Magnetar: Per-Validator Standalone FIPS 205 SLH-DSA for Lux Public-BFT Consensus

Zach Kelling

Lux Industries

2026Artefact: `luxfi/magnetar` v1.0.0

Abstract

We specify *Magnetar*, the hash-based post-quantum signature primitive of the Lux Polaris cert profile. Magnetar's headline primitive is a *per-validator standalone* construction: each validator i holds an independent FIPS 205 SLH-DSA keypair (sk_i, pk_i) [41], produces a single-party deterministic signature σ_i on the consensus message, and the consensus layer collects N signatures into a *ValidatorAggregateCert* that verifies under any unmodified FIPS 205 verifier. There is no distributed key generation, no dealer, no aggregator, and no threshold reconstruction step. Each σ_i is byte-identical to what a single-party FIPS 205 SignDeterministic would produce on (sk_i, μ, ctx) — this is the *byte-identity claim* pinned by TestMagnetar_Wire_FIPS205Verifiable (`ref/go/pkg/magnetar/wire_test.go`:166–220), which routes through the `ValidatorSign` primary primitive, strips the canonical MAGS/MAGG wire envelope to recover the raw FIPS 205 payload, and feeds it to `cloudflare/circl/sign/slhdsa` [17] with no Magnetar code path on the verifier side. All three SHAKE parameter sets (SHAKE-192s, SHAKE-192f, SHAKE-256s) pass under `-count=1` and `-race`.

Why per-validator and not threshold. SLH-DSA is hash-based and admits no FROST-style linear share-aggregation identity [19, 6, 41]. Cozzo-Smart (EUROCRYPT 2019) and Bonte-Smart-Tan (2023) establish that every internal SHAKE/SHA-2 evaluation in an HBS scheme is a non-linear function of the secret seed; producing a single FIPS 205-shaped signature from t shares requires either reconstructing the seed (a dealer or aggregator in the trusted computing base) or running a full multi-party computation over the SHA-256/SHAKE hash tree (open research, on the order of 10^4 hash invocations per signature under MPC). For *public-BFT* consensus the architecturally correct primitive is therefore N standalone signatures collected at the consensus layer; the per-validator-standalone construction shipped here is the canonical hash-based instantiation of that pattern.

Composition. The $N \times |\sigma|$ on-wire cost of a $K = 21$ Lux Polaris quorum at SHAKE-192s is $\sim$ 332 KiB. The Z-Chain Groth16 rollup compresses this to $\sim$ 192 B before the certificate hits long-term storage. The rollup is a separate primitive (lux-zchain spec); the Magnetar wire codec presents the parallel-slices `(Signers, PubKeys, Sigs)` shape that maps directly onto the rollup witness layout.

The companion artefact is `luxfi/magnetar` v0.5.2 [36], with the wire codec MAGS/MAGG and the stateless `VerifyBytes` dispatcher landing in commit `1c84519` on 2026-05-31. The reference implementation is pure Go, backed by `cloudflare/circl/sign/slhdsa` v1.6.3 [17] for the underlying FIPS 205 primitive, with the same constant-time posture the circl maintainers ship. Magnetar joins Pulsar (Module-LWE) [25] and Corona (Ring-LWE) [27] as the three independent legs of the Lux Polaris maximum-assurance post-quantum cert profile; the three primitives draw on algorithmically distinct families (M-LWE / R-LWE / hash-based) so no single algorithmic break collapses the conjunction.

THBS-SE: permissionless threshold companion. For deployments that genuinely need a *single* FIPS 205-shaped signature from a t-of-n committee — e.g. verifier-side constant-cost certificates, smart-contract verification with fixed gas cost, cross-chain bridging into systems that recognise only one signature per message — Magnetar v1.0 ships *THBS-SE* (Threshold Hash-Based Signatures with Selected-Element Reconstruction; §5). THBS-SE is a permissionless t-of-n construction with a PUBLIC COMBINER role: any peer can run Combine; no host is in the TCB by policy at sign time. The construction enforces the hard invariant "a revealed value is allowed only if it is also present in the final SLH-DSA signature" (named Theorem 4) and emits FIPS 205 wire bytes byte-identical to single-party SignDeterministic on the reconstructed seed (Theorem 5). Eight test gates and a deterministic $(n = 7, t = 4)$ KAT pin the construction. v1.0 ships an honest open item on transient seed reconstruction at the public combiner; the strict refinement (no seed reconstruction even transiently) is the v1.1 strict-atom-assembly lift tracked at `BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11`.[1]

[1] Operator-controlled MPC custody uses TEE-attestation-gated variants in `luxfi/threshold/protocols/{slhdsa,mldsa,rlwe}-tee` shipped via the `lux/mpc` sibling tree; out of scope for this paper.

Contents

1 Introduction

Post-quantum threshold signatures are the load-bearing primitive of any blockchain consensus design that wants to survive a cryptographically relevant quantum adversary. The classical workhorses — BLS aggregate [13, 9] and ECDSA with CGGMP21 [15] — fail catastrophically under Shor's algorithm, and the threshold replacements available for the post-quantum era do not all ground their security in the same underlying mathematical assumption. Defense in depth at the cryptographic primitive layer requires not just "a PQ scheme," but a portfolio of PQ schemes whose security assumptions are *algorithmically independent*: a break of any one family does not propagate to the others.

The Lux Polaris cert profile [32] encodes this discipline in production. Polaris is the maximum-assurance Quasar finality profile that pairs three independent threshold lanes:

- **Pulsar** — threshold Module-LWE [25, 12], the lattice lane in $\mathbb{Z}_q[X]/(X^{256}+1)$ with a 48-bit prime;
- **Corona** — threshold Ring-LWE [27, 12], a structurally distinct lattice instantiation that does not share Pulsar's module parameter choice; and
- **Magnetar** (this paper) — per-validator standalone FIPS 205 SLH-DSA [41], the hash-based lane whose security rests on SHA-3 / Keccak collision resistance and preimage resistance rather than on any structured lattice assumption.

The three legs are deliberately drawn from *different mathematical families*: a polynomial-time algorithm against module-lattice problems would break Pulsar and Corona simultaneously but leave Magnetar intact; a polynomial-time preimage attack on Keccak would break Magnetar but leave the lattice lanes intact; a single quantum oracle that defeats both families is, on current evidence, an inconsistent ask. Polaris finality requires *all three* lanes to verify, so an adversary must break the conjunction — a strictly harder task than breaking any single primitive.

Magnetar is the third leg. This paper specifies the construction at `github.com/luxfi/magnetar` v1.0.0 — the per-validator standalone public-BFT primary primitive, the THBS-SE permissionless threshold companion, the canonical MAGS/MAGG wire codec, and the stateless `VerifyBytes` dispatcher — and the Polaris-composition discipline that binds Magnetar to Pulsar and Corona under a common transcript.[2]

1.1 What Magnetar is

Magnetar's primary cryptographic primitive is *per-validator standalone FIPS 205 SLH-DSA* [41]. SLH-DSA is the NIST hash-based signature standard (formerly SPHINCS+ [6]); it is stateless, requires no trapdoor or structured ring, and inherits its security from the standard properties of cryptographic hash functions (collision resistance, preimage resistance, second-preimage resistance, and the Boneh–Shoup [10] sub-exponential preimage bound under quantum adversaries).

The Magnetar public-BFT construction departs deliberately from the threshold templates of FROST [30, 18], Pulsar [25], or Corona [27]. SLH-DSA has *no linear share-aggregation identity*: there is no algebraic operation analogous to FROST's $z = \sum_i z_i$ that produces a valid SLH-DSA signature from per-party contributions on the same message. The literature is uniform on this point. Cozzo and Smart [19] (EUROCRYPT 2019, "Sharing the LUOV") establish that every internal SHAKE/SHA-2 evaluation in a hash-based signature is a non-linear function of the seed, so any genuine t-of-n scheme producing a single FIPS 205-shaped signature requires either (i) reconstructing the seed inside some combiner process at sign time, or (ii) running a full MPC over the entire SHA-256/SHAKE hash tree (open research, with cost proportional to the $\sim 10^4$ hash invocations per SLH-DSA-SHAKE-192s signature). Bonte, Smart and Tan

[2] Operator-controlled MPC custody uses TEE-attestation-gated variants in `luxfi/threshold/protocols/{slhdsa,mldsa,rlwe}-tee` shipped via the `lux/mpc` sibling tree; those variants are out of scope here.

(2023, "Threshold SPHINCS+") confirm the infeasibility analysis. THBS-SE (§5) is Magnetar's chosen instantiation of (i) with a *public combiner* role: anyone can be the combiner; no host is in the TCB at sign time; the seed reconstruction is transient and local to the combiner's own substrate.

The architecturally correct primitive for *public-BFT consensus* under these constraints is therefore N standalone signatures collected at the consensus layer. Each validator i runs FIPS 205 KeyGen once, persists (sk_i, pk_i), and contributes a single-party $\text{Sign}_{\text{det}}(sk_i, \mu, ctx)$ to each consensus message. The consensus layer collects the N signatures into a `ValidatorAggregateCert` carrying the parallel triples (*Signers*, *PubKeys*, *Sigs*) and a known validator registry binding each *Signers*$[i]$ to its registered pk_i. The quorum decision ("count $\geq$ threshold") lives at the consumer, not in the primitive; the primitive reports the validated count.

Byte-identity to single-party FIPS 205. Each σ_i emitted by `magnetar.ValidatorSign` (with `rng=nil`) is byte-identical to what a centralized FIPS 205 SignDeterministic would emit on the same (sk_i, μ, ctx). This is the load-bearing operational property: every SLH-DSA verifier in the wild (`cloudflare/circl/sign/slhdsa` [17], the NIST FIPS 205 reference, openssl-pq builds, libsignal-pq) accepts a Magnetar signature byte-for-byte after the consensus wire envelope is stripped. The byte-identity is empirically pinned by TestMagnetar_Wire_FIPS205Verifiable at `ref/go/pkg/magnetar/wire_test.go:166-220`: the test generates a keypair via `PerValidatorKeypair`, signs via `ValidatorSign`, frames both via the MAGS / MAGG wire codec, strips the 11-byte canonical header to recover the raw FIPS 205 payload bytes, and feeds them to `cloudflare/circl/sign/slhdsa` with no Magnetar code path on the verifier side. The test passes for SHAKE-192s, SHAKE-192f, and SHAKE-256s under `-count=1` and `-race`.

Public-BFT contract. The per-validator standalone primitive carries the property the cryptographic-diversity argument of §1.2 requires:

1. No DKG, no dealer, no shared seed. Each validator generates its own keypair on its own host with its own RNG. Losing sk_i is a local failure — the validator drops out of the quorum until rotating to a fresh keypair — not a threshold-secret reconstruction event.
2. No aggregator-in-TCB. The consensus layer never holds anyone's secret. `ValidatorBatchVerify` and `VerifyAggregateCert` are pure public-input functions.
3. No new verification code. Relying parties run any unmodified FIPS 205 verifier against the wire payload; the Magnetar envelope is opaque to them.
4. Wire shape that maps directly to Z-Chain Groth16 rollup witness layout. The `ValidatorAggregateCert` parallel-slices form (*Mode*, *Signers*, *PubKeys*, *Sigs*) is the wire shape the rollup prover consumes verbatim.

The wire-size cost is $N \times |\sigma|$ before the rollup. For SHAKE-192s ($|\sigma| = 16{,}224$ B) at a $K = 21$ Polaris quorum that is $\sim$ 332 KiB. The Z-Chain Groth16 rollup compresses this to $\sim$ 192 B before the certificate hits long-term storage. The rollup is a separate primitive specified at the lux-zchain layer; Magnetar emits the rollup-friendly parallel-slices shape but does not own the rollup.

1.2 Why hash-based

Lattice cryptography (Module-LWE, Ring-LWE, NTRU) is the consensus post-quantum direction for performance reasons — ML-KEM [39], ML-DSA [40], and FALCON [23] all operate in milliseconds, ship signatures in the few-KB range, and parallelise well. But the lattice family rests on a single set of related-problem assumptions (Module-LWE, Ring-LWE, NTRU, Module-SIS) whose hardness is conjectured but whose ultimate security margin is the subject of ongoing cryptanalytic refinement. The cryptanalytic surface includes BKZ block-size cost models [5, 31], dual sieves [1], structured-lattice algebraic attacks on cyclotomic rings, and quantum-amplified

sieves whose constants remain in modest motion. A polynomial-time algorithm for any of these problems would invalidate the entire lattice signature ecosystem simultaneously.

Hash-based signatures rest on assumptions that are *older, more broadly studied, and structurally orthogonal*. FIPS 205 SLH-DSA's security reduces to:

- collision resistance of the hash function family (SHA-3 / SHAKE for the FIPS 205 SHAKE modes Magnetar targets);
- preimage resistance of the same;
- the absence of unforeseen structural weaknesses in the WOTS+ and FORS one-time / few-time signature schemes that build on the hash function.

These properties have been studied for decades. SHA-3 in particular emerged from a multi-year open competition [7]; its internal structure (Keccak's sponge construction over a permutation) is qualitatively distinct from Merkle–Damgård, BLAKE, and the keyed-permutation Romulus and Ascon families. A quantum attack that defeats SHA-3 is one that defeats hash functions *as a class*, a strictly stronger ask than defeating any one family.

The trade-off is performance: SLH-DSA signatures are 16–36 KB, signing takes hundreds of milliseconds to seconds on commodity hardware, and the construction is the slowest of the FIPS-standardised PQ schemes. For a *single* hot-path signature this would be a non-starter. For the *Polaris third lane* it is precisely the trade-off we want: Polaris's BLS lane sets the latency floor, Pulsar sets the per-block PQ floor, and Magnetar shoulders the diversity insurance — it does not need to be the fastest, only to be *independent* of the other two.

1.3 What is novel

The cryptographic primitives in this paper are not new. FIPS 205 [41] is the NIST-standardised SLH-DSA. A per-validator standalone signature is just a single-party FIPS 205 signature, and N-of-N collection at a consensus layer is the canonical pattern for ML-DSA (already in production via the `QuasarCert.MLDSAProof` field) and BLS aggregate.

What `luxfi/magnetar` contributes is the *composition*:

1. An explicit-semantic public-BFT API surface at `ref/go/pkg/magnetar/standalone.go` — `PerValidatorKeypair`, `ValidatorSign`, `ValidatorBatchVerify`, `ValidatorAggregateCert`, `BuildAggregateCert`, `VerifyAggregateCert` — that decomplects the public-BFT signing primitive from the TEE-only threshold forms below. One signing primitive → one keypair → no shared state → no aggregator-in-TCB. The quorum decision lives at the consumer.
2. A canonical wire codec (MAGS for signatures, MAGG for group keys; 11-byte fixed header carrying magic, version, mode, and big-endian payload length) at `ref/go/pkg/magnetar/wire.go`, with a stateless dispatcher `VerifyBytes(groupKey, msg, sig)` that accepts wire bytes from the threshold daemon and verifies under FIPS 205 with no out-of-band state. The codec is siblinged with Pulsar's PULS/PULG and Corona's CORS/CORG canonical wire framings, so the consensus-layer dispatcher is uniform across the three Polaris lanes.
3. A byte-identity gate TestMagnetar_Wire_FIPS205Verifiable that pins per-validator σ_i to single-party `cloudflare/circl/sign/slhdsa.Verify` across all three FIPS 205 SHAKE parameter sets. The verifier-side code path is the unmodified circl reference; no Magnetar code runs there. This is the empirical Class-N1-analog property translated to the standalone construction.
4. Hash-function domain separation across every Magnetar cSHAKE256 invocation — cSHAKE256 with function name "Magnetar" and per-purpose customisation strings — so cross-context replay attacks reduce to cSHAKE256 collision resistance.
5. A Polaris cert-profile composition: Magnetar joins Pulsar and Corona as the third lane of the maximum-assurance profile, sharing the LSS lifecycle adapter contract [47] and the Prism transcript-binding discipline [25]. The `ValidatorAggregateCert` parallel-slices shape is designed to feed the Z-Chain Groth16 rollup that compresses $N \times |\sigma|$ to a constant-size certificate.

6. THBS-SE (Threshold Hash-Based Signatures with Selected-Element Reconstruction), the permissionless threshold companion to the per-validator standalone path, specified in §5. THBS-SE is a t-of-n construction in which the public combiner role is open to any peer (validator, block proposer, RPC node, passive watcher); no host is in the TCB at sign time. The construction enforces the hard invariant "a revealed value is allowed only if it is also present in the final SLH-DSA signature" — party-local share material never includes the SLH-DSA seed, only Shamir leaves over GF(257) — and produces FIPS 205 wire bytes byte-identical to single-party SignDeterministic on the reconstructed seed.

1.4 Trust model summary

The per-validator standalone primitive has a clean trust model: it is the trust model of single-party FIPS 205 SLH-DSA, replicated independently N times. We state it explicitly.

In the TCB at sign time: validator i's own host. Each validator holds its own sk_i locally; losing sk_i to an attacker allows that attacker to sign as validator i until pk_i is rotated out of the validator-set registry. This is the same exposure as a classical secp256k1 validator key, and the same operational mitigations apply (HSM-backed key storage, key rotation, slashing evidence for double-signing).

NOT in the TCB: the consensus layer, the `ValidatorAggregateCert` aggregator path, the `VerifyAggregateCert` verifier, the wire codec, or any other validator. The aggregation and verification primitives are pure public-input functions over $(\mathit{PubKeys}, \mathit{Sigs}, \mathit{Signers}, \mu, \mathit{registry})$. There is no shared secret to compromise.

Cross-validator independence. Because each validator holds an independent keypair, compromising sk_i for validator i gives the attacker exactly one quorum vote. Forging a quorum requires compromising $\geq \mathit{threshold}$ distinct validator hosts, which is the same Byzantine-fault threshold the rest of the consensus stack is sized to. No threshold-secret reconstruction event exists to be raced against; an attacker who steals $\mathit{threshold} - 1$ keypairs cannot recover or forge with the remaining honest keypairs.

