) with stateless `VerifyBytes`; the headline cross-check is `TestPulsar_Wire_FIPS204Verifiable`, which decodes a Pulsar wire signature and verifies it via the reference FIPS 204 verifier on the corresponding ML-DSA public key.
Hardware tested. Commodity x86_64 Linux; AVX2 optional for NTT; Apple Silicon Metal and CUDA dispatch via `luxfi/lattice/v7`.
Standards referenced. FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), FIPS 205 (SLH-DSA), RFC 9591 (FROST), RFC 7693 (BLAKE2), NIST IR 8214C (MPTC).
Contact. security@lux.network.