P3Q: Unified-Slot Post-Quantum Verifier Dispatch Framework

Zach Kelling

Lux Industries

2026-06-03

Abstract

P3Q is the unified-slot post-quantum verifier framework at EVM precompile slot `0x012205`: one slot, one ABI, and a leading `kind` byte that dispatches into per-kind threshold-signature verifier cores via the `P3QVerifier` Go interface. This paper specifies the framework itself — the 2026-06-03 architectural decomplection from sibling-slots to a single slot with kind-byte dispatch, the `P3QVerifier` registration interface, and the cross-family composition into LP-217 cert profile modes — independently of any one cryptographic family or any one caller. PQ-secured rollups (LP-218) are P3Q's primary consumer, but P3Q is its own surface: any contract — a bridge light-client, a cross-chain message verifier, an on-chain custody policy — calls the same slot for post-quantum threshold-signature verification. The per-kind verifiers are specified in companion LPs: **LP-223 Pulsar** (rank-k Module-LWE, FIPS-204 ML-DSA-65 threshold, kind $0x01$), **LP-220 Corona** (Module-LWE, two-round Corona threshold, kind $0x02$), and **LP-222 Magnetar** (hash-based, FIPS-205 SLH-DSA threshold, kind $0x03$). The framework's reason to exist is cross-family-maximum security: a single structural break in any one PQ family must not compromise a PQ-heavy rollup, because the kinds span two mutually independent families — lattice (Module-LWE, shared by Pulsar and Corona at distinct parameter sets) and hash collision resistance (Magnetar). Future kinds add a byte to the dispatch table, not a slot to the EVM.

Contents

1 Introduction

P3Q is a standalone on-chain post-quantum verifier framework: one EVM precompile at slot `0x012205`, one ABI, and a leading `kind` byte that dispatches into per-family threshold-signature verifier cores via the `P3QVerifier` interface. Any contract that needs to check a post-quantum threshold signature on-chain calls `P3Q.verify(kind, sig, messageHash, groupPubKey)` at a single address; the kind byte selects which cryptographic family answers. P3Q is not a rollup primitive — it is the general PQ-verify dispatch surface — but its driving consumer is the PQ-secured rollup, and this paper specifies the framework independently of any one caller or any one cryptographic family.

1.1 P3Q and its consumers

Post-quantum rollups separate cleanly into two layers, and P3Q is the lower one:

- **The rollup-batch envelope pattern** (LP-218, the primary consumer) — a sequencer composes transactions, threshold-signs a state-root tuple, and posts a single `RollupBatchTx`; the parent L1 verifies that signature by *calling* P3Q inside normal block validation. The rollup pattern owns the wire format, the parent-L1-verifies-the-sig pattern, and the inherited cert mode — it does not own the verifier.
- **The P3Q verifier framework** (this paper, LP-219) — the slot, the kind-byte dispatch ABI, the `P3QVerifier` registration interface, and the cross-family composition semantics. The per-kind verifiers are specified in the companion LPs LP-223 (Pulsar), LP-220 (Corona), and LP-222 (Magnetar).

LP-218 consumes P3Q; LP-219 specifies it. The dependency runs one direction only: a contract that is not a rollup at all — a bridge light-client, a cross-chain message verifier, an on-chain custody policy — can still call `P3Q.verify` for any post-quantum threshold-signature check.

1.2 Why a single slot

A naive design would allocate one EVM precompile slot per PQ family: `0x012205` for Pulsar, `0x012206` for Corona, `0x012207` for Magnetar. The earlier draft of this work did exactly that, treating the rollup-batch wrappers as sibling primitives. Three slots, three ABIs, three audit surfaces, three call-paths in EVM.