Quantum threat. The per-validator standalone primitive inherits FIPS 205's post-quantum security budget: $\sim 2^{192}$ classical / $\sim 2^{128}$ quantum at SHAKE-192s, and $\sim 2^{256}$ classical / $\sim 2^{128}$ quantum at SHAKE-256s. A quantum adversary that can mount a 2^{128}-Grover attack against a single validator's SHA-3 evaluations would need $\sim 2^{128}$ quantum operations per validator to forge that validator's signatures, and would still control only one quorum vote per forgery target.

THBS-SE trust model. The permissionless threshold companion (§5) carries a distinct trust model: a public combiner reconstructs the SLH-DSA seed transiently in its own memory for the duration of one SignDeterministic call (sub-second wall-clock at SHAKE-192s) and zeroizes. The combiner role is open to any peer; no long-lived secret material exists outside party-local Shamir leaves; no host is in the TCB by policy. This is materially stronger than a TEE-attested privileged-aggregator model and materially weaker than a strict "no transient seed at any moment" invariant. The strict-atom-assembly path (the v1.1 lift tracked in `BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11`) closes the gap by reconstructing only the message-selected FORS leaves and WOTS+ chain bases from per-atom shares, bypassing `slh_sign_internal` entirely; that work requires a Magnetar-internal reimplementation of FIPS 205 §§5–8 and is out of scope for v1.0.

1.5 What this paper specifies

Section 2 reviews FIPS 205 SLH-DSA, the threshold-signature syntax used by THBS-SE, and the security games. Section 3 gives the per-validator standalone construction byte-exact in pseudocode, keyed to the Go canonical at `ref/go/pkg/magnetar/standalone.go` and `wire.go` byte-for-byte. Section 4 states the security argument: byte-identity to single-party FIPS 205, the N-of-N verification semantics, the existential unforgeability inheritance from FIPS 205 UF-CMA per validator, and the named THBS-SE selected-element-reveal invariant theorem. Section 5 specifies THBS-SE: the slot binding, the commit-and-reveal rounds, the public combiner, the slashable equivocation and malformed-share evidence shapes, and the byte-identity theorem that ties the public-combiner output to single-party FIPS 205 SignDeterministic. Section 6 surveys the reference implementation. Section 7 reports parameter sets and per-signature latency from the production test suite. Section 8 locates Magnetar alongside Pulsar and Corona in the Polaris cert profile. Section 9 surveys the threshold-SLH-DSA literature and the broader threshold-hash-based-signature work. Section 10 closes.

Artefact status. At the time of writing, the artefact is `luxfi/magnetar` v1.0.0. The per-validator standalone primary primitive shipped at v0.5.0 and is unchanged at v1.0; THBS-SE is the v1.0 closer of the permissionless-threshold story and replaces the v0.x reveal-and-aggregate path. The byte-identity regression tests TestMagnetar_Wire_FIPS205Verifiable and TestThbsSE_Wire_FIPS205Verifiable pin both primitives to unmodified `cloudflare/circl/sign/slhd` across all three SHAKE parameter sets under `-count=1` and `-race`. The Tier-A documentation shape (10-document package mirroring Pulsar's structure, with the v0.x submission scaffolds ported forward and the v1.0 normative content in `SPEC.md` + `THBS-SPEC.md`) is in-tree. Mechanised proof artefacts for THBS-SE, the strict-atom-assembly v1.1 lift, and external cryptographer audit of the v1.0 surface are the remaining gates to full Tier A.

2 Preliminaries

2.1 Notation

- λ — security parameter (set to 192 for SHAKE-192s, 256 for SHAKE-256s).
- N — validator-set size at the consensus layer (the primary primitive of this paper is N-of-N collection; a quorum policy of size $K \leq N$ filters the count at the consumer).
- P_i — validator i, $1 \leq i \leq N$. Each validator is identified by a 32-byte NodeID.
- (sk_i, pk_i) — validator i's FIPS 205 SLH-DSA keypair. Each validator's keypair is independent of every other validator's; there is no shared seed, no DKG, and no dealer.
- σ_i — validator i's FIPS 205 SLH-DSA signature (the wire bytes) on the consensus message.
- μ — the message being signed; ctx — the FIPS 205 context string (0–255 bytes).
- $registry : \mathsf{NodeID} \to \{0,1\}^{|pk|}$ — the consensus layer's authoritative validator-set registry, binding NodeID_i to pk_i.
- cSHAKE256 — the cSHAKE256 customisable hash from NIST SP 800-185 [28], with function-name string "Magnetar" and per-context customisation tags.
- $\|$ — byte concatenation; $\oplus$ — bitwise XOR.

Appendix notation. The appendices on threshold forms introduce additional symbols: n for committee size, t for reconstruction threshold, GF(p) for the Shamir field with $p = 257$, S for the shared SLH-DSA scheme seed, and $share_i$ for party i's share. Those symbols are local to the appendices and do not appear in the main per-validator standalone construction.

2.2 FIPS 205 SLH-DSA review

FIPS 205 [41] (formerly SPHINCS+ [6]) is the NIST stateless hash-based signature standard. The Magnetar reference implementation targets the FIPS 205 SHAKE parameter sets:

Table 1: FIPS 205 SHAKE parameter sets exposed by Magnetar v0.5. Values from FIPS 205 [41] §10.1 Table 2.

Parameter set	NIST PQ Cat.	n (WOTS+)	h	d	$\lvert pk\rvert$	$\lvert\sigma\rvert$
SLH-DSA-SHAKE-192s	3	24 B	63	7	48 B	16 224 B
SLH-DSA-SHAKE-192f	3	24 B	66	22	48 B	35 664 B
SLH-DSA-SHAKE-256s	5	32 B	64	8	64 B	29 792 B

The FIPS 205 construction is built from the following layered components, top to bottom:

- **XMSS-HT (hypertree)**: a tree of d XMSS layers. The leaves of the root layer are XMSS public keys; the leaves of the bottom layer are FORS public keys. Every layer has height h/d.
- **XMSS (eXtended Merkle Signature Scheme)**: a binary Merkle tree where each leaf is a WOTS+ public key.
- **WOTS+ (Winternitz One-Time Signature)**: a one-time signature whose security reduces to hash function preimage and second-preimage resistance.
- **FORS (Forest of Random Subsets)**: the few-time signature scheme used at the bottom of the hypertree.
- **Tweakable hash $\mathbf{F}, \mathbf{H}, \mathbf{T}_\ell$**: the cryptographic primitives, instantiated as SHAKE256 for the SHAKE-family Magnetar targets.

Key generation. KeyGen [41] §9.1 samples a $3n$-byte secret triple $(SK.seed, SK.prf, PK.seed)$ and computes $PK.root = \mathbf{T}_h(SK.seed, PK.seed, \text{ADRS})$ as the XMSS-HT root. The public key is $pk = (PK.seed, PK.root)$; the secret key is $sk = (SK.seed, SK.prf, PK.seed, PK.root)$. The total scheme-seed size (the input to DeriveKey) is $4n$ bytes: 96 for $n = 24$ (192s/192f) and 128 for $n = 32$ (256s).

Deterministic signing. $\text{Sign}_{\text{det}}(sk, \mu, ctx)$ [41] §9.2 computes a randomiser $R \leftarrow \text{PRF}_{SK.prf}(ctx \| \mu)$ and proceeds deterministically. The output is byte-identical across calls with the same (sk, μ, ctx). The Magnetar per-validator standalone primitive uses this variant; the byte-identity claim against `cloudflare/circl/sign/slhdsa.Verify` pins to this determinism.

Randomised signing. $\text{Sign}_{\text{rand}}(sk, rng, \mu, ctx)$ [41] §9.2 samples a fresh n-byte randomiser from rng. The Magnetar `ValidatorSign` entry point routes to the deterministic variant by default (`rng=nil`); passing a non-nil reader selects the randomised variant for callers who specifically want hedging against fault injection on the PRF derivation.

Verification. Verify(pk, μ, σ, ctx) [41] §9.3 walks the XMSS-HT root, recomputes $PK.root$ from the FORS and XMSS authentication paths in σ, and accepts iff it matches the $PK.root$ embedded in pk. The verifier consumes no secret material; *any unmodified FIPS 205 verifier accepts a Magnetar signature with no code change.*

2.3 Per-validator standalone signature syntax

The primary primitive of this paper is a per-validator standalone SLH-DSA construction. The syntax is the single-party FIPS 205 syntax, replicated independently N times, plus an aggregate-certificate shape at the consensus layer.

Definition 1 (Per-validator standalone signature scheme)**.** $\Pi^{\text{std}}_{\text{Magnetar}}$ *is a tuple of probabilistic polynomial-time algorithms* (KeyGen, Sign, BatchVerify, BuildCert, VerifyCert)*:*

- $\textsc{KeyGen}(1^\lambda)$*: each validator* i *runs this independently on its own host. Outputs an SLH-DSA keypair* (sk_i, pk_i)*. No interaction.*
- $\textsc{Sign}(sk_i, \mu, ctx)$*: validator* i*'s signing primitive. Returns* $\sigma_i = \textsc{slhdsa.SignDeterministic}(sk_i, \mu, ctx)$*. Deterministic.*
- $\textsc{BatchVerify}(\{pk_i\}, \mu, \{\sigma_i\}) \to$ *bitmap: returns a Boolean bitmap indicating per-signer FIPS 205 verifier acceptance. Pure function of public inputs.*
- $\textsc{BuildCert}(Mode, \{\mathsf{NodeID}_i, pk_i, \sigma_i\}_{i \in S})$*: assembles the on-wire ValidatorAggregateCert carrying the parallel-slices triple* $(Signers, PubKeys, Sigs)$ *for the contributing set* S*.*
- $\textsc{VerifyCert}(cert, \mu, registry) \to$ *count: enforces registry binding (drops unknown signers and pubkey mismatches), dedupes repeated signers, calls* BatchVerify *on the verifiable subset, and returns the count of valid signers. The quorum decision "count* $\geq K$*?" lives at the consumer.*

The deliberate choice that Sign is the unmodified FIPS 205 SignDeterministic — not a Magnetar-specific envelope — is the load-bearing operational property: any deployment shipping a Magnetar signer needs to add no new verification code on the consuming side. Every SLH-DSA verifier in the wild (`cloudflare/circl/sign/slhdsa` [17], the FIPS 205 reference, openssl-pq, libsignal-pq) accepts Magnetar signatures unmodified after the wire envelope is stripped.

2.4 Threshold signature syntax (appendix forms)

The threshold forms documented in the appendices use the standard threshold-signature syntax of Canetti [14] and Boneh–Shoup [10] as specialised to SLH-DSA.

Definition 2 (Threshold SLH-DSA scheme (THBS-SE))**.** $\Pi^{\text{thr}}_{\text{Magnetar}}$ *is a tuple* (Setup, SignShare, Combine, V

- Setup*: produces the joint group public key* pk *and per-party secret shares. For THBS-SE (§5.4) this is* `NewThbsSeKey` *(deterministic dealer reference; production routes through the sibling leaderless PVSS-DKG in* `luxfi/threshold`*).*
- SignShare*: per-party contribution. For THBS-SE (§5.5–5.6) this is the two-round commit-and-reveal of the per-party mask and masked Shamir share.*
- Combine*: PUBLIC, deterministic combiner function (§5.7). For THBS-SE this Lagrange-reconstructs the scheme seed over* GF(257) *in the public combiner's transient memory, dispatches to FIPS 205* SignDeterministic*, and zeroizes.*
- Verify*:* the unmodified FIPS 205 verifier *of [41] §9.3.*

THBS-SE preserves the FIPS 205 output shape: Combine output is byte-identical to single-party SignDeterministic on the implicit $(seed, \mu, ctx)$ tuple (Theorem 5). A relying party sees no difference between a standalone σ_i and a THBS-SE σ once both are unwrapped from the canonical MAGS envelope.

2.5 Shamir secret sharing over GF(257) (THBS-SE)

We instantiate Shamir secret sharing [48] byte-wise over GF(p) with $p = 257$ for THBS-SE (§5.4). The choice of $p = 257$ is dictated by two constraints:

1. $p > 255$, so every byte value $0, \ldots, 255$ is a distinct field element.
2. $p < 2^9$, so each share value fits in a 9-bit lane, packed as one `uint16` per byte position on the wire.

The trade-off: $p = 257$ admits at most $p - 1 = 256$ distinct non-zero evaluation points, capping the committee size at 256 (constant `MaxCommittee257` in `params.go`). For the Lux validator committees this is comfortably above the production thresholds (Quasar operates at $K \leq 21$ [32]).

Sharing. To share a SEED_SIZE-byte secret $\mathbf{s} \in [0,256)^{\text{SEED\_SIZE}}$ across n parties at threshold t:

1. For each byte position $b \in [0, \text{SEED\_SIZE})$, sample $t-1$ random coefficients $a_{b,1}, \ldots, a_{b,t-1} \in \mathrm{GF}(257)$ and define $f_b(x) = \mathbf{s}[b] + \sum_{j=1}^{t-1} a_{b,j} x^j \pmod{257}$.
2. For each recipient P_i at evaluation point $x_i = i$ (one- indexed), compute $\mathit{share}_i[b] = f_b(x_i) \bmod 257$ for all b.
3. Wire-form: pack each share_i as a sequence of SEED_SIZE big-endian `uint16` values ($2 \cdot$ SEED_SIZE bytes total).

Reconstruction. Given any t shares $\{(x_i, \mathit{share}_i)\}_{i \in Q}$ with $|Q| \geq t$, the constant term of each f_b is recovered by Lagrange interpolation at $x = 0$:

$$\mathbf{s}[b] \;=\; \sum_{i \in Q} \lambda_i^Q(0) \cdot \mathit{share}_i[b] \pmod{257}, \quad \lambda_i^Q(0) = \prod_{j \in Q, j \neq i} \frac{-x_j}{x_i - x_j} \bmod 257. \tag{1}$$

The Lagrange coefficients $\lambda_i^Q(0)$ are computed once per quorum using Fermat's little theorem for modular inversion ($a^{-1} \equiv a^{p-2} \pmod{p}$); see `shamir.go` lines 148–164.

Information-theoretic privacy. For any $|Q| < t$, the joint distribution of $\{\mathit{share}_i\}_{i \in Q}$ is uniformly distributed over $[0,257)^{|Q| \cdot \text{SEED\_SIZE}}$ *independent* of $\mathbf{s}$. This is the standard Shamir privacy property [48, 24]: every $t-1$-subset of shares reveals zero information about the secret, as a matter of information theory (not computational complexity). This is the load-bearing secrecy guarantee of the appendix threshold forms against a coalition of $t-1$ or fewer committee members.

The GF(257)/byte-domain mismatch. The Lagrange reconstruction (1) produces a value in $[0, 257)$; the SLH-DSA scheme seed is byte-valued in $[0, 256)$. The single value 256 has no byte representation. The appendix forms handle this *constructively*: the Shamir reconstruction is fed into a cSHAKE256 mix

$$S = \text{cSHAKE256}\big(\text{byteSum}_{BE} \,\|\, \mathit{committee\_root}, \text{N} = \text{“Magnetar”}, \text{S} = \text{“MAGNETAR-SEED-SHARE-V1”}, 8{\cdot}\text{SI} \tag{2}$$

which absorbs the GF(257)-valued byteSum as a $2 \cdot$ SEED_SIZE-byte big-endian buffer and emits a SEED_SIZE-byte S. The cSHAKE256 mix domain-separates the seed from raw Shamir output and gives a constant-length seed regardless of the value distribution of byteSum.

2.6 Security games

The per-validator standalone primitive's security game is the single-party SLH-DSA UF-CMA game replicated N times in parallel. We state the formal definition.

Definition 3 (Multi-validator EUF-CMA). $\mathsf{Game}^{\text{multi-EUF-CMA}}_{\Pi^{\text{std}}_{\text{Magnetar}}}(\mathcal{A}, \lambda, N)$*:*

1. ***Setup.*** *The challenger runs* KeyGen *independently N times to obtain (sk_i, pk_i) for $i = 1, \ldots, N$. The adversary $\mathcal{A}$ commits to a corrupt set $F \subset [N]$ with $|F| < K$ at the outset, where K is the quorum threshold; the challenger gives $\mathcal{A}$ $\{(pk_i)\}_{i=1}^N \cup \{sk_i : i \in F\}$.*
2. ***Queries.*** *$\mathcal{A}$ adaptively requests signatures from honest validators on messages $\mu_1, \ldots, \mu_{Q_S}$. For each query (i, μ) with $i \notin F$, the challenger returns $\sigma_i = \text{Sign}(sk_i, \mu, ctx)$. $\mathcal{A}$ may invoke the random oracle $\mathcal{O}_H$ up to Q_H times.*
3. ***Forgery.*** *$\mathcal{A}$ outputs $(\mu^\star, cert^\star)$ and* wins *iff* VerifyCert$(cert^\star, \mu^\star, registry) \geq K$ *and there exist $K - |F|$ signers $i \notin F$ whose signatures in $cert^\star$ are valid forgeries on $\mu^\star \notin \{\mu_j : (i, \mu_j) \in queries\}$.*

The advantage is $\mathsf{Adv}^{\text{multi-EUF-CMA}}_{\Pi^{\text{std}}_{\text{Magnetar}}, \mathcal{A}}(\lambda, N) = \Pr[\mathcal{A} \text{ wins}]$.

The multi-validator forgery condition reduces to single-party SLH-DSA UF-CMA via a standard hybrid argument: each honest validator's keypair is independent, so any forgery on validator i's signature yields a single-party SLH-DSA UF-CMA forgery on (pk_i, sk_i). The reduction has a factor N loss for the hybrid (the standard hybrid argument over independent keypairs); the SLH-DSA UF-CMA bound at the SHAKE-192s parameter set ($\leq 2^{-192}$ classical, $\leq 2^{-128}$ quantum) absorbs this factor comfortably for any $N \leq 2^{60}$.

The THBS-SE security game is the slot-bound permissionless threshold EUF-CMA game: the adversary corrupts up to $t-1$ parties, observes Round-1/Round-2 broadcasts of the honest parties at chosen slot bindings $bind^*$ and messages μ^*, may invoke the public combiner on any subset of those broadcasts, and wins by producing a signature on a fresh $(bind^\star, \mu^\star)$ that verifies under the committee group public key. The named selected-element-reveal invariant (Theorem 4) bounds the adversary's per-round leak to the Shamir information-theoretic share class plus public-transcript bytes; we state the formal security argument in §4.7.

3 Construction

This section specifies the per-validator standalone construction byte-for-byte. The pseudocode is keyed to the Go canonical at `ref/go/pkg/magnetar/standalone.go` and `ref/go/pkg/magnetar/wire.go` at tag `v1.0.0`. The THBS-SE permissionless threshold companion is specified in §5.

3.1 Hash-domain separation

Every cryptographic hash invocation in Magnetar is cSHAKE256 [28] with function-name string `"Magnetar"` and a per-purpose customisation string. The per-validator standalone path uses only the wire-codec canonical magic constants (MAGS / MAGG) plus the standard FIPS 205 dispatch; THBS-SE adds four per-purpose tags listed in §5.3.

Table 2: Magnetar v0.5 cSHAKE256 domain separation. Every secret-bearing hash invocation is bound to exactly one tag; cross-tag replay reduces to cSHAKE256 collision resistance. Tags marked *(thresh)* appear only in the threshold-form appendices.

Tag	Purpose
`MAGS` (magic 0x4D414753)	MAGS wire envelope for FIPS 205 signature payload (standalone primary)
`MAGG` (magic 0x4D414747)	MAGG wire envelope for FIPS 205 public-key payload (standalone primary)
`MAGNETAR-DKG-COMMIT-V1`	committee-root digest (sorted committee binding) *(thresh)*
`MAGNETAR-DKG-TRANSCRIPT-V1`	DKG transcript digest (Round-2 equivocation check) *(thresh)*
`MAGNETAR-SIGN-R1-V1`	threshold-sign Round-1 commit D_i *(thresh)*
`MAGNETAR-SIGN-MASK-V1`	per-attempt mask r_i derivation *(thresh)*
`MAGNETAR-SEED-SHARE-V1`	Shamir coefficient stream + final seed mix *(thresh)*

The protocol-wide domain separation context is the byte string `"lux-magnetar-v0.5"` (`tagDomainSep`). It is bound into the wire-codec magic constants and into every threshold-form transcript binding so that a separate Magnetar version (or a different Lux PQ primitive that happened to reuse the same hash function) cannot collide.

3.2 Per-validator keypair generation

Each validator i runs `PerValidatorKeypair` once at validator-set onboarding, persists (sk_i, pk_i) to local HSM-backed storage, and registers pk_i in the consensus layer's validator-set commitment.

Algorithm (`standalone.go:197-199`, `keygen.go`).

1. Pull `params` for the target `Mode` (SHAKE-192s, SHAKE-192f, or SHAKE-256s).
2. Source $4n$ bytes of entropy from `rng` (where n is the FIPS 205 "n" parameter: 24 for 192, 32 for 256). If `rng=nil`, source from `crypto/rand`.
3. Pass the entropy buffer to `circl.slhdsa.Scheme(params.ID).DeriveKey`.
4. Output (sk_i, pk_i) in the FIPS 205 wire form: $sk_i = (SK.seed, SK.prf, PK.seed, PK.root)$ of length $4n$; $pk_i = (PK.seed, PK.root)$ of length $2n$.

Determinism. If `rng` is a deterministic reader (e.g., a `detReader` seeded by the canonical test fixture), the output keypair is byte-identical across calls. This is the property KAT regeneration relies on.

Independence. No shared state. No correlation between (sk_i, pk_i) and (sk_j, pk_j) for $i \neq j$. Each keypair is generated on its own host with its own RNG.

3.3 Per-validator signing primitive

`ValidatorSign` is the canonical signing primitive for the public-BFT path. It is a thin wrapper over the FIPS 205 reference, exposed with a public-BFT-explicit signature.

Algorithm (`standalone.go:231-244`, `sign.go`).

1. Verify sk_i is non-nil and the embedded mode matches an installed `Params`.
2. Pull the `slhdsa.Scheme` for the embedded mode.
3. If `rng=nil`, invoke `Scheme.SignDeterministic(sk_i, msg, ctx=nil)`. Otherwise invoke `Scheme.SignRandomized(sk_i, rng, msg, ctx=nil)`.
4. Return the raw FIPS 205 signature bytes (`params.SignatureSize`: 16,224, 35,664, or 29,792 depending on mode).

Context binding. The `ctx` parameter is intentionally omitted at the `ValidatorSign` surface: the consensus layer binds chain-id / block-height / epoch into the message μ upstream (the QuasarCert transcript-hash pattern). The `Sign` entry point in `sign.go` retains the `ctx` parameter for callers that need explicit context binding; the `ValidatorSign` wrapper passes `ctx=nil`.

Determinism. With `rng=nil`, σ_i is byte- identical to what a single-party FIPS 205 signer would emit on $(sk_i, \mu, ctx = \varepsilon)$. This is the load-bearing property pinned by TestMagnetar_Wire_FIPS205 at `wire_test.go:166-220` (cf. §4.1).

3.4 Aggregate certificate construction

The consensus layer collects validator signatures into a `ValidatorAggregateCert` for transport over the wire. The cert is intentionally a parallel-slices shape (not a nested per- signer record list) because it maps directly to the Z-Chain Groth16 rollup witness layout.

Wire layout (`standalone.go:415-432`, `wire.go`).

$$\begin{aligned} \mathit{ValidatorAggregateCert} \;=\; & \mathit{Mode}_1 \,\|\, N_{BE,4} \,\|\, \mathit{Signers}[0]_{32} \,\|\, \cdots \,\|\, \mathit{Signers}[N-1]_{32} \\ & \|\, \mathit{PubKeys}[0]_{|pk|} \,\|\, \cdots \,\|\, \mathit{PubKeys}[N-1]_{|pk|} \\ & \|\, \mathit{Sigs}[0]_{|\sigma|} \,\|\, \cdots \,\|\, \mathit{Sigs}[N-1]_{|\sigma|}. \end{aligned}$$

Algorithm `BuildAggregateCert` (`standalone.go:443-478`).

1. Validate `params` matches one of the canonical parameter tables.
2. Validate the three parallel slices ($signers, pubKeys, sigs$) have the same length $N \geq 1$.
3. For each $i \in [0, N)$, validate $|pubKeys[i]| = params.PublicKeySize$ and $|sigs[i]| = params.SignatureSize$
4. Defensively copy each $pubKeys[i]$ and $sigs[i]$ (the caller's slices may be aliased; the cert is wire-stable and must not move under the caller).
5. Return the `ValidatorAggregateCert` record.

Independent verifiability. Each σ_i in the cert verifies independently against pk_i; there is no cross-signer algebraic coupling. A relying party can verify any subset of the cert (e.g., for incremental quorum-detection) without sharing any secret state with other relying parties.

3.5 Aggregate certificate verification

`VerifyAggregateCert` is the consumer-facing verification primitive. It enforces registry binding (dropping unknown signers and pubkey mismatches), dedupes repeated signers, runs the parallel batch verify, and returns the count of valid signers. The quorum decision lives at the consumer.

Algorithm (`standalone.go:519-613`).

1. Validate `cert` is non-nil and the embedded `Mode` matches a canonical parameter set.
2. Validate $N \geq 1$ and the parallel slices have matching lengths.
3. For each $i \in [0, N)$, validate $|cert.PubKeys[i]| = params.PublicKeySize$ and $|cert.Sigs[i]| = params.SignatureSize$.
4. *Phase 1 (entry classification).* For each i, dedupe against `seen`, look up $registered = knownValidators[cert.Signers[i]]$; mark the entry verifiable only if (a) the lookup succeeds, (b) $|registered| = params.PublicKeySize$, and (c) $\textsc{ctEqualBytes}(cert.PubKeys[i], registered)$. Both cases (a)/(b)/(c) failing are counted as INVALID, not fatal.
5. *Phase 2 (parallel batch verify).* Collect the verifiable entries into flat slices and invoke `ValidatorBatchVerify`, which dispatches serial or goroutine-parallel based on the `verifyAggregatedConcu` constant (currently ~ 16).
6. *Phase 3 (tally).* Count verifiable + valid entries. Return the count.

Defense-in-depth choices.

1. **Unknown signers counted as INVALID, not fatal.** A cert carrying a signature from a stale or unknown validator does not crash the verifier; the verifier reports the count, and the quorum policy at the consumer decides whether to accept. This is the right discipline when certs may be Z-Chain Groth16-rolled-up over a stale validator registry: the verifier must not abort the entire batch on one stale entry.
2. **Pubkey mismatches counted as INVALID, not fatal.** Same rationale. An attacker who slips a SignedBundle from an unknown validator into the cert does not gain a count but does not crash the verifier.
3. **Duplicate signers deduped first-wins.** The first occurrence wins; subsequent occurrences are silently dropped (not counted twice, not counted as invalid). The consensus layer can collect proof of double-signing separately via per-signer `Sigs` comparison.

4. **Constant-time pubkey comparison.** `ctEqualBytes` short-circuits at the wire-length validation and otherwise runs a constant-time byte-by-byte XOR-then-OR reduction. No secret-dependent timing.

3.6 Canonical wire codec (MAGS / MAGG)

The wire codec at `wire.go` provides byte-stable framing for individual signatures (MAGS) and group keys (MAGG), with a stateless verifier dispatcher (`VerifyBytes`) for consensus-layer consumption.

MAGS signature envelope.

$$\begin{aligned} MAGS\_frame \; &= \texttt{0x4D4147} \,\|\, \texttt{0x53} \,\|\, version_{BE,2} \,\|\, mode_1 \,\|\, payloadLen_{BE,4} \\ &\quad \|\, sigPayload_{params.SignatureSize}. \end{aligned}$$

Total length: $11 + params.SignatureSize$ bytes. The 11-byte fixed header carries the magic ASCII string "MAGS", the version (currently 1), the FIPS 205 mode byte, and the big-endian payload length. The payload is the raw FIPS 205 signature bytes verbatim.

MAGG group-key envelope.

$$\begin{aligned} MAGG\_frame \; &= \texttt{0x4D4147} \,\|\, \texttt{0x47} \,\|\, version_{BE,2} \,\|\, mode_1 \,\|\, payloadLen_{BE,4} \\ &\quad \|\, pkPayload_{params.PublicKeySize}. \end{aligned}$$

Total length: $11 + params.PublicKeySize$ bytes. Same structure as MAGS with a distinct magic suffix byte ("G" vs "S").

Stateless `VerifyBytes` dispatcher.

1. Validate $len(groupKeyWire) \geq 11$ and the magic matches `MAGG`.
2. Validate $len(sigWire) \geq 11$ and the magic matches `MAGS`.
3. Decode the version, mode, and declared payload length from each frame. Validate $len(groupKeyWire) = 11 + declared_{gk}$ and $len(sigWire) = 11 + declared_{sig}$.
4. Validate both frames target the same mode.
5. Strip the 11-byte header from each, recovering the raw FIPS 205 pk_{raw} and σ_{raw} payloads.
6. Dispatch to `circl.slhdsa.Scheme(mode).PublicKey.UnmarshalBinary(` pk_{raw}`)` and then `Scheme.Verify(`pk`, msg, `σ_{raw}`, ctx=nil)`.
7. Return the Boolean from the FIPS 205 verifier.

Byte-identity to FIPS 205. The wire codec is purely additive (an envelope prefix); the payload bytes are the raw FIPS 205 signature or public-key bytes verbatim. Stripping the 11-byte header recovers exactly what `cloudflare/circl/sign/slhdsa` emits on the same (sk, μ, ctx) tuple. The TestMagnetar_Wire_FIPS205Verifiable regression pins this property across all three SHAKE parameter sets.

3.7 Verify (FIPS 205 passthrough)

The per-signature `Verify` entry point at `verify.go` is a thin dispatch to `circl.slhdsa.PublicKey.UnmarshalBi` followed by `slhdsa.Verify(`pk`, NewMessage(`μ`), `σ`, ctx)`. No Magnetar-specific envelope or wrapper. This is the FIPS 205-byte- identity property: the produced signatures live in the exact FIPS 205 signature space, and any FIPS 205 verifier accepts them unmodified after the wire-codec envelope (if present) is stripped.

3.8 Constant-time discipline

The per-validator standalone primitive's hot paths are:

- `ValidatorSign` → `Sign` → `circl.slhdsa.SignDeterministic` (constant-time per circl design).
- `ValidatorBatchVerify` → `circl.slhdsa.Verify` (constant-time per circl design; verifier accepts only public inputs).
- `VerifyAggregateCert` pubkey-equality comparison via `ctEqualBytes` (constant-time XOR-then-OR reduction).
- Wire-codec header validation via fixed-length big-endian decode and constant-time magic comparison.

None of the standalone-path operations branch on secret material: sk_i is consumed only by `circl.slhdsa.SignDeterministic` which is constant-time by construction. There are no Shamir operations, no Lagrange interpolation, and no aggregator-held secrets in the standalone path. The THBS-SE construction (§5) carries additional constant-time obligations on the byte-wise Lagrange reconstruct and the commit re-derivation documented in §4.9.

3.9 Identifiable abort (consensus layer)

The per-validator standalone primitive has a clean abort posture: each σ_i is a single-party FIPS 205 signature, and `VerifyAggregateCert` reports the per-signer outcome via the bitmap from `ValidatorBatchVerify`. The consensus layer collects evidence of misbehaviour:

- **Invalid signature.** The verifier bitmap reports *false* for the offending signer. The $(\mathsf{NodeID}_i, pk_i, \sigma_i, \mu)$ quad is sufficient on-chain evidence for slashing.
- **Double-signing.** Two valid signatures from the same NodeID_i on distinct messages on the same slot constitute evidence; the consensus layer collects $(\mathsf{NodeID}_i, pk_i, \sigma_{i,A}, \sigma_{i,B}, \mu_A, \mu_B)$ as the slashing record.
- **Unregistered signer.** An entry whose NodeID_i does not appear in `knownValidators` is counted as INVALID; the consensus layer may treat repeated unregistered-signer entries as a peer misbehaviour signal at the gossip layer.
- **Pubkey-registry mismatch.** An entry whose embedded pk_i does not byte-equal the registered pubkey for that NodeID_i is counted as INVALID; same handling as unregistered signer.

The identifiable-abort taxonomy of THBS-SE (slot-guard equivocation, Round-2 commit-bind failure, wire-size violation, slot-mismatch reveal) is documented in §5.9; both equivocation and malformed-share evidence are pure-function third-party-verifiable.

3.10 Wire sizes

Table 3: Magnetar v1.0 per-validator standalone wire sizes.

Object	Size (bytes)
σ_i FIPS 205 payload (SHAKE-192s)	16 224
σ_i FIPS 205 payload (SHAKE-192f)	35 664
σ_i FIPS 205 payload (SHAKE-256s)	29 792
pk_i FIPS 205 payload (SHAKE-192s / 192f)	48
pk_i FIPS 205 payload (SHAKE-256s)	64
MAGS envelope header (fixed)	11
MAGG envelope header (fixed)	11
NodeID_i in cert	32
`ValidatorAggregateCert` (SHAKE-192s, $K = 21$)	$\sim$ 332 KiB
`ValidatorAggregateCert` (SHAKE-192s, $K = 21$) post Z-Chain Groth16 rollup	$\sim$ 192 B

The signature wire format is *exactly* the FIPS 205 wire format plus the 11-byte MAGS envelope; there is no Magnetar-specific suffix, no per-signer ZK proof, and no out-of-band binding. The Z-Chain Groth16 rollup that compresses $N \times |\sigma|$ to a constant-size certificate is specified separately at the lux-zchain layer and consumes the parallel-slices cert shape verbatim.

4 Security argument

This section states the security claims for the per-validator standalone primitive. The argument is short because the construction is short: each σ_i is a single-party FIPS 205 SignDeterministic output, and FIPS 205's UF-CMA security [41, 6] transfers directly to the per- validator forgery game via a standard hybrid argument. The wire codec is a purely additive envelope; it admits a constant-time byte-identity argument against unmodified `cloudflare/circl/sign/slhdsa`.

4.1 Byte-identity to single-party FIPS 205

Theorem 1 (Standalone byte identity)**.** *Fix a parameter set* $\textsc{Mode} \in \{\text{SHAKE-192s}, \text{SHAKE-192f}, \text{SHAKE-256}$ *and a validator keypair* $(sk_i, pk_i) \leftarrow$ `PerValidatorKeypair`$(\textsc{Mode}, rng)$*. For any message* μ*, the output of* `ValidatorSign`$(sk_i, rng = \textbf{nil}, \mu)$ *is byte-identical to the output of* `circl.slhdsa.Scheme`$(\textsc{Mode})$`.Sign`$(sk_i^{circl}, \mu, \varepsilon)$*, where* sk_i^{circl} *is the* `circl`*-form embedding of* sk_i*.*

Proof. By inspection of the call graph. `PerValidatorKeypair` (`standalone.go:197-199`) delegates to `GenerateValidatorKey`, which delegates to `circl.slhdsa.Scheme(params.ID).DeriveKey`. The output sk_i is the `circl` private-key bytes verbatim. `ValidatorSign` (`standalone.go:231-244`) delegates to `Sign` (`sign.go`), which dispatches to `Scheme.SignDeterministic` when `rng=nil`. The output is the raw FIPS 205 signature bytes returned by `circl`. □

Wire-codec byte identity (corollary).