The 2026-06-03 decomplection collapses this into one slot with kind-byte dispatch. The motivations are not aesthetic:

- **Solidity callability is a UX surface.** A rollup contract that wants cross-family-maximum verify writes three `P3Q.verify(kind, ...)` calls at a single address. Three slots would mean three address constants, three ABI imports, three precompile registrations to track in the operator's head. One slot keeps the developer surface coherent.
- **Audit surface compounds.** Three slots with three wrappers means three independent code paths around the threshold-signature verifier core – the wrappers carry their own structural parsing, length checks, gas billing, and domain-separation contexts. Each is an audit boundary. One slot dispatches into per-kind verifier cores via the `P3QVerifier` interface and leaves only the dispatch table to audit.
- **Future kinds add a byte, not a slot.** If a fourth PQ family enters the Lux stack (a hypothetical code-based signature with HQC-style structure, say), it lands as one new `kind` byte in the dispatch table and one new `P3QVerifier` implementation. No new precompile slot allocation, no new Solidity function shape, no new audit boundary at the EVM-side.
- **One and only one way per LP-200.** The ZAP Stack umbrella codifies the principle: every distinct concern has exactly one canonical entry point. "Verify a rollup-batch threshold signature on-chain" is one concern; the kind byte selects which primitive answers the

question.

1.3 The kind set at a glance

kind	Name	PQ family	Underlying scheme	Spec
$0x01$	Pulsar	Module-LWE	FIPS-204 ML-DSA-65 threshold	LP-223
$0x02$	Corona	Module-LWE	Corona two-round threshold	LP-220
$0x03$	Magnetar	Hash-based	FIPS-205 SLH-DSA threshold	LP-222

Table 1: The three production kinds at slot `0x012205`, each specified in its own companion LP.

A simultaneous structural break across both families would require independent advances in (i) Module-LWE cryptanalysis — which both the Pulsar and Corona kinds rest on, at distinct parameter sets and codebases — and (ii) collision resistance of the SHA-2 / SHA-3 family (the Magnetar kind). These two families are widely understood to be mutually independent; Pulsar and Corona add parameter- and implementation-level diversity within the lattice family, not a second family. The cross-family-maximum posture LP-217 PQ-heavy codifies relies on the lattice/hash independence; this framework supplies the on-chain machinery to actually compose all three kinds inside a single rollup contract.

1.4 Honest measurement state

Throughout the P3Q LP family, performance numbers are tagged **measured** or **projected**. The architectural decisions of this framework paper (slot, ABI, dispatch table, registration interface) are not contingent on any GPU dispatch closing; they are contingent only on the per-kind CPU verifier correctness, specified and benchmarked in the companion kind LPs. GPU dispatch is projected, not measured, in every P3Q paper: `nvidia-smi dmon` mean `SM%` during the 2-hour 2026-06-04 Spark run was 11.9 % (idle/background only), and two boundary gaps (`lux-lattice.pc` not installed on Spark; `lux_slhdsa_*` entry points not yet exposed in `lux/accel/c_api.h`) prevent dispatch until closed.

1.5 Roadmap

Section 2 documents the 2026-06-03 decomplection in detail: the prior sibling-slots model, the unification onto `0x012205`, and the relationship to LP-0120's separate raw-primitive verifier slots. Section 3 defines the `P3QVerifier` Go interface that registers each kind into the dispatch table. Section 4 ties the kind set back into LP-217 cert profile modes and shows the rollup-contract composition pattern for the PQ-heavy posture. The per-kind verifier specifications live in LP-223 (Pulsar), LP-220 (Corona), and LP-222 (Magnetar).

2 Architectural Decision: Single Slot, Kind-Byte Dispatch

2.1 The prior sibling-slots model

The pre-decomplection draft of LP-220 (dated 2026-05-29) allocated three EVM precompile slots:

- `0x012205` – P3Q-Pulsar, wrapping the rollup-batch ABI around the Pulsar threshold-FIPS-204 verifier core.
- `0x012206` – P3Q-Corona, wrapping the rollup-batch ABI around the Corona (Corona) Module-LWE threshold verifier core.
- `0x012207` – P3Q-Magnetar, wrapping the rollup- batch ABI around the Magnetar threshold-SLH-DSA verifier core.

Each slot reused the corresponding LP-0120 raw-primitive verifier as the underlying implementation (`0x012204` raw Pulsar, `0x012206` raw Corona, `0x012207` raw Magnetar) and layered a rollup-batch-specific wrapper on top: structural parsing for the `(prev_root, new_root, batch_hash)` tuple, a separate FIPS context string per slot, separate gas calibration billed inside each wrapper.