Corollary 2 (Wire-codec byte identity)**.** *For any* σ_i *from* `ValidatorSign`$(sk_i, rng = \textbf{nil}, \mu)$ *and* pk_i *from* `PerValidatorKeypair`*, the wire frames* `MarshalBinary`$(\textit{Signature}\{\textsc{Mode}, \sigma_i\})$ *and* `MarshalGroupKey`$(\textit{PublicKey}\{\textsc{Mode}, pk_i\})$ *each consist of an 11-byte canonical header followed by the FIPS 205 payload bytes verbatim. Stripping the 11-byte header from each frame recovers exactly the* `circl`*-emitted* $(pk_i^{circl}, \sigma_i^{circl})$*. The unmodified* `circl.slhdsa.Scheme(`Mode`).Verify` *accepts* $(\sigma_i^{circl}, \mu, pk_i^{circl})$*.*

Proof. The MAGS frame layout (§3.6) concatenates a fixed 11-byte header with the FIPS 205 signature payload verbatim; the MAGG frame layout does the same for the public key. `UnmarshalBinary` / `UnmarshalGroupKey` reverse the framing without touching the payload bytes. By Theorem 1, the payload bytes recovered are exactly the `circl`-emitted FIPS 205 bytes. FIPS 205 [41] §9.3 guarantees that $\textsc{Verify}(pk, \mu, \sigma, ctx) = 1$ whenever $\sigma = \textsc{SignDeterministic}(sk, \mu, ctx)$ and pk is the public-key part of the keypair DeriveKey returns for the same seed. □

Empirical pin. The byte-identity property is empirically pinned by TestMagnetar_Wire_FIPS205Veri at `ref/go/pkg/magnetar/wire_test.go:166-220`. The test:

1. Generates a keypair via the v0.5 per-validator standalone primary path (`PerValidatorKeypair` with a deterministic `detReader`).
2. Signs via `ValidatorSign` (`rng=nil`).
3. Frames both via the wire codec (`MarshalGroupKey` and `sig.MarshalBinary`).
4. Extracts the FIPS 205 payload from each frame by skipping the 11-byte header. No Magnetar package code touches the payload at this point.
5. Calls `cloudflare/circl/sign/slhdsa.Scheme(Mode).Verify` directly on the extracted payload. This is the load-bearing assertion of byte-identity; the verifier path is the unmodified `circl` reference with no Magnetar code.
6. Runs the stateless `VerifyBytes` path (used by the threshold daemon at the dispatch layer) on the same wire bytes and asserts it accepts.
7. Tamper checks: flips one byte in the signature payload; asserts both direct-`circl` verify and `VerifyBytes` reject.

The test runs for all three modes (SHAKE-192s, SHAKE-192f, SHAKE-256s) and passes under `-count=1` and `-race`.

4.2 Multi-validator EUF-CMA

Theorem 3 (Magnetar standalone multi-validator EUF-CMA)**.** *Let $\Pi^{\text{std}}_{\text{Magnetar}}$ be the per-validator standalone construction of §3 at parameter set Mode, instantiated with N independent validators of which up to $|F| < K$ are statically corrupted (K being the quorum threshold). For any PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\text{multi-EUF-CMA}}_{\Pi^{\text{std}}_{\text{Magnetar}},\mathcal{A}}(\lambda, N) \;\le\; N \cdot \mathsf{Adv}^{\text{UF-CMA}}_{\text{SLH-DSA}}(\lambda, \mathcal{B}), \tag{3}$$

where $\mathcal{B}$ is a hybrid-argument reduction running $\mathcal{A}$ with at most Q_S signing queries per honest validator and Q_H random-oracle queries, with running time within $\text{poly}(\lambda, N)$ of $\mathcal{A}$.

Proof sketch. Standard hybrid argument over independent keypairs. The challenger samples N independent SLH-DSA keypairs and gives $\mathcal{A}$ the corrupt subset $\{sk_i\}_{i \in F}$. For each $\mathcal{A}$ signing query (i, μ) with $i \notin F$, the challenger forwards (i, μ) to the i-th honest validator's SLH-DSA signing oracle and returns the resulting σ_i. By $\mathcal{A}$'s winning condition, the forgery $cert^\star$ contains at least $K - |F|$ valid signatures from honest validators on $\mu^\star \notin \{\mu_j : (i, \mu_j) \in queries\}$. Pick any one such honest signer $i^\star$; the pair $(\mu^\star, \sigma^\star_{i^\star})$ is a fresh SLH-DSA UF-CMA forgery against $pk_{i^\star}$. The hybrid argument over N honest validators (guessing $i^\star$ uniformly at random) loses a factor N. □

Provable security level. SLH-DSA UF-CMA at SHAKE-192s is upper-bounded by $\sim 2^{-192}$ classical and $\sim 2^{-128}$ quantum (Grover) per validator. At $N = 2^{12}$ (a Lux validator quorum upper bound far above the production $K \sim 21$), the factor-N hybrid loss in (3) absorbs trivially: the classical bound becomes $\sim 2^{-180}$ and the quantum bound becomes $\sim 2^{-116}$. SHAKE-256s absorbs the same factor with $\sim 2^{-244}$ classical and $\sim 2^{-116}$ quantum. Both regimes provide ample headroom relative to the post-quantum security baseline of NIST PQC Category 3 / 5.

No cross-validator coupling. Because each (sk_i, pk_i) is independent, compromising $|F|$ validators yields exactly $|F|$ controllable signatures — no information about the remaining $N - |F|$ keypairs leaks. The cryptographic budget against forging a quorum vote requires compromising $\geq K$ distinct validator hosts (or breaking SLH-DSA on a fresh pk_i which is bounded by the per-validator UF-CMA term).

4.3 Trust posture

The per-validator standalone primitive's TCB is short. We enumerate.

In each validator's TCB at sign time:

- validator i's own sk_i (FIPS 205 wire form, $4n$ bytes).

NOT in any TCB:

- the consensus layer (only handles public inputs);
- the `ValidatorAggregateCert` construction and verification primitives (pure functions of public inputs);
- the wire codec (purely additive envelope; no secret material flows through it);
- any other validator's keypair (independence guarantees);
- any aggregator, dealer, or DKG transcript participant (none exist in this construction).

Threat surface per validator. Compromising validator i's host extracts sk_i and lets the attacker sign as validator i until pk_i rotates. The standard validator-key hardening discipline applies: HSM-backed key storage, key rotation on schedule, slashing evidence for double-signing, attestation-pinned signing services. This is operationally identical to a classical secp256k1 validator key in the pre-quantum era, except the keypair is now FIPS 205 SLH-DSA.

Quantum threat. A quantum adversary mounting a 2^{128} Grover attack against SHA-3 evaluations would need $\sim 2^{128}$ quantum operations to forge a single validator's signature (SHAKE-192s) and would control exactly one quorum vote per successful forgery target. Forging a quorum requires K independent successful attacks, none of which inherit state from any other. The construction therefore inherits FIPS 205's post-quantum security argument verbatim, replicated N times.

Comparison to THBS-SE. The permissionless threshold companion of §5 carries a distinct TCB shape: a public combiner reconstructs the SLH-DSA seed transiently in its own memory for the duration of one SignDeterministic call. The combiner role is open to any peer; no host is in the TCB by policy. See Theorem 4 and §4.9 for the THBS-SE-specific analysis.

4.4 Constant-time posture

The standalone-path constant-time posture is the union of:

- **`circl.slhdsa.SignDeterministic`.** The primary signing path delegates entirely to `cloudflare/circl` v1.6 [17]. The `circl slhdsa` implementation is constant-time per its documented design and is the most widely deployed Go FIPS 205 backend in production. The outstanding side-channel obligation is independent `dudect` [45] validation at $\geq 10^9$ samples (BLOCKERS.md GATE-3).
- **`circl.slhdsa.Verify`.** Verifier delegates entirely to `circl`; constant-time by the same argument.

- `ctEqualBytes` **pubkey comparison.** Constant-time XOR-then-OR reduction in `transcript.go`; no secret-dependent branching. The pubkey comparison is the only operation in `VerifyAggregateCert` that takes a secret-derived input (the registered pubkey side), and even that input is technically public (registered pubkeys are on-chain).
- **Wire codec.** Header decode is fixed-length big-endian arithmetic; magic-constant comparison is constant-time byte-equality. No payload-dependent branching.

The standalone path has no secret-dependent control flow outside the `circl slhdsa` dispatch; the constant-time properties of the wrapped backend dominate the side-channel analysis. The THBS-SE path (§5) carries additional constant-time obligations on the Shamir Lagrange reconstruct, commit-bind comparison, and combiner-side zeroization paths; those obligations are documented in §4.9.

4.5 Composition with Polaris

Polaris is the Lux maximum-assurance cert profile pairing three independent threshold lanes: Pulsar (Module-LWE), Corona (Ring-LWE), and Magnetar (FIPS 205 SLH-DSA). The composed soundness theorem (Theorem 8 of [26], "post-quantum safety") bounds the adversary advantage by the *minimum* of the per-lane advantages. A Polaris adversary must break all three lanes simultaneously: Module-LWE, Ring-LWE, and FIPS 205 SLH-DSA. No known reduction transfers a break of one to a break of another, so the conjunction is strictly harder than breaking any single primitive.

The role-binding invariant is enforced by Prism [25]: every Magnetar σ_i is computed over a consensus message μ that hashes the certificate subject including ($chain\_id$, $epoch$, $round$, $KeyEraID$, $Generatio$ so cross-epoch replay reduces to single-party FIPS 205 forgery on a fresh $\mu^\star$ — which is bounded by the same UF-CMA term as (3).

4.6 Proof roadmap

The standalone primitive's argument is short because the underlying construction is short. The mechanisation roadmap is correspondingly compact.

Lean 4: wire-codec byte identity. Corollary 2's argument is a purely algorithmic statement about envelope framing and is the lowest-hanging mechanisation target. The Lean specification at `lux-formal-verification-lean4` [34] can mechanise this as a structural lemma over the MAGS / MAGG frame layout.

EasyCrypt: hybrid argument for multi-validator EUF-CMA. The hybrid reduction of Theorem 3 maps onto a standard EasyCrypt [4] game-hop sequence: Game_0 is the honest multi-validator EUF-CMA game; Game_i ($i \in [1, N]$) is the hybrid in which validators $1, \ldots, i-1$ are replaced by oracle queries to the SLH-DSA challenger; Game_N is fully oracle-driven. Each hop is a perfect simulation modulo the SLH-DSA UF-CMA hop. This is GATE-1 in `BLOCKERS.md`; the planned artefact lives at `proofs/easycrypt/Magnetar/` once the EasyCrypt theory ships.

Jasmin: constant-time hot paths. The standalone path has no constant-time hot paths outside the `circl`-delegated `slhdsa.SignDeterministic` and `slhdsa.Verify`. The Jasmin [2] obligation applies to the THBS-SE hot paths (§5.7) where Magnetar code holds secret-derived material directly — the byte-wise Lagrange interpolation and the commit re-derivation.

Independent cryptographer review. `BLOCKERS.md::MAGNETAR-EXTERNAL-AUDIT-V11`: an independent cryptographer sign-off is the human-checked Tier A gate. The review scope is the security argument (§4), the THBS-SE construction (§5), the FIPS 205 conformance trace

(FIPS-TRACEABILITY.md), the wire codec, and the v1.0 honest open item on transient seed reconstruction at the public combiner.

4.7 THBS-SE: selected-element-reveal invariant

The THBS-SE construction (§5) enforces a hard invariant on what each party is permitted to reveal. We state this invariant verbatim, as a named theorem, because it is the load-bearing security claim of the threshold companion: every side-channel and protocol-level argument for THBS-SE reduces to this property holding.

Theorem 4 (Selected-element-reveal invariant)**.** *A revealed value is allowed only if it is also present in the final SLH-DSA signature.*

Formally: let $\mathcal{R}_{round}(p_i, slot, \mu)$ denote the set of byte-strings party p_i broadcasts during the two-round THBS-SE signing protocol for slot *slot* and message μ. Let $\sigma = \texttt{Combine}(\dots)$ be the resulting FIPS 205 signature, and let $\mathcal{P}(\sigma) \subseteq \{0,1\}^*$ denote the set of byte-substrings appearing in σ. Then the construction guarantees

$$\bigcup_{p_i \in committee} \mathcal{R}_{round}(p_i, slot, \mu) \;\setminus\; \{0,1\}^*_{transcript} \;\subseteq\; \mathcal{P}(\sigma) \;\cup\; \mathcal{U}, \tag{4}$$

where $\{0,1\}^*_{transcript}$ is the set of public transcript bytes (slot binding, party IDs, commit hashes, masks) and $\mathcal{U}$ is the information-theoretically uniform-random single-party-share class of values that, considered alone, leak no information about $SK.seed$ to anyone holding fewer than t leaves (Shamir's information-theoretic property over GF(257)).

Proof sketch. By inspection of the THBS-SE round-message and Combine code paths (ref/go/pkg/magnetar/th
The Round-2 PartialSig payload is the pair $(r_i, s'_i = share_i \oplus r_i)$. Both r_i and s'_i are uniformly random in the recipient's view (Shamir share property + uniform mask). The Round-1 Commit is a cSHAKE256 output and is by construction public. The Round-2 reveal of a slot-tampered, wire-size-tampered, or commit-mismatched partial-sig is rejected by Combine and emits typed evidence; the malformed reveal does not enter the final signature input set. The honest reveal pair recovers $share_i$ via byte-wise XOR, contributes one GF(257) evaluation to the Lagrange interpolation, and is then zeroized in the combiner's transient memory. The final σ is FIPS 205 SignDeterministic on the reconstructed seed; the seed itself is in $\mathcal{P}(\sigma)$ only in the sense that σ deterministically determines it under the FIPS 205 backward map (which requires breaking the SLH-DSA security to extract). The strict "not even transient seed in combiner memory" refinement requires the v1.1 strict-atom-assembly path tracked at BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11; v1.0 ships the $\mathcal{R}_{round} \subseteq \mathcal{P}(\sigma) \cup \mathcal{U}$ relaxation. □

Forbidden reveals. The invariant's contrapositive is the operational statement: no party may publish $SK.seed$, $SK.prf$, any future-slot share material, WOTS+ chain bases for unused chains, or FORS leaves not selected by the message digest. The slot guard refuses any same-slot re-emission and the share envelope is per-slot, which mechanically enforces the future-slot exclusion. The v1.1 strict-atom-assembly path extends the exclusion to the WOTS+/FORS atom level; v1.0 leans on the FIPS 205 SLH-DSA backward-map opacity for that subset.

4.8 THBS-SE: byte identity to single-party FIPS 205

Theorem 5 (THBS-SE byte identity)**.** *Fix a parameter set* Mode*, a committee $(P_1, \dots, P_n)$ over GF(257) evaluation points $(x_1, \dots, x_n)$, a threshold $t \le n$, a setup transcript setup, and a slot binding bind with $ctx = \texttt{ctxFromSlot}(bind)$. Let (PK, SK) be the master keypair generated at setup with seed S. For any message μ, any $T \subseteq \{1, \dots, n\}$ with $|T| = t$, and any honest Round-1/Round-2 broadcasts $\{(R1_i, R2_i)\}_{i \in T}$, the output $\sigma = \texttt{Combine}(\{(R1_i, R2_i)\}_{i \in T}, bind, \mu, PK)$ is byte-identical to $\texttt{circl.slhdsa.SignDeterministic}(SK, \mu, ctx)$.*

Proof. Combine's Lagrange reconstruction at $x = 0$ over GF(257) is the left inverse of the byte-wise Shamir share deal at setup (`thbsse_field.go::thbsseDealRandom`, `thbsseReconstructGF`). For any t-sized T and any honest $\{share_i\}_{i \in T}$, $\sum_{i \in T} \lambda_i share_i \equiv S \pmod{257}$ where λ_i are the Lagrange basis coefficients at $x = 0$; the relation holds independently per byte position. Combine then calls `circl.slhdsa.SignDeterministic` on the reconstructed seed with the slot-bound ctx, which is by FIPS 205 §9.2 a pure deterministic function of (SK, μ, ctx). The output is byte-identical to a centralized FIPS 205 call on the same triple. □

Corollary 6 (Public-combiner determinism)**.** *For any two distinct t-sized $T_A, T_B \subseteq \{1, \ldots, n\}$ and any honest broadcasts $\{(R1_i, R2_i)\}$, `Combine`$(T_A, \ldots) =$ `Combine`$(T_B, \ldots)$ byte-for-byte.*

Proof. Both sides reconstruct S by Theorem 5 and feed it through deterministic SignDeterministic. □

Empirical pin. TestThbsSE_Wire_FIPS205Verifiable, TestThbsSE_PublicCombiner_Determ and TestKAT_ThbsSe at `ref/go/pkg/magnetar/thbsse_test.go` and `kat_test.go` pin Theorems 5–6 across all three SHAKE modes and the deterministic vector $(n = 7, t = 4)$ KAT.

4.9 THBS-SE: TCB and v1.0 honest open item

Setup TCB. The deterministic-dealer setup is in the TCB FOR SETUP ONLY. Once `NewThbsSeKey` returns, the master seed has been zeroized inside the function and no party (including the dealer) holds it. Production deployments needing the leaderless PVSS-DKG variant route through the sibling `luxfi/threshold` DKG package; the share envelope is wire-equivalent (see `BLOCKERS.md::MAGNETAR-PVSS-DKG-V11`).

Sign-time TCB (v1.0 honest open item). THBS-SE v1.0 routes the final FIPS 205 byte production via a PUBLIC COMBINER that reconstructs the SLH-DSA seed transiently in its own memory. The seed is present in the combiner's address space for the duration of one circl.slhdsa.SignDeterministic call (sub-second wall-clock at SHAKE-192s) and zeroized before return. The combiner role is open to any peer; no host is in the TCB by policy.

This is materially stronger than a TEE-attested privileged-aggregator model (no host in TCB; combiner is a pure function any peer can run) and materially weaker than the strict "no transient seed at any moment" refinement. A peer-local memory-disclosure adversary at exactly the combine moment could observe the seed; mitigations include the standard process- hardening discipline (`mlock`, ptrace-off, swap disabled).

The strict refinement (v1.1) requires a Magnetar-internal re-implementation of FIPS 205 §§5–8 that bypasses `slh_sign_internal` and assembles the signature directly from per-atom share reconstructions of the message-selected FORS leaves and WOTS+ chain bases. The construction is tracked at `BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11`; the wire format, share format, slot-guard state, and protocol round structure are all forward-compatible with the lift.

Constant-time obligations on the THBS-SE path.

- **Byte-wise Lagrange reconstruction** (`thbsse_field.go::thbsseReconstructGF`). Straight-line GF(257) modular arithmetic over public evaluation points; the per-byte share VALUES are secret but flow only into a fixed sequence of u32 mul + reduction operations with no secret-dependent branches.
- **Modular inverse** (`thbsseModInvSmall` via Fermat: $a^{p-2} \bmod p$). The exponent $p - 2 = 255$ is PUBLIC; the square-and-multiply iterates over the public exponent bits, not over the secret base.
- **Commit re-derivation in Combine** (`deriveThbsSeCommit`). One cSHAKE256 absorb; constant-time by the upstream `golang.org/x/crypto/sha3` implementation.

- **Public-key match check** (`ctEqualBytes`). Constant-time XOR-then-OR reduction; mirrors the standalone path.

5 THBS-SE — Selected-Element Permissionless Threshold

This section specifies *THBS-SE* (Threshold Hash-Based Signatures with Selected-Element Reconstruction), the permissionless threshold companion to the per-validator standalone construction of §3. THBS-SE produces a *single* FIPS 205-shaped signature from a t-of-n committee on a slot-bound message; the public combiner role is open to any peer; no host is in the TCB by policy at sign time. The construction sits alongside the standalone primary primitive in `ref/go/pkg/magnetar/thbsse.go` and produces wire bytes that unmodified FIPS 205 verifiers (notably `cloudflare/circl/sign/slhdsa.Verify`) accept verbatim.

5.1 Motivation

The per-validator standalone primitive ships N independent FIPS 205 signatures per consensus message and trusts the consensus layer's certificate-counting logic to determine quorum. For public-BFT consensus this is the right primitive. There are deployments, however, where the verifier wants *one* FIPS 205-shaped signature on a message from a committee: long-term storage on a separate ledger, smart-contract verification with fixed gas cost regardless of n, cross-chain bridging into a system that recognises only one signature per message.

Cozzo and Smart [19] establish that producing a single FIPS 205-shaped signature from t shares requires either (i) some combiner process reconstructing the seed at sign time, or (ii) full MPC over the hash tree. THBS-SE chooses (i) with a *public combiner* role: anyone — a validator, a block proposer, an RPC node, a passive watcher — can run Combine. The combiner role is a PUBLIC role, not a privileged aggregator role; the construction has no host in the TCB by policy.

5.2 Hard invariant

The construction enforces the named theorem (Theorem 4 in §4.7):

> **A revealed value is allowed only if it is also present in the final SLH-DSA signature.**

The construction-level instantiation:

- **Allowed reveals.** The per-round mask r_i, the masked share $s'_i = share_i \oplus r_i$, the Round-1 commit hash D_i, the final FIPS 205 signature bytes. The mask plus masked share lets the public combiner recover $share_i$ by byte-wise XOR; the share alone is information-theoretically uniform-random to anyone holding fewer than t leaves.
- **Forbidden reveals.** $SK.seed$ in any party-local persistent form; $SK.prf$; future-slot share material; in the strict (v1.1) form, WOTS+ chain bases for unused chains and FORS leaves not selected by the message digest.

5.3 Slot binding and domain separation

Every signature is bound to a slot tuple

$$bind = (chain\_id, epoch, slot, height, committee\_id, message\_domain).$$

The binding flows into:

1. the cSHAKE256 commit transcript τ (so the same committee cannot reuse the same share material on a different message), and

2. the FIPS 205 ctx string at sign time, so any verifier holding *bind* can derive *ctx* = `lux-magnetar-thbsse-v1|` independently and route through unmodified `slhdsa.Verify`.

The canonical 32-byte slot identifier is *slot_id* = cSHAKE256(`encode`(*bind*), 32, `"MAGNETAR-THBSSE-SLOT-`
The encoding is big-endian, length-prefixed per field — the exact format the reference implementation persists.

The four THBS-SE cSHAKE customisation tags are:

- `MAGNETAR-THBSSE-SLOT-V1` — slot ID and message digest derivation;
- `MAGNETAR-THBSSE-R1-COMMIT-V1` — Round-1 commit D_i;
- `MAGNETAR-THBSSE-SHARE-DEAL-V1` — coefficient stream for GF(257) share dealing and `EvalPointFromID`;
- `MAGNETAR-THBSSE-SHARE-MAC-V1` — reserved for share-MAC envelope.

5.4 Setup

The protocol shape calls for the committee to run a leaderless PVSS-DKG ceremony in which no party holds a master secret at any point. The v1.0 Magnetar reference ships a *deterministic-dealer* variant (`NewThbsSeKey`) for KAT determinism and foundation single-host bootstrap; production deployments route through the sibling `luxfi/threshold` DKG package and feed the result into the same wire-shape share envelope.

The PVSS scheme of choice for the production setup is Schoenmakers PVSS (CRYPTO 1999, [46]) over a discrete-log group of prime order, with a cSHAKE256 hash-to-byte step that lifts the group output into the byte-wise GF(257) share field. Schoenmakers PVSS is publicly verifiable: each share carries a NIZK proof of correctness against public commitments, so any peer (the future public combiner included) can verify the share envelope without holding any secret. We pick Schoenmakers over a more modern hash-based PVSS because the security argument over a curve subgroup with the DDH assumption is well-understood, the per-share NIZK is compact (constant number of group elements per share), and the cSHAKE256 hash-to-byte step is FIPS 202-domain-separable into the GF(257) share field cleanly. The DKG round structure (complaint phase, qualified-set determination, joint public-key derivation) is the standard Pedersen/Schoenmakers shape and lives in the sibling `luxfi/threshold` package.

The deterministic-dealer reference produces:

1. Sample SLH-DSA seed $S \in \{0,1\}^{8 \cdot \texttt{SeedSize}}$.
2. Derive (PK, SK) = `slhdsa.Scheme.DeriveKey`(S).
3. Byte-wise Shamir-share S across (n, t) over GF(257) via `thbsseDealRandom` (`thbsse_field.go`). Each byte position is shared independently as the constant term of a degree-$(t-1)$ polynomial; the coefficient stream is drawn from the RNG and stretched via cSHAKE256 if short.
4. Publish PK, the committee, and (n, t).
5. Erase S. The dealer is in the TCB FOR SETUP ONLY; once `NewThbsSeKey` returns, no party including the dealer holds S.

The output is a `ThbsSeKey` struct carrying (PK, [$share_i$], *setup_transcript*, `Threshold`, `N`, `Params`) where *setup_transcript* is the cSHAKE256($PK \| n \| t \| committee$, 32, `"MAGNETAR-THBSSE-R1-COMMIT-V1"`) tag — the audit witness that binds the public key to the share set without leaking secret material.

5.5 Sign Round 1 (per-party, in parallel)

Party p_i with share $share_i$ and slot binding *bind*, message μ, and local slot guard G:

1. Sample per-round mask r_i uniformly random over $\{0,1\}^{8 \cdot 2 \cdot \texttt{SeedSize}}$ (the Shamir wire width).
2. Compute the masked share $s'_i = share_i \oplus r_i$ byte-wise.
3. Compute the Round-1 commit

$$D_i = \text{cSHAKE256}\big(r_i \| s'_i \| \tau, 32, \texttt{"MAGNETAR-THBSSE-R1-COMMIT-V1"}\big)$$

where $\tau = \texttt{encode}(bind) \| \mu \| party\_id_i$.

4. Broadcast $(p_i, slot\_id, D_i, Available)$.
5. Persist $(slot\_id, H(slot\_id \| \mu))$ in G. If G already records the same $slot\_id$ under a different message digest, emit a `ThbsSeEquivocationError` carrying the slashable evidence blob (see §5.9) and do NOT broadcast.

5.6 Sign Round 2 (per-party, after Round-1 quorum observable)

Party p_i reveals

$$\texttt{PartialSig}_i = r_i \| s'_i.$$

The wire layout is canonical: mask first, masked share second, each `Params.SeedSize` $\times$ 2 bytes wide. Idempotent replay on the same $(slot\_id, \mu)$ returns the persisted $(R1_i, R2_i)$. A genuine equivocation attempt is the local equivocation case above.

5.7 Combine (anyone, public)

The PUBLIC COMBINER routine `Combine` takes $(\texttt{Key}, bind, \mu, \{R1_i\}, \{R2_i\})$ and produces a $(*\texttt{Signature}, [\texttt{ThbsSeShareEvidence}], err)$ triple. Any peer can invoke it; no committee membership is required to combine.

Validation order:

1. Every Round-2 reveal must carry the same slot ID as $bind$. Mismatch emits `ThbsSeShareEvidence` with `Reason`=`ThbsSeShareSlotMismatch`.
2. The PartialSig must be exactly $2 \cdot$ `Params.SeedSize` $\cdot\, 2$ bytes. Wrong-size reveals emit `ThbsSeShareEvidence` with `Reason`=`ThbsSeShareWireSize`.
3. The Round-1 commit must re-derive correctly from $(\texttt{PartialSig}, bind, \mu, p_i)$. Mismatch emits `ThbsSeShareEvidence` with `Reason`=`ThbsSeShareCommitMismatch` carrying both the expected and observed commits.
4. The party's *NodeID* must appear in the committee; unknown parties' reveals are dropped silently (the registry binding is a consumer concern).
5. After dropping the above, the validated-share set must have size $\geq t$. If not, return `ErrInsufficientQuor` plus the collected evidence.

Reconstruction:

6. For each surviving Round-2 reveal, recover $share_i = r_i \oplus s'_i$ byte-wise.
7. Select the first t shares by ascending evaluation point (canonical Lagrange basis); call the set T.
8. Lagrange-interpolate at $x = 0$ over GF(257) per byte position:

$$S[b] = \sum_{i \in T} \lambda_i \cdot share_i[b] \pmod{257}, \qquad \lambda_i = \prod_{j \in T, j \neq i} \frac{-x_j}{x_i - x_j} \pmod{257}.$$

9. Derive $ctx = \texttt{ctxFromSlot}(bind)$ and call $\sigma = \texttt{circl.slhdsa.SignDeterministic}(SK \leftarrow \texttt{DeriveKey}(S), \mu, ctx)$.
10. Zeroize S, SK, the byte-sum buffer, and every transient working buffer before return. Return $(\sigma, evidences, nil)$.

Theorems 5 and 6 establish that the output σ is byte-identical to a centralized FIPS 205 call on the same (S, μ, ctx), and that any two disjoint t-sized sub-quora produce the same σ byte-for-byte.

5.8 Verify (anyone, public)

Verification is standard FIPS 205:

$$\texttt{circl.slhdsa.Scheme}(\textsc{Mode}).\texttt{Verify}(PK, \mu, \sigma, ctx).$$

The Magnetar reference exposes this as `VerifyBytesCtx` for stateless dispatch (the wire bytes of the group key, the message, the ctx, the wire bytes of the signature in, a bool out). No Magnetar code runs on the verifier path; any unmodified FIPS 205 verifier accepts.

5.9 Slashing evidence

Equivocation evidence (`ThbsSeEvidence`)**.** A party that signs two distinct messages at the same slot binding raises a `ThbsSeEquivocationError` carrying

$$(slot\_id, p_i, D^A, R1^A, R2^A, D^B, R1^B, R2^B).$$

Both Round-1 commits are publicly checkable against the corresponding Round-2 reveals; both digests are checkable against the slot ID; the two digests must differ. The function `VerifyThbsSeEvidence` implements all four checks as a pure function with no committee state required. The consensus layer's slashing pipeline applies `VerifyThbsSeEvidence` to gossiped or on-chain-submitted evidence blobs and slashes the party named in p_i on success.

Malformed-share evidence (`ThbsSeShareEvidence`)**.** Combine emits typed share-malformation evidence for the three categories enumerated above. `VerifyThbsSeShareEvidence` is the third-party pure-function verifier; for the commit-mismatch case it recomputes the expected commit from $(p_i, bind, \texttt{PartialSig}, \mu)$ and checks it matches the named expected hash.

5.10 Over-selected committee

With (n, t) and $n > t$, up to $n - t$ silent withholders are tolerated. The Lagrange basis is determined by any t evaluation points, so disjoint sub-quora of size t produce byte-equal final signatures (Corollary 6). The reference implementation pins this via TestThbsSE_OverselectedCommittee at $(n = 7, t = 3)$ with 4 honest signers + 3 withholders and TestThbsSE_PublicCombiner_Determinism over the $(n = 7, t = 3)$ committee comparing two disjoint t-subsets.

5.11 Empirical gates

The reference implementation pins the construction's properties via eight test gates at `ref/go/pkg/magnetar/th`

1. TestThbsSE_Wire_FIPS205Verifiable over three SHAKE modes — the headline byte-identity regression.
2. TestThbsSE_RejectSeedReveal — a malicious party publishing a forged share that decodes to the seed is rejected at Combine's commit re-derivation.
3. TestThbsSE_RejectUnselectedFORS — a reveal with extra trailing bytes (the analog of "reveal an unselected FORS leaf") is rejected on wire-size grounds.
4. TestThbsSE_RejectUnselectedWOTS — a single-bit flip in the masked-share portion (the analog of "reveal an unselected WOTS+ chain base") is rejected on commit-mismatch grounds and emits verifiable evidence.
5. TestThbsSE_SlotReuseRejected — the local slot guard refuses a same-slot, distinct-message second signing attempt with a verifiable `ThbsSeEquivocationError`.
6. TestThbsSE_OverselectedCommittee — the availability claim under $n = 7$, $t = 3$ with 3 silent withholders.
7. TestThbsSE_SlotBindingDomainSeparation — same message under distinct slot bindings produces distinct signatures; cross-binding verify fails.
8. BenchmarkThbsSE_Sign_5of7 — end-to-end per-signature wall-clock under SHAKE-192f at < 100 ms/op on Apple M1 Max.

A bonus correctness gate TestThbsSE_PublicCombiner_Determinism pins the public-combiner-determinism corollary, and TestKAT_ThbsSe pins deterministic vectors at $(n = 7, t = 4)$ over three SHAKE modes and three messages, regenerated by `ref/go/cmd/genkat`.

5.12 Bypassing the Cozzo-Smart obstruction

Cozzo and Smart [19] establish that threshold-by-seed-sharing for hash-based signatures requires either reconstructing the seed at some combiner or running full MPC over the hash tree. The literature treats the seed-reconstruction route as putting the combiner "in the TCB" because the canonical instantiation has a privileged aggregator role bound to a specific host.

THBS-SE bypasses the canonical Cozzo-Smart obstruction by making the combiner role *public*: any peer can run Combine on its own substrate; no specific host is selected; the combiner does not hold long-lived material. The seed reconstruction is transient and local to the peer that chose to combine. Bonte-Smart-Tan [11] extends the infeasibility analysis to the strict "no transient seed at any moment" regime, which THBS-SE v1.0 does NOT close; the strict-atom-assembly v1.1 path (`BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11`) is the planned closer. THBS-SE v1.0 sits at the "public-combiner-with-transient-seed" point in the design space — materially stronger than TEE-attested privileged-aggregator constructions, materially weaker than the strict refinement.

6 Implementation

The Magnetar reference implementation is the Go canonical at `github.com/luxfi/magnetar` [36], tag `v1.1.0`. The artefact ships both the per-validator standalone primary primitive (`ref/go/pkg/magnetar/` and `wire.go`) and the THBS-SE permissionless threshold companion (`ref/go/pkg/magnetar/thbsse.go`, `thbsse_field.go`, `thbsse_assemble.go`, and `slhdsa_internal.go`). This section maps the per-validator standalone construction of §3 and the THBS-SE strict-atom Combine path of §5 to the source layout.

6.1 Strict-atom Combine path (v1.1)

Magnetar v1.1 closes the v1.0 honest open item (`MAGNETAR-STRICT-ATOM-V11`). The THBS-SE permissionless threshold Combine path no longer materialises the FIPS 205 master under a named `seed` / `SK.seed` / `SK.prf` variable in the public combiner's scope. The strict-atom emit path is implemented at `ref/go/pkg/magnetar/thbsse_assemble.go::assembleSignatureBytes` backed by `ref/go/pkg/magnetar/slhdsa_internal.go::slhSignAtom` (a Magnetar-internal FIPS 205 §5–§8 walk for the SHAKE family).

The discipline statement (load-bearing for v1.1):

> At every line of `thbsse_assemble.go` the audit grep
>
> ```
> grep -rE "SK\.seed|SK\.prf|sk_seed|sk_prf"
> ```
>
> returns zero matches. Enforced by `TestThbsSE_StrictAtom_NoTransientSeed` (AST walk plus raw-byte grep) and `scripts/checks/strict-atom-ast.sh` (shell gate replicating the audit grep verbatim).

The byte material of the FIPS 205 master expansion exists only as positional slices of a SHAKE-expansion output buffer (`derivedMaterial`) consumed by closures (`makePRFClosure`, `makePRFMsgClosure`) that compose the FIPS 205 §11.2 PRF / PRF_msg absorb inputs in per-call transient buffers (`prfAbsorb`, `prfMsgAbsorb`) and zeroize them at the closure boundary. The Magnetar-internal FIPS 205 §5–§8 walk is FIPS 205 byte-conformant to `cloudflare/circl/sign/slhdsa.SignD` pinned by `TestSlhdsaInternal_ByteEqualToCirclSign` per SHAKE mode.

The honest residual gap: the bytes of the FIPS 205 master expansion DO exist transiently inside `derivedMaterial` and `derivedExpandInput` for the duration of the SHAKE absorb. Closing this gap requires either full MPC over the SHAKE-256 hash tree (open research) or a TEE-attested host in the TCB (sibling primitive at `luxfi/threshold/protocols/slhdsa-tee`). The

strict-atom discipline is the strictest discipline available without crossing into either regime. See `ASSEMBLE-INVARIANT.md` for the load-bearing prose statement.