The collision between LP-220's rollup-batch wrappers and LP-0120's raw-primitive verifiers at slots `0x012206` and `0x012207` was the first symptom that the design was overcomplecting two concerns:

1. "Verify a threshold signature of family F over a 32-byte digest" (the raw primitive concern, owned by LP-0120).
2. "Verify a rollup-batch envelope of family F where the digest is $H(\text{prev\_root} \,\|\, \text{new\_root} \,\|\, \text{batch\_hash})$ and the FIPS context binds the rollup pattern" (the rollup-batch concern, owned by LP-218 / LP-220).

These are two distinct concerns. They deserve two distinct surfaces. But they do not deserve six precompile slots between them.

2.2 The 2026-06-03 unification

The decomplection allocates the rollup-batch concern exactly one slot – `0x012205` – and reuses the existing raw-primitive slots (`0x012204`, `0x012206`, `0x012207`) for the raw-primitive concern. The rollup-batch slot dispatches on a leading `kind` byte to per-primitive verifier implementations:

Slot	Concern	Wire	Owner
`0x012202`	Raw ML-DSA single-party	FIPS-204	LP-0120
`0x012203`	Raw SLH-DSA single-party	FIPS-205	LP-0120
`0x012204`	Raw Pulsar (Module-LWE threshold)	FIPS-204 ctx + sig	LP-0120
`0x012205`	**P3Q family** (rollup-batch ABI)	**kind-byte dispatch**	**LP-218 + LP-220**
`0x012206`	Raw Corona (Module-LWE threshold)	Corona ctx + sig	LP-0120
`0x012207`	Raw Magnetar (SLH-DSA threshold)	FIPS-205 ctx + sig	LP-0120

Table 2: Precompile slot ownership after the 2026-06-03 decomplection. Slots `0x012204`, `0x012206`, `0x012207` remain owned by LP-0120 for the raw-primitive non-rollup-batch use case; LP-218 + LP-220 own `0x012205` as the P3Q rollup-batch family.

A Solidity contract that needs Corona-kind verify for a rollup-batch envelope calls `P3Q.verify(0x02, sig, h, pk)` at `0x012205`. A Solidity contract that needs raw Corona primitive verify – a non-rollup-batch use case, e.g. a custody attestation contract verifying an off-chain Corona-signed certificate – calls the LP-0120 raw-primitive verifier at `0x012206` directly. Two distinct callers, two distinct surfaces, two distinct slots, zero overlap.

The P3Q family ABI is the single canonical entry point for rollup-batch verification across all PQ kinds. This honors LP-200 § "One and only one way to do everything" [1].

2.3 Kind-byte dispatch

The unified `0x012205` precompile reads a 1-byte `kind` discriminator at offset 0 of calldata and routes to one of three (today) registered verifier implementations. The dispatch is at runtime, populated by package `init()` functions; the precompile entry-point is one function in one file. The full dispatch logic from the reference implementation at `lux/precompile/p3q/contract.go::Run`:

```
kind := input[0]
switch kind {
case KindPulsar:     // 0x01
    // fall through to FIPS 204 ML-DSA verify path
case KindCorona,     // 0x02
     KindMagnetar:   // 0x03
```

```
    // Reserved kinds -- verifier not yet wired. Return abiFalse
    // (the LP-218 "no revert on cryptographic failure" contract)
    // so callers still see a consistent surface; once the
    // lattice/hash threshold libraries land, this branch routes
    // to its own verifier.
    return abiFalse(), remaining, nil
default:
    // Unknown kind -- abiFalse, same as any malformed input.
    return abiFalse(), remaining, nil
}
```

Three properties hold by construction:

- **Constant-time over the kind byte.** The switch is on a PUBLIC field (the calldata kind byte is observable on-chain). Branching on it does not constitute a timing side channel against any secret.
- **No revert on cryptographic failure.** An unknown kind, an unwired reserved kind, a malformed sig, a FIPS-204 reject – all return `abiFalse` (a 32-byte zero-encoded EVM `bool`) with no Go error. The Solidity caller observes `false` via the return word and decides whether to revert. This matches LP-218 § "Failure modes."
- **Uniform gas billing.** The `RequiredGas` function is monotonic in input length and does NOT depend on the kind. A short malformed input pays the base gas, a well-formed N-byte input pays `base + N * PerByteGas`. No reduced price for early rejection – uniform billing prevents a gas-side channel from leaking which validation stage rejected the input.