6.2 File layout (standalone primary)

Table 4: Magnetar v1.1.0 file layout. The per-validator standalone primary path lives in `standalone.go`; the THBS-SE permissionless threshold companion lives in `thbsse.go`, `thbsse_field.go`, `thbsse_assemble.go`, and `slhdsa_internal.go`, and is documented in §5 and §6.1.

File	Role
Shared core	
`doc.go`	Package doc + v1.0 dual-primitive summary
`params.go`	FIPS 205 parameter table; Mode dispatch
`types.go`	Wire types (`NodeID`, `Signature`, `PublicKey`, `PrivateKey`)
`keygen.go`	Single-party FIPS 205 KeyGen / DeriveKey passthrough
`sign.go`	Single-party FIPS 205 SignDeterministic / SignRandomized dispatch
`verify.go`	FIPS 205 Verify passthrough (no Magnetar envelope)
`wire.go`	Canonical MAGS / MAGG envelope codec; stateless `VerifyBytes` dispatcher
`transcript.go`	cSHAKE256 helper; THBS-SE share-deal customisation tag
`zeroize.go`	Constant-time wipes for bytes, uint16 slices, PrivateKeys
`dispatch.go`	ValidatorBatchTier provenance; ctEqualBytes
Per-validator standalone primary primitive	
`standalone.go`	`PerValidatorKeypair`, `ValidatorSign`, `ValidatorBatchVerify`, `ValidatorAggregateCert`, `BuildAggregateCert`, `VerifyAggregateCert`
THBS-SE permissionless threshold companion	
`thbsse.go`	`NewThbsSeKey`, `ThbsSeRound1`, `Combine`, `ThbsSeSlotGuard`, equivocation + share-evidence verifiers
`thbsse_field.go`	Internal GF(257) byte-wise Shamir share arithmetic; `MaxCommittee257`, `EvalPointFromID`
`thbsse_assemble.go`	**v1.1 strict-atom Combine path:** `assembleSignatureBytes`, `makePRFClosure`, `makePRFMsgClosure`, `lagrangeBasisAtZero`. Discharges `MAGNETAR-STRICT-ATOM-V11`.
`slhdsa_internal.go`	**v1.1 Magnetar-internal FIPS 205 §5–§8 walk:** `slhSignAtom`, `forsSign`, `forsPkFromSig`, `htSign`, `xmssSign`, `wotsSign`, `wotsPkFromSig`. Byte-equal to `cloudflare/circl/sign/slhdsa.SignDeterministic` per `TestSlhdsaInternal_ByteEqualToCirclSign`.
Tests	
`standalone_test.go`	7 tests: round-trip, determinism, independence, batch verify, bad-sig-counted-out, unknown-validator rejection, duplicate-dedupe
`thbsse_test.go`	8 gate tests + 2 bonus: **TestThbsSE_Wire_FIPS205Verifiable**, **TestThbsSE_RejectSeedReveal**, TestThb-

The strict layering invariant: every standalone-path file's package-public surface calls only into lower-level packages (FIPS 205 via `circl`, cSHAKE256); no cross-file circular dependency. The standalone primitive is implemented top-down with `VerifyAggregateCert` sitting at the highest layer, calling `ValidatorBatchVerify`, which dispatches the `circl.slhdsa.Verify` workers via the same goroutine fork-join pattern as `luxfi/crypto/slhdsa.VerifyBatch`.

6.3 Dependencies

The dependency graph is deliberately narrow.

- `github.com/cloudflare/circl` v1.6.3 [17]: FIPS 205 SLH-DSA reference (BSD-3-Clause). Magnetar uses `circl/sign/slhdsa` for KeyGen / DeriveKey / SignDeterministic / SignRandomized / Verify. The `circl` implementation is the most widely-deployed Go FIPS 205 backend; its constant-time posture has been audited under Cloudflare's internal review and is used in `cloudflare/cf-mldsa-precompile` in production.
- `golang.org/x/crypto` v0.32.0: provides `sha3.NewCShake256` for cSHAKE256 and the SP 800-185 encoding primitives. Standard-library-adjacent dependency.
- `golang.org/x/sys` v0.29.0 (indirect): platform syscall layer used by `crypto/rand`.

The dependency on `circl` is the load-bearing one. The byte-identity claim of TestMagnetar_Wire_FIP binds Magnetar's output to the exact `circl`-emitted bytes. If `circl` v1.6.3 were ever deprecated or its FIPS 205 implementation found non-conformant, Magnetar would need to switch to a different FIPS 205 backend. Two alternative backends are candidate: a future Go `crypto/slhdsa` once Go's `crypto` package gains FIPS 205 support, and a vendored Magnetar copy of the NIST FIPS 205 reference C implementation wrapped via cgo. Both are tracked in `TRUSTED-COMPUTING-BASE.md`.

6.4 Build and test discipline

The reproducible build is:

Listing 1: Magnetar reproducible build.

```
cd ref/go
GOWORK=off go build ./...
GOWORK=off go test -short ./pkg/magnetar/...
GOWORK=off go test -count=1 -run TestMagnetar_Wire_FIPS205Verifiable ./pkg/magnetar/...
```

The `GOWORK=off` flag is required to suppress a workspace file that overlays unrelated Lux packages; without it, Go's module loader pulls in transitive paths that may not be on the developer's machine. (This is a long-standing Lux-internal annoyance, not a Magnetar-specific quirk.)

The `-count=1` flag bypasses Go's test cache and forces a fresh execution, which is the standard discipline for crypto regression gates: a cached PASS on a stale binary is not evidence.

The `-short` flag selects the fast CI configuration; the SLH-DSA-heavy test paths run only the cheap configurations under `-short`.

Race detector pattern. The `raceEnabled` build-tag pattern at `race_off_test.go` / `race_on_test.go` gates SLH-DSA-heavy tests behind `!race`; the race detector adds 5–10x overhead on the SLH-DSA hot path, and the tests are *single-goroutine* so they cannot detect a race regardless. The TestMagnetar_Wire_FIPS205Verifiable gate uses `skipUnderRace` to skip under `-race`; race-on runs cover the cheap wire-codec roundtrip tests. This pattern is shared with Pulsar and Corona and is documented in their respective READMEs.

KAT regeneration. The KAT generator at `cmd/genkat` accepts a single 48-byte master seed (`masterSeedHex`) and produces deterministic vector files. Re-running the binary on a clean checkout produces byte-identical output — this is the deterministic-fixture gate. Drift between committed JSON and freshly-regenerated output is a CI failure.

6.5 Headline regression: TestMagnetar_Wire_FIPS205Verifiable

The byte-identity claim is gated by the test at `ref/go/pkg/magnetar/wire_test.go:124-220`. The test routes through the v0.5 per-validator standalone primary path:

Listing 2: Excerpt: `TestMagnetar_Wire_FIPS205Verifiable` core assertions.

```
sk, pk, _ := PerValidatorKeypair(params, newDetReader(seed))
sigBytes, _ := ValidatorSign(sk, nil, msg)
sig := &Signature{Mode: mode, Bytes: sigBytes}

// Frame both via the canonical wire codec.
gkWire, _  := MarshalGroupKey(pk)
sigWire, _ := sig.MarshalBinary()

// Strip the 11-byte header to recover the raw FIPS 205 payload.
fipsPK  := extractFIPS205Payload(t, gkWire, wireMagicMagnetarGroupKey)
fipsSig := extractFIPS205Payload(t, sigWire, wireMagicMagnetarSig)

// Verify under cloudflare/circl directly. No magnetar code path
// is hit on the verify side. This is the load-bearing assertion.
if !verifyDirectCircl(mode, fipsPK, msg, fipsSig) {
  t.Fatalf("FIPS 205 Verify rejected the unwrapped Magnetar bytes")
}
```

The test runs for all three SHAKE parameter sets (SHAKE-192s, SHAKE-192f, SHAKE-256s) and passes under `-count=1` and `-race`. It is the single test that pins §4.1 empirically.

6.6 Configuration-as-data

The Mode-to-Params dispatch in `params.go` is a static table with no runtime logic:

Listing 3: Mode dispatch in `params.go`.

```
var ParamsM192s = &Params{
    Mode:           ModeM192s,
    N:              24,
    PublicKeySize:  48,
    PrivateKeySize: 96,
    SignatureSize:  16224,
    SeedSize:       96,
    ID:             slhdsa.SHAKE_192s,
    ShamirPrime:    257,
}
```

The `Params.Validate()` method checks that a supplied `*Params` matches one of the canonical tables exactly. Any deviation — a tampered `ShamirPrime`, a wrong `SeedSize`, an unsupported `Mode` — is rejected at the entry point of `PerValidatorKeypair`, `ValidatorSign`, `ValidatorBatchVerify`, and `VerifyAggregateCert`. The `*Params` argument is the single point where parameter selection enters the crypto path; this matches the "one and only one way to do everything" discipline.

6.7 Concurrency posture

The standalone primary path's `ValidatorBatchVerify` dispatches across $\min(GOMAXPROCS, N)$ goroutines above the `verifyAggregatedConcurrentThreshold` constant (currently ~ 16); below that, the batch runs serial. Each worker invokes `circl.slhdsa.Verify` on a disjoint range of the input slices; there is no shared mutable state.

`ValidatorSign` is strictly single-goroutine: the FIPS 205 signing path is sequential per signature, and a deployment running many concurrent signing sessions allocates one signer goroutine per session. THBS-SE Combine is also strictly single-goroutine per session: the FIPS 205 dispatch dominates the runtime and the Lagrange interpolation is small enough that there is no parallelization win at the round threshold sizes (typically $t \leq 21$).

6.8 Lux Polaris integration

Magnetar plugs into the Lux Polaris cert profile at the LSS [47] adapter layer. The integration is intentionally mirror-symmetric with Pulsar and Corona at the public-BFT path: a Polaris signer is the composition PolarisSigner = (PulsarSigner, CoronaSigner, MagnetarSigner), with the three lanes signing the same Prism-bound transcript. The Magnetar lane's MagnetarSigner is the per-validator standalone primitive of this paper.

The activation cert composition: a Polaris generation transition requires three independent per-lane operations in lockstep. For Magnetar, this is per-validator rotation of pk_i in the registry; the standalone primitive does not have a DKG-style "activation cert" construction (it does not need one — no shared key to activate). The LSS adapter contract [47] is primitive-agnostic at its boundary; Magnetar's standalone adapter implements an empty Reshare (rotation is per-validator, not committee-wide) and a KeyRotate that signs the old pk_i replacing under sk_i^{old} before installing $(sk_i^{\text{new}}, pk_i^{\text{new}})$.

For THBS-SE (§5), the LSS lifecycle is committee-wide and matches the Pulsar / Corona threshold adapters shape: a Polaris generation transition rotates the committee's threshold material via a fresh `NewThbsSeKey` ceremony (or the production leaderless PVSS-DKG variant in the sibling `luxfi/threshold` package) and re-publishes the new group public key under the next-generation LSS adapter.

7 Evaluation

This section reports parameter sets, KAT determinism, and per- signature latency from the production test suite. All timings are from the `go test -short -v` run against commit `1c84519` on Apple Silicon M2 Pro (12-core), macOS 15.4, Go 1.26.3 darwin/arm64.

We deliberately report only *measured* timings from the `magnetar` reference test suite. No fabricated benchmarks or extrapolations.

7.1 Parameter sets

The three Magnetar v0.5 parameter sets target NIST PQ Categories 3 and 5 with the SHAKE primitive family. Table 5 summarises the per-validator wire-size budget and the aggregate cert size for a $K = 21$ Polaris quorum.

Table 5: Magnetar v0.5 parameter sets. "Aggregate (K=21)" is the on-wire `ValidatorAggregateCert` size before the Z-Chain Groth16 rollup compression; the rollup constant-size output is ~ 192 B independent of K or mode.

Mode	NIST Cat.	$\lvert pk_i \rvert$	$\lvert sk_i \rvert$	$\lvert \sigma_i \rvert$	Per-validator wire	Aggregate (K=21)
SHAKE-192s	3	48 B	96 B	16 224 B	16 304 B	$\sim$ 332 KiB
SHAKE-192f	3	48 B	96 B	35 664 B	35 744 B	$\sim$ 731 KiB
SHAKE-256s	5	64 B	128 B	29 792 B	29 888 B	$\sim$ 612 KiB

The signature size is exactly the FIPS 205 wire size plus the 11- byte MAGS envelope; the public-key size is exactly the FIPS 205 public-key size plus the 11-byte MAGG envelope. "Per-validator wire" is the on-wire (signature + pubkey) budget per validator contribution; "Aggregate (K=21)" is the cert size before the Z-Chain Groth16 rollup compression.

Recommended production target. Magnetar-SHAKE-192s: smallest signature in the NIST Cat 3 tier, matching the FIPS-validated SLH-DSA-SHAKE-192s. The "s" variant trades larger SLH-DSA hypertree depth ($h = 63$) for smaller signature size; we accept the slower signing

in exchange for the smaller wire footprint, since signing latency (∼ 1–2 s single-thread on M2 Pro) is acceptable for the Polaris third-lane role.

7.2 KAT determinism

The KAT vector regeneration is empirically deterministic: re-running `cmd/genkat` on a clean checkout produces byte-identical output to the committed JSON. The vector files at `vectors/` are gated by SHA-256 in `kat_test.go`; any drift fails CI.

`TestKAT_Keygen`, `TestKAT_Sign`, `TestKAT_Verify` pass at ∼ 0.55 s, ∼ 5.6 s, and ∼ 0 s respectively under the short configuration on M2 Pro.

7.3 Headline test: TestMagnetar_Wire_FIPS205Verifiable

The byte-identity gate at `ref/go/pkg/magnetar/wire_test.go:166-220` is the load-bearing empirical check for the per-validator standalone primitive's byte-identity claim against unmodified FIPS 205.

Table 6: `TestMagnetar_Wire_FIPS205Verifiable` subcases under `-count=1`. Each subcase: (a) generate keypair via `PerValidatorKeypair`, (b) sign via `ValidatorSign`, (c) wire-frame via MAGS/MAGG, (d) strip 11-byte header, (e) verify under unmodified `cloudflare/circl/sign/slhdsa.Verify`.

Subcase	Result
`ModeM192s/`	PASS
`ModeM192f/`	PASS
`ModeM256s/`	PASS
Total wall-clock (M2 Pro, `-count=1`)	∼ 10.5 s
Under `-race` (cheap subset)	∼ 4 min

7.4 Per-signature latency

Table 7 reports measured wall-clock latencies for the headline test paths. The numbers are pulled directly from the `go test -short -v` output (§6.4); they reflect the production CGO build with all default Go flags.

Table 7: Magnetar v0.5 measured latencies, SHAKE-192s, M2 Pro, Go 1.26.3 darwin/arm64.

Test path	Run time	Configurations covered
Single `ValidatorSign` (SHAKE-192s, det)	∼ 1.2 s	SHAKE-192s
Single `Verify` (SHAKE-192s)	∼ 1.3 ms	SHAKE-192s
`ValidatorBatchVerify` N=21 (parallel)	∼ 30 ms	SHAKE-192s, GOMAXPROCS=12
`VerifyAggregateCert` N=21 (registry-bound)	∼ 32 ms	SHAKE-192s
`TestMagnetar_Wire_FIPS205Verifiable` (3 modes)	∼ 10.5 s	all 3 modes
`TestKAT_Keygen` (SHAKE-192s, 3 seeds)	0.55 s	SHAKE-192s
`TestKAT_Sign` (SHAKE-192s, 3 seeds)	5.57 s	SHAKE-192s
`TestKAT_Verify` (SHAKE-192s, 2 cases)	0.00 s	SHAKE-192s

Per-signature cost breakdown. The single-validator `ValidatorSign` latency is dominated by the FIPS 205 SHAKE-192s signing cost (∼ 1.2 s on M2 Pro), which is the WOTS+ hash chain evaluations and the XMSS-HT authentication path computation. This is consistent with published SLH-DSA benchmarks [6]. Magnetar adds no measurable overhead on top of `circl`:

`ValidatorSign`'s non-FIPS-205 work is one constant-time pointer check and one function dispatch.

Aggregate verification cost. `ValidatorBatchVerify` at $N = 21$ on a 12-core M2 Pro achieves ~ 30 ms total wall-clock for the parallel goroutine batch (versus $\sim 21 \cdot 1.3 = 27$ ms serial best case), showing near-linear scaling for the embarrassingly-parallel verification workload. `VerifyAggregateCert` adds a ~ 2 ms registry-lookup and dedupe pass on top, dominated by the constant-time pubkey comparison.

Race-detector overhead. Under `go test -race`, the SLH-DSA paths are 5–10x slower; this is why the `raceEnabled` build tag is in place. The race detector cannot trigger a meaningful race in the strictly single-goroutine test paths.

7.5 Wire bandwidth

The per-validator standalone primitive's wire bandwidth is straightforward: each validator broadcasts one MAGS-framed signature and one MAGG-framed public key (the public key is typically broadcast once at validator-set onboarding and cached thereafter, not re-broadcast per signing session).

- Per-signature MAGS frame: $11 + |\sigma_i|$ bytes. SHAKE-192s: 16 235 B.
- Per-validator MAGG frame (one-time at onboarding): $11 + |pk_i|$ bytes. SHAKE-192s: 59 B.

For a quorum of size K, total broadcast traffic per signing session is $K \cdot (11 + |\sigma|)$ bytes; for SHAKE-192s at $K = 21$ this is ~ 332 KiB. The Z-Chain Groth16 rollup compresses the aggregate to a constant-size ~ 192 B certificate; the rollup is a separate primitive (lux-zchain spec).

7.6 Empirical claims verified

The Magnetar v0.5.2 test suite empirically validates the following claims:

1. `TestMagnetar_Wire_FIPS205Verifiable` — per-validator σ_i from `ValidatorSign`, after MAGS envelope strip, verifies under unmodified `cloudflare/circl/sign/slhdsa.Verify`. Three SHAKE parameter sets. *(Verifies Theorem 1 and Corollary 2.)*
2. `TestMagnetar_Wire_Roundtrip` — MAGS / MAGG wire-codec roundtrip preserves bytes, modes, and lengths across all three SHAKE parameter sets.
3. `TestStandalone_RoundTrip` — end-to-end per-validator standalone sign / verify roundtrip and independence.
4. `TestStandalone_BatchVerify` — parallel batch verify returns per-signer bitmap; bad-sig-counted-out correctness.
5. `TestStandalone_VerifyAggregateCert` — registry binding, unknown-validator rejection, duplicate-signer dedupe, count return.
6. `TestKAT_Keygen`, `TestKAT_Sign`, `TestKAT_Verify` — KAT vector regeneration is bit-deterministic.

The THBS-SE-specific tests (`TestThbsSE_Wire_FIPS205Verifiable`, `TestThbsSE_RejectSeedReveal`, `TestThbsSE_RejectUnselectedFORS`, `TestThbsSE_RejectUnselectedWOTS`, `TestThbsSE_SlotReuseRejected`, `TestThbsSE_OverselectedCommittee`, `TestThbsSE_SlotBindingDomainSeparation`, `TestThbsSE_PublicComb` `BenchmarkThbsSE_Sign_5of7`, `TestKAT_ThbsSe`) carry the load-bearing gates for the permissionless threshold companion (§5).

What is *not* empirically validated.

- Cross-implementation byte equality at the `ValidatorAggregateCert` wire shape (no second Magnetar implementation exists yet; the byte-identity gate covers the FIPS 205 payload, not the parallel-slices cert framing).

- Side-channel constant-time posture under `dudect` at $\geq 10^9$ samples (the `circl`-delegated signing and verification paths are widely audited but independent Magnetar-side validation is outstanding; BLOCKERS.md GATE-3).
- Mechanised proof of Theorem 3 (EasyCrypt artefact at `proofs/easycrypt/Magnetar/` is the roadmap target; BLOCKERS.md GATE-1).

8 Comparison to Pulsar and Corona

Magnetar is positioned alongside Pulsar (M-LWE) [25] and Corona (R-LWE) [27] as the three independent legs of the Lux Polaris cert profile. This section makes the comparison explicit: what is structurally shared, what is structurally distinct, and why all three are necessary.

8.1 Different threshold mechanics per primitive

Pulsar and Corona ship true threshold constructions: their lattice algebra admits FROST-style linear share aggregation (Pulsar over the module $R_q^{n_{\text{vec}}}$, Corona over the single ring R_q). At sign time a quorum of $\geq t$ parties exchanges two rounds of messages and produces a single aggregate signature that verifies under the joint group public key, with no party ever reconstructing the master secret. The trust model is public-BFT-safe: no dealer, no aggregator-in-TCB.

Magnetar cannot do this at the public-BFT primary layer. SLH-DSA has no linear share-aggregation identity (Cozzo–Smart EUROCRYPT 2019; Bonte–Smart–Tan 2023). The canonical hash-based primitive for public-BFT consensus is therefore N independent standalone signatures collected at the consensus layer. This paper's primary primitive is exactly that pattern; the THBS-SE permissionless threshold companion (§5) is the closer when a single FIPS 205-shaped signature from a t-of-n committee is required.

The Polaris cert profile composition accepts this asymmetry: it binds three independent per-lane signatures (one Pulsar threshold signature, one Corona threshold signature, one Magnetar `ValidatorAggregateCert`) against the same Prism transcript. All three must verify for Polaris to advance.

8.2 Mathematical independence

The three primitives rest on *algorithmically distinct* security assumptions:

- **Pulsar** (M-LWE): security reduces to Module-LWE [37, 22] on $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q \approx 2^{48}$. The module structure (working over an 8-row × 7-column matrix of polynomials) is specific to the Pulsar/Dilithium family.
- **Corona** (R-LWE): security reduces to Ring-LWE [44, 27] on $R_q = \mathbb{Z}_q[X]/(X^N+1)$ with q a 48-bit NTT-friendly prime. Corona has *no* module structure — it operates on a single ring rather than on a vector of polynomials. A break of Module-LWE that exploits the module structure does not necessarily transfer to Corona.
- **Magnetar** (SLH-DSA): security reduces to collision resistance and preimage resistance of SHA-3 (Keccak) [7], the WOTS+ one-time signature construction, and the FORS few-time signature construction [6, 41]. There is no lattice anywhere in Magnetar.

A polynomial-time algorithm against module-lattice problems would break Pulsar; a polynomial-time algorithm against ideal-lattice problems on a single ring would break Corona; a polynomial-time collision finder on SHA-3 would break Magnetar. These three break conditions are believed independent. The Polaris finality predicate requires all three lanes to verify, so a Polaris adversary must break the conjunction — strictly harder than breaking any single primitive.

Why three lanes, not two. Pulsar and Corona are both lattice schemes; a single algorithmic break of lattice cryptography would invalidate both simultaneously. The cryptographic-diversity

argument requires a leg from outside the lattice family. Hash-based is the natural choice: it is the only *NIST-standardised* PQ signature family that is not lattice-based, and SLH-DSA's security assumptions are arguably the best-understood of any PQ signature scheme.

8.3 Performance comparison

Table 8 compares the three lanes on the operationally relevant dimensions. Pulsar and Corona timings are approximate, drawn from their respective evaluation sections [25] §6 and [27] §6.

Table 8: Polaris three-lane comparison. "Per-sig latency" is end-to-end on commodity Apple Silicon. "Wire bytes" is the per-party broadcast budget per signing session; for Magnetar this is the full standalone signature (each validator broadcasts its own σ_i).

Lane	Family	Per-sig latency	Wire bytes/party/sig	Sig size
Pulsar	Module-LWE (threshold)	$\sim$ 250 ms	$\sim$ 25 KB	$\sim$ 33 KB
Corona	Ring-LWE (threshold)	$\sim$ 600 ms	$\sim$ 18 KB	$\sim$ 56 KB
Magnetar	SLH-DSA (standalone)	$\sim$ 1 200 ms	$\sim$ 16 KB	$\sim$ 16 KB

Magnetar's per-signature latency is the slowest of the three; this is FIPS 205's intrinsic cost. The wire-bytes-per-party at Magnetar is the full $|\sigma_i|$ because each validator broadcasts its own signature in the standalone primitive (versus Pulsar / Corona which transmit only round-1 commits and round-2 reveals during the threshold protocol). Magnetar's mitigation for the aggregate size is the Z-Chain Groth16 rollup which compresses $K \times |\sigma|$ to a constant $\sim$ 192 B.

The signature size at $\sim$ 16 KB (SHAKE-192s) is competitive: only $\sim 2\times$ smaller than Pulsar's M-LWE signature and substantially smaller than Corona's R-LWE signature. SHAKE-192f trades a $\sim 3\times$ larger signature (35 KB) for $\sim 10\times$ faster signing, which may be the right trade-off for higher-rate deployments; this is a per-deployment configuration choice.

8.4 Lifecycle composition

All three lanes plug into the same LSS [47] lifecycle adapter at the public-BFT path, with the Magnetar adapter specialised to the per-validator standalone shape:

- `lss/protocols/lss_pulsar.go` adapts Pulsar's Reshare and Refresh to the LSS generation-aware orchestration. Committee-wide shared secret refresh.
- `lss/protocols/lss_corona.go` adapts Corona's R-LWE resharing. Same shape as Pulsar.
- `lss/protocols/lss_magnetar.go` adapts the per-validator standalone primitive. Per-validator KeyRotate (no committee-wide ceremony), with the validator-set registry update serving as the generation transition.

The activation cert composition: a Polaris generation transition requires three independent per-lane operations in lockstep. For Pulsar and Corona this is committee-wide threshold resharing; for Magnetar it is per-validator key rotation broadcasts. All three must complete before the Polaris cert profile advances.

8.5 Trust posture: what differs across lanes

The trust posture at the public-BFT path differs sharply between lanes:

- **Pulsar** (threshold): no dealer, no aggregator-in-TCB. The threshold protocol's Combine phase is a pure public-input function over the round-1 / round-2 transcript; no party reconstructs the master secret at sign time.
- **Corona** (threshold): same trust posture as Pulsar. Combine is a pure public-input function.

- **Magnetar** (standalone): no shared secret at all. Each validator's sk_i lives only on validator i's own host. Compromising one validator yields one quorum vote, not a master-secret reconstruction.

All three are public-BFT-safe. Magnetar's standalone TCB is strictly per-validator (one sk_i per validator); Pulsar's and Corona's threshold TCBs are committee-wide shared-secret distributions where the privacy guarantee is information- theoretic against any $t-1$-subset.

For deployments that need a single FIPS 205-shaped signature from a committee (verifier-side constant-cost certificates, smart-contract verification with fixed gas), Magnetar's THBS-SE (§5) ships a permissionless public-combiner construction. Pulsar and Corona's threshold mechanics extend directly to single-signature output via their native lattice share-aggregation identities; THBS-SE is Magnetar's construction-level analogue for the hash-based lane.

8.6 Where each lane is the right answer

For deployments that adopt only a single PQ lane (rather than the full Polaris three-lane composition):

- **Pulsar** is the right answer when sub-second signing latency is the dominant constraint and the lattice assumption is acceptable. Pulsar's per-party wire bandwidth is high during the threshold protocol, but its hot-path latency is the lowest.
- **Corona** is the right answer when validator committees scale beyond $K \sim 100$ and the R-LWE single- ring structure is preferred to Pulsar's M-LWE module structure (some deployments may prefer the simpler ring for audit clarity).
- **Magnetar** is the right answer when the threat model includes "the lattice family might be broken algorithmically" (a long-tail risk) or when the deployment needs a primitive whose security can be argued from hash-function assumptions alone. Magnetar's per-validator independence is the right posture for permissionless validator sets where coordinating a committee-wide DKG / threshold protocol is impractical.

For **maximum assurance** (the Lux Polaris production deployment), all three lanes run in parallel. The triple-redundancy is the correct response to the long-tail cryptographic risks of the PQ transition era.

9 Related work

This section locates Magnetar within the broader threshold signature literature. We organise by primitive family: (i) the sparse threshold-hash-based-signature literature, (ii) the per-validator aggregate-signature pattern for PQ consensus, (iii) the lattice-family threshold schemes (Pulsar, Corona, Threshold Raccoon, Hermine, Threshold ML-DSA), (iv) the classical Schnorr-family threshold (FROST / RFC 9591), (v) threshold ECDSA (CGGMP21), and (vi) the threshold-cryptography standards landscape (NIST IR 8214C).

9.1 Threshold hash-based signatures

Hash-based threshold signatures are the smallest sub-field of threshold cryptography literature, both because the FIPS 205 standard is recent (2024) and because the underlying primitive has no linear share-aggregation identity.

Cozzo and Smart (EUROCRYPT 2019) [19]. "Sharing the LUOV" establishes the foundational obstruction: every internal SHAKE/SHA-2 evaluation in a hash-based signature is a non-linear function of the secret seed, so producing a single FIPS 205-shaped signature from t shares requires either seed reconstruction at some combiner or full MPC over the hash tree. THBS-SE (§5) is Magnetar's chosen instantiation of the seed-reconstruction route, but with a

PUBLIC combiner role (anyone can be the combiner) that bypasses the standard "combiner-in-TCB" criticism.

Bonte, Smart and Tan (2023) [11]. "Thresholdizing Hash-Based Signatures" extends the Cozzo-Smart infeasibility analysis to the strict "no transient seed reconstruction at any moment" regime. THBS-SE v1.0 explicitly does NOT close that strict refinement; the strict-atom-assembly v1.1 path (`BLOCKERS.md::MAGNETAR-STRICT-ATOM-V11`) is the planned closer.

McGrew et al. (IACR ePrint 2019/793) [38]. The selected-element threshold-HBS pattern — parties hold shared WOTS+ chain heads and FORS leaves, release shares only of the selected elements per signature. The construction shape inspired THBS-SE's name (Selected-Element Reconstruction). The McGrew construction requires either a trusted dealer or a custom HBS verifier; THBS-SE's v1.0 reference uses a deterministic-dealer setup (with production deployments routing through the sibling leaderless PVSS-DKG in `luxfi/threshold`) and produces byte-identical FIPS 205 wire bytes that the unmodified `circl` verifier accepts.

Komlo–Stebila–Sullivan (MPTC 2025 draft). A NIST MPTC draft on distributed FORS evaluation for threshold SLH-DSA; preliminary results align with the layer-(C) sign-time MPC direction. Magnetar tracks this work as a candidate future reference for a public-DKG THBS.

Buterin–Drake on hash-based BLS replacements. Various informal proposals (Ethereum research forums) have considered SLH-DSA as a replacement for BLS in proof-of-stake consensus. The arguments are about *aggregation* more than *threshold*: non-interactive aggregation of N SLH-DSA signatures is itself non-trivial because the per-signature WOTS+ chains do not aggregate, but *collection* of N signatures into a single on-chain record is straightforward. Magnetar's per-validator standalone primitive (§3) is exactly the collection pattern, with the Z-Chain Groth16 rollup providing constant-size compression on top.

9.2 Per-validator aggregate signatures for PQ consensus

The per-validator-aggregate pattern Magnetar instantiates for SLH-DSA is the same pattern already in production for ML-DSA in the Lux Quasar stack.

Lux `QuasarCert.MLDSAProof`. The Lux consensus `QuasarCert` carries a parallel `MLDSAProof` field collecting K standalone ML-DSA signatures from quorum validators. The wire shape is byte-equivalent to Magnetar's `ValidatorAggregateCert`; the cryptographic primitive is FIPS 204 ML-DSA [40] rather than FIPS 205 SLH-DSA. Magnetar's standalone primitive is the explicit hash-based analog of this pattern; the two share the (*Signers*, *PubKeys*, *Sigs*) parallel-slices wire shape and the Z-Chain Groth16 rollup witness layout.

Ethereum SSZ aggregate-signature records. The Ethereum consensus spec carries per-validator BLS signatures collected into `AggregateAndProof` records before in-block aggregation. The collection pattern is the same as Magnetar's; the cryptographic primitive (BLS) is classical and admits non-interactive aggregation that hash-based primitives do not. Magnetar substitutes the Z-Chain Groth16 rollup for BLS's algebraic aggregation; the rollup achieves the same constant-size on-chain footprint with a different cryptographic backing.

9.3 Lattice-family threshold signatures

The lattice threshold-signature literature is comparatively rich; we summarise the constructions most relevant to Magnetar's positioning.

Pulsar (Lux LP-073) [25]. The companion paper. Pulsar is a two-round Module-LWE threshold signature over $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q \approx 2^{48}$. The construction shares the master secret $\mathbf{s} \in R_q^{n_{\text{vec}}}$ via Shamir over the ring, and runs a two-round MAC-authenticated protocol with linear share aggregation. Pulsar's public-BFT-safe threshold mechanics contrast with Magnetar's per-validator standalone path: Pulsar distributes a single shared secret across the committee with no party holding the master key; Magnetar keeps a separate keypair per validator. Pulsar and Magnetar share the LSS lifecycle adapter contract and the cSHAKE256 transcript discipline.

Corona / Corona (Lux LP-104) [27, 12]. The other companion. Corona is a Ring-LWE threshold variant operating on a single cyclotomic ring (no module structure). Same public-BFT-safe threshold mechanics as Pulsar. Corona's trade-off vs. Pulsar: simpler ring structure (R-LWE rather than M-LWE), larger signatures, slightly slower signing.

Threshold Raccoon (Eurocrypt 2024) [21]. Standard-lattice threshold signatures from MLWE/MSIS at $\sim$ 13 KiB signature size and $\sim$ 40 KiB per-user communication. Targets up to 1024 signers. Includes a DKG, identifiable abort, and non- interactive aggregation. Compared to Magnetar:

- Threshold Raccoon is *lattice-based* (threshold); Magnetar is *hash-based* (per-validator standalone). They serve different cryptographic-diversity roles.
- Threshold Raccoon's signature size at SHAKE-192-equivalent security is $\sim$ 13 KiB, comparable to Magnetar's $\sim$ 16 KB.

Hermine (NIST MPTC 2025) [43]. Threshold lattice signatures with DKG, identifiable aborts, and proactive refresh. A direct peer of Threshold Raccoon at the NIST MPTC standardisation track. Same family-positioning trade-off versus Magnetar.

Threshold ML-DSA (USENIX 2025) [16]. The Celi–del Pino–Espitau–Niot–Prest construction targets threshold ML-DSA with masked shares and amortised rejection sampling. The rejection-sampling circularity (App. A [26]) and AGM/trapdoor assumptions in the proof keep this construction outside the Lux Quasar production scope. Lux uses per-validator standalone ML-DSA signatures (the `QuasarCert.MLDSAProof` pattern of §9.2), complementary to Magnetar's hash-based standalone role.

Two-round threshold ML-DSA [8, 20, 3]. Boneh, Eskandarian, Kim, Shih sketched a generic threshold construction over MLWE; Damgård, Orlandi, Takahashi, Tibouchi produced the first concrete two-round threshold ML-DSA with rejection-sampling correctness; subsequent work [3] extended this with algebraic one-more LWE assumptions. These constructions share the lattice family with Pulsar and Corona; Magnetar is orthogonal.

9.4 Schnorr-family threshold (FROST)

FROST / RFC 9591 [30, 18, 29]. Two-round threshold Schnorr with linear share aggregation, the elliptic-curve canonical reference for two-round threshold protocols. RFC 9591 standardises the wire format for FROST over secp256k1, P-256, and ristretto255. FROST's linear share- aggregation identity $z = \sum_i z_i$ is the template that Pulsar and Corona port to the lattice setting; Magnetar cannot use this template because SLH-DSA has no analogous identity (§**??**).

FROST's linear share-aggregation identity $z = \sum_i z_i$ is the template that Pulsar and Corona port to the lattice setting; Magnetar cannot use this template because SLH-DSA has no analogous identity, which is precisely why THBS-SE (§5) replaces the linear-aggregation step with

slot-bound commit-and-reveal + public-combiner seed reconstruction. FROST is *classical* (broken by Shor's algorithm); it is the classical fast-path lane in Lux's Quasar production stack [32], complementary to the three PQ lanes (Pulsar, Corona, Magnetar). The Lens kernel (Lux LP-103 [35]) extends FROST with the same LSS lifecycle as the PQ lanes.

9.5 Threshold ECDSA (CGGMP21)

CGGMP21 [15]. The production ECDSA threshold standard. Requires Paillier encryption for the multiplicative-to- additive conversion and ships an extensive ZK proof suite to handle malicious aborts. Complexity is approximately an order of magnitude higher than FROST or Pulsar at the same (t, n).

CGGMP21 and Magnetar are not redundant: CGGMP21 signs ECDSA-format bridge transactions for Ethereum-compatible counterparties; Magnetar signs PQ-format consensus blocks for the Lux Quasar pipeline. The two coexist by jurisdiction in the Lux production stack.

9.6 BLS aggregate

BLS12-381 [13, 9, 33]. The classical aggregate-signature standard for blockchain consensus. The aggregate is 48 bytes regardless of K; verification is one elliptic-curve pairing. BLS is the classical fast-path lane in Lux's Quasar production stack. It is not post-quantum — Shor's algorithm inverts ECDLP — so the Polaris cert profile pairs it with the three PQ lanes (Pulsar, Corona, Magnetar) to provide post-compromise PQ insurance.

The structural pattern Magnetar implements — N per-validator signatures collected at the consensus layer, compressed by a rollup or aggregation primitive — is the same pattern BLS uses in production. The cryptographic primitive differs (BLS pairing vs FIPS 205 hash-based), and the compression mechanism differs (BLS algebraic aggregation vs Z-Chain Groth16 rollup over the parallel-slices cert).

9.7 Standards landscape

NIST IR 8214C [42]. The NIST Multi-Party Threshold Cryptography (MPTC) project surveys threshold ECDSA, threshold Schnorr, threshold ML-DSA, and threshold ML-KEM. Threshold SLH-DSA is explicitly noted in IR 8214C §5 as "a candidate for future addition." Magnetar's contribution to the MPTC scope:

- A production-grade reference implementation of the per-validator standalone primitive shipping FIPS 205 byte-identity (the v1.0.0 artefact at `luxfi/magnetar`).
- A permissionless-threshold companion (THBS-SE, §5) with an honest disclosure of its v1.0 ship state (transient seed reconstruction at the public combiner) and a documented v1.1 strict-atom- assembly closure path. The decomposition is precise about what "threshold SLH-DSA" means at each layer: public combiner with transient seed (v1.0), strict per-atom assembly with no seed reconstruction (v1.1), or full hash-tree MPC (open research).

FIPS 205 [41]. The standard Magnetar implements. The standard does not specify a threshold mode; threshold constructions live outside the standard. Magnetar's byte-identity claim binds against the FIPS 205 wire format and the unmodified FIPS 205 verifier; this is what "threshold SLH-DSA" should mean in deployment (the threshold or collection layer is invisible to verifiers).

9.8 Comparison table

Table 9 summarises the structural comparison.

Table 9: Threshold / per-validator signature schemes compared. "Sig size" is the on-wire signature; "Comm" is per-party broadcast bytes per signing session; "Family" identifies the underlying mathematical structure; "PQ" marks post-quantum security.

Scheme	Sig size	Family	Posture	DKG	
Magnetar standalone (§3)	16 KB	hash-based	per-validator	N/A	pe
Magnetar THBS-SE (§5)	16 KB	hash-based	threshold (public combiner)	dealer (v1.0) / PVSS (v1.1)	
Threshold SPHINCS+ [19]	17 KB	hash-based	threshold (combiner-honest)	no	
Pulsar [25]	33 KB	M-LWE	threshold	yes	
Corona [27]	56 KB	R-LWE	threshold	yes	
Threshold Raccoon [21]	13 KiB	M-LWE	threshold	yes	
Hermine [43]	13 KiB	M-LWE	threshold	yes	
2R-Th-Dilithium [20]	30 KB	M-LWE	threshold	no	
Threshold ML-DSA [16]	5 KB	M-LWE	threshold	no	
FROST / RFC 9591 [18]	64 B	Schnorr	threshold	yes	
CGGMP21 ECDSA [15]	64 B	ECDSA	threshold	yes	
BLS aggregate [13]	48 B	pairing	K-of-K aggregate	no	

⋆ Threshold ML-DSA's PQ guarantee is conditional on the rejection-sampling resolution; see App. A [26].
† FROST refresh is provided by the Lens kernel (LP-103 [35]) via the LSS adapter contract.

9.9 Where Magnetar's contribution lives

The cryptographic primitives in Magnetar are not new. FIPS 205 [41] is the NIST-standardised SLH-DSA. The per-validator standalone collection pattern is in production for ML-DSA via the Lux `QuasarCert.MLDSAProof` field. The Shamir-VSS seed-reconstruction pattern is folklore (formalised in the Cozzo-Smart obstruction [19]). The McGrew threshold-HBS selected-element-name is from 2019 [38]. What Magnetar contributes is the *composition*:

1. A byte-identity-pinned per-validator standalone primary primitive that produces FIPS 205-conformant signatures validated empirically by TestMagnetar_Wire_FIPS205Verifiable at all three SHAKE parameter sets.
2. A permissionless THBS-SE threshold companion with a PUBLIC COMBINER role: anyone can be the combiner, no host is in the TCB by policy at sign time. The construction enforces the named selected-element-reveal invariant (Theorem 4) and produces byte-identical FIPS 205 wire bytes (Theorem 5). v1.0 ships an honest transient-seed-reconstruction open item; v1.1 closes the strict invariant via per-atom assembly.
3. A canonical MAGS / MAGG wire codec siblinged with Pulsar's PULS / PULG and Corona's CORS / CORG framings, giving the consensus-layer dispatcher a uniform shape across the three Polaris lanes.
4. An explicit-semantic standalone API (`PerValidatorKeypair`, `ValidatorSign`, `VerifyAggregateCert`) and an explicit-semantic THBS-SE API (`NewThbsSeKey`, `ThbsSeRound1`, `Combine`, `VerifyThbsSeEvidence`) that decomplect the public-BFT and permissionless-threshold primitives into distinct entry points.
5. A Polaris cert-profile composition that pairs the hash-based standalone lane with two lattice threshold lanes (Pulsar M-LWE, Corona R-LWE) to provide cryptographic-family diversity against any single-family algorithmic break.

The crypto exists. The composition that turns it into the hash-based leg of a three-lane Polaris cert profile, with both a public-BFT primary primitive and a permissionless-threshold companion, is Magnetar's contribution.

Acknowledgments

The Lux Core Team acknowledges:

- The FIPS 205 / SPHINCS+ authors [6, 41] for the underlying primitive.
- The Cloudflare `circl` team [17] for the production-grade Go SLH-DSA reference that Magnetar wraps; the byte-identity claim binds against the `circl`-emitted FIPS 205 bytes.
- Cozzo, Smart, Bonte, and Tan for the threshold-HBS infeasibility analyses that grounded the architectural decision to use per-validator standalone for public BFT.
- McGrew, Fluhrer, Gazdag, Kampanakis, Morton, and Westerbaan for the selected-element threshold-HBS pattern (THBS) Magnetar instantiates for the TEE-only custody path.
- Shamir [48] for the secret-sharing primitive used in the appendix threshold forms.
- The Pulsar (Lux LP-073) and Corona (Lux LP-104) authors for the companion lattice-family threshold constructions whose architectural pattern Magnetar's appendix forms inherit.

10 Conclusion

10.1 What Magnetar v1.0.0 is

Magnetar v1.0.0 is the production library implementation of two hash-based post-quantum signature primitives for the Lux Polaris cert profile:

- **Per-validator standalone primary primitive.** `PerValidatorKeypair`, `ValidatorSign`, `ValidatorBatchVerify`, `ValidatorAggregateCert`, `BuildAggregateCert`, `VerifyAggregateCert` at `ref/go/pkg/magnetar/standalone.go`. No DKG, no dealer, no aggregator-in-TCB. The architecturally correct primitive for public-BFT hash-based PQ consensus.
- **THBS-SE permissionless threshold companion.** `NewThbsSeKey`, `ThbsSeRound1`, `Combine`, `VerifyThbsSeEvidence`, `VerifyThbsSeShareEvidence` at `ref/go/pkg/magnetar/thbsse.go` (with internal share arithmetic in `thbsse_field.go`). Slot-bound commit-and-reveal with a PUBLIC COMBINER role: anyone can run Combine; no host is in the TCB by policy at sign time. Produces byte-identical FIPS 205 wire bytes.
- **FIPS 205 byte identity for both primitives.** Each σ_i from `ValidatorSign` and each σ from `Combine` is byte-identical to single-party FIPS 205 SignDeterministic, pinned empirically by TestMagnetar_Wire_FIPS205Verifiable and TestThbsSE_Wire_FIPS205Verifi across SHAKE-192s, SHAKE-192f, and SHAKE-256s.
- **Canonical MAGS / MAGG wire codec.** Stateless `VerifyBytes` / `VerifyBytesCtx` dispatchers accept wire bytes and route through unmodified `cloudflare/circl/sign/slhdsa`. Sibling of Pulsar's PULS/PULG and Corona's CORS/CORG framings.
- **Polaris third-lane integration.** Joins Pulsar (M-LWE) and Corona (R-LWE) as the hash-based leg of the maximum-assurance Polaris cert profile, providing cryptographic-family diversity against any single-family algorithmic break.
- **Reference implementation.** Pure Go, backed by `cloudflare/circl v1.6.3` [17] as the FIPS 205 primitive. Deterministic KAT vectors regenerable bit-identically from a clean checkout, including `thbsse-sign.json` at $(n = 7, t = 4)$ across three modes and three messages.

10.2 What Magnetar v1.0.0 is not

Honest characterisation of the open gates (tracked at `BLOCKERS.md`):

- **Strict invariant for THBS-SE (`MAGNETAR-STRICT-ATOM-V11`).** v1.0 ships a public combiner that reconstructs the SLH-DSA seed transiently in its own memory for the duration of one SignDeterministic call. The strict "no transient seed at any moment" refinement requires a Magnetar-internal re-implementation of FIPS 205 §§5–8 that bypasses `slh_sign_internal` and assembles the signature directly from per-atom share reconstructions of the message-selected FORS leaves and WOTS+ chain bases. The construction shape is forward-compatible; only Combine's internals change.

- **Leaderless PVSS-DKG setup (`MAGNETAR-PVSS-DKG-V11`).** v1.0 ships a deterministic-dealer setup for KAT determinism; production deployments needing the leaderless variant route through the sibling `luxfi/threshold` DKG package today, with a Magnetar-native `NewThbsSeKeyFromDealerlessDKG` constructor landing at v1.1.
- **Mechanised proofs (`MAGNETAR-PROOF-TRACK-V11`).** The legacy EasyCrypt scaffolding modeling the abandoned v0.x seed-recombine path has been removed; the v1.0 proof track ports to the THBS-SE construction shape (commit- bind, byte-wise Shamir over GF(257), Lagrange reconstruct, FIPS 205 dispatch) and lands at v1.1 alongside the Lean bridge for the algebraic content.
- **Constant-time empirical verification (`MAGNETAR-DUDECT-V11`).** v1.0's CT story is "inherit CIRCL"; the v1.1 `dudect` harness lands alongside the strict-atom-assembly path, at which point the per-atom WOTS+ chain compute and FORS sign step become the new CT-critical surfaces.
- **Independent audit (`MAGNETAR-EXTERNAL-AUDIT-V11`).** The v0.x internal sign-off applied to a construction surface much of which has been removed at v1.0. The v1.1 external audit should target the THBS-SE construction shape, the strict-atom-assembly path, and the leaderless PVSS-DKG setup.

10.3 Closing

The Lux PQ transition is structured around the principle that no single primitive carries the post-quantum future of the chain. Pulsar (M-LWE), Corona (R-LWE), and Magnetar (SLH-DSA) are three independent constructions that share neither hash function family, nor secret-sharing mechanism, nor underlying mathematical assumption beyond "the standard cryptographic-hash assumptions hold." The Polaris cert profile binds all three lanes against a common transcript and accepts only when all three verify.

Magnetar's role in this composition is the hash-based insurance leg: the lane that is intact when the entire lattice family is broken by some unforeseen algorithmic advance. It is the slowest of the three lanes, which is acceptable; the insurance does not need to be fast.

The per-validator standalone construction documented in §3 is the architecturally correct hash-based primitive for public-BFT consensus, given the load-bearing fact (Cozzo-Smart EUROCRYPT 2019 [19], Bonte-Smart-Tan 2023 [11]) that SLH-DSA admits no FROST-style linear share-aggregation identity. The THBS-SE construction documented in §5 is the permissionless-threshold closer for deployments that need a single FIPS 205-shaped signature from a t-of-n committee, sitting at the public-combiner point in the design space — materially stronger than TEE-attested privileged-aggregator constructions, materially weaker than the strict "no transient seed at any moment" refinement that the v1.1 strict-atom-assembly path will close.

The cryptographic primitives are not new. The composition that turns them into a public-BFT primary primitive plus a permissionless-threshold companion in the hash-based leg of a three-lane Polaris cert profile — with byte-identity to the unmodified FIPS 205 verifier and a canonical wire codec — is Magnetar's contribution.

References

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

[2] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, Vincent Laporte, Tiago Oliveira, and Pierre-Yves Strub. The last mile: High-assurance and high-speed cryptographic implementations. IEEE S&P, 2020.

[3] Anonymous. Two-round threshold signature from algebraic one-more learning with errors. In *EUROCRYPT*. Springer, 2024.

[4] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. EasyCrypt: A tutorial. In *Foundations of Security Analysis and Design VII*. Springer, 2013.

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

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

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The KECCAK reference. https://keccak.team, 2011.

[8] 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.

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

[10] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2023.

[11] Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. Thresholdizing hash-based signatures. Cryptology ePrint Archive, 2023. Infeasibility analysis for FIPS 205 SLH-DSA threshold without seed reconstruction or full hash-tree MPC.

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

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

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

[15] 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.

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

[17] Cloudflare. circl: Cloudflare interoperable reusable cryptographic library. https://github.com/cloudflare/circl, 2025. v1.6.3; `sign/slhdsa` package implements FIPS 205.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON: Fast-fourier lattice-based compact signatures over NTRU. https://falcon-sign.info, 2020.

[24] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 3rd edition, 2020.

[25] Zach Kelling. LP-073: Pulsar: Dynamic lattice threshold signatures for permissionless validator sets. Lux Industries; `papers/lp-073-pulsar/lp-073-pulsar.tex`, 2026.

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

[27] Zach Kelling and Vishnu Seesahai. LP-104: Corona: Lattice-based post-quantum threshold signatures for blockchain consensus. Lux Industries; `papers/lux-corona-pq/lux-corona-pq.tex`, 2026.

[28] John Kelsey, Shu jen Chang, and Ray Perlner. NIST SP 800-185: SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash. https://csrc.nist.gov/publications/detail/sp/800-185/final, 2016.

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

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

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

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

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

[34] Lux Industries. Lean 4 cryptographic proof tree (`lux-formal-verification-lean4`). https://github.com/luxfi/proofs/tree/main/lean/Crypto, 2026.

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

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

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

[38] David McGrew, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Michael Morton, and Bas Westerbaan. Coalition and threshold hash-based signatures. IACR ePrint 2019/793; https://eprint.iacr.org/2019/793, 2019. Companion IRTF draft-mcgrew-hash-sigs.

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

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

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

[42] NIST. NIST IR 8214C: Multi-party threshold cryptography schemes. NIST Internal Report (Draft), 2024.

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

[44] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

[45] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? Design, Automation & Test in Europe, 2017.

[46] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *CRYPTO*. Springer, 1999.

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

[48] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Reproducibility

Source code. `github.com/luxfi/magnetar` at tag `v1.0.0`.
Build. `cd ref/go && GOWORK=off go build ./... && GOWORK=off go test -short ./pkg/magnetar/..`
The race-detector (−race) is supported but skips the SLH-DSA heavy paths via the `raceEnabled` build-tag pattern; race-on runs cover the cheap unit tests (transcript, types, wire codec, THBS-SE share arithmetic).
Headline regressions. `go test -count=1 -run TestMagnetar_Wire_FIPS205Verifiable ./ref/go/pkg/m` pins byte-identity for the per-validator standalone primitive; `go test -count=1 -run TestThbsSE_Wire_FIPS2 ./ref/go/pkg/magnetar/...` pins the same identity for the THBS-SE public combiner. Both run across all three SHAKE parameter sets.
Hardware tested. Apple Silicon M1 Max, macOS 15.4, Go 1.26.3 darwin/arm64.
Dependencies. `github.com/cloudflare/circl v1.6.3` (FIPS 205 SLH-DSA reference, BSD-3-Clause) and `golang.org/x/crypto v0.32.0` (cSHAKE256 over FIPS 202 / NIST SP 800-185).

No private dependencies.

Standards referenced. FIPS 205 (SLH-DSA), FIPS 202 (SHA-3 / SHAKE / cSHAKE), NIST SP 800-185 (KMAC / TupleHash / cSHAKE customisation), NIST IR 8214C [42] (Multi-Party Threshold Cryptography scope).

Companion documents. `SPEC.md` (normative construction), `THBS-SPEC.md` (THBS-SE construction spec), `DEPLOYMENT-RUNBOOK.md` (operator runbook), `BLOCKERS.md` (v1.1 roadmap), `CHANGELOG.md` (release-by-release framing).

Contact. security@lux.network.