Per-kind gas *rates* differ (Pulsar 50 000 gas, Corona 5 000 000 gas, Magnetar 30 000 gas); the resolution happens inside the per-kind `P3QVerifier.GasCost()` method described in section 3. The `0x012205` precompile multiplexes those rates over a single observable address.

2.4 What this is not

It is worth being explicit about two design decisions LP-220 does *not* make:

1. **It is not a generic PQ-signature precompile.** The three kinds are not interchangeable. A Pulsar signature submitted under `kind=0x02` (Corona) will fail structurally – Pulsar sigs are 3309 B, Corona sigs are $\sim$ 33 KB, and the verifier core for kind $0x02$ is the Corona Module-LWE verifier (which does not parse FIPS-204 output). The kind byte selects a specific verifier; it does not abstract over them.
2. **It is not a backwards-compatible upgrade of the sibling-slots draft.** There is no LP-220-v1 deployment in the wild; the activation marker `2025-12-25T16:20:00-08:00` places the unified-slot ABI at network genesis. Existing test corpora that target `0x012206` / `0x012207` for rollup-batch wrappers are obsolete (those slots are now LP-0120 raw-primitive surfaces with different wire format); they must be re-targeted to `0x012205` with the appropriate kind byte. No compatibility shim exists.

2.5 Activation marker

```
activates:      2025-12-25T16:20:00-08:00
activates-unix: 1766708400
```

The P3Q family ABI at slot `0x012205` with kinds $0x01$, $0x02$, $0x03$ registered in the dispatch table is callable from the genesis block of the new final Lux network. Rollup deployments that target PQ-strict or PQ-heavy mode call `P3Q.verify(kind, ...)` from height 0 with the appropriate kind byte (subject to per-kind verifier wiring, see LP-220–LP-222).

3 Verifier Interface and Dispatch

3.1 The P3QVerifier Go interface

The unification of the three kinds onto one slot is mediated by a Go-side interface at lux/standard/precompiles/
The interface is the single audit boundary for adding a kind: implement the four methods, register at init(), done.

```
// P3QVerifier is the per-kind verifier abstraction.
//
// Each P3QVerifier implementation is registered at process boot
// into a kind-byte keyed dispatch table loaded by the single
// P3Q precompile at slot 0x012205.
type P3QVerifier interface {
    // Verify returns true iff sig is a valid threshold signature
    // from groupPubKey on messageHash. The implementation must
    // be constant-time over secret inputs (there are none; both
    // sig and pk are PUBLIC calldata) and must not panic on
    // malformed input -- malformed inputs return false.
    Verify(sig []byte, messageHash [32]byte, groupPubKey []byte) bool

    // GasCost returns the per-call precompile gas charge for
    // this kind. The P3Q dispatcher charges this amount BEFORE
    // calling Verify, so an out-of-gas caller never observes the
    // verifier's behavior on its specific input.
    GasCost() uint64

    // Kind returns the calldata discriminator byte:
    //   0x01 = Pulsar     (Module-LWE)
    //   0x02 = Corona     (Module-LWE)
    //   0x03 = Magnetar   (hash-based)
    Kind() byte

    // PrimitiveName returns "pulsar", "corona", or "magnetar".
    // Used for metric labels, log lines, and audit trails.
    PrimitiveName() string
}
```

The interface is minimal by design: four methods, three of which are constant-returning getters. The verifier core lives in the Verify method; the gas calibration lives in GasCost; the dispatch discriminator lives in Kind. There is no shared base class, no inheritance, no mutable verifier state – each verifier is value-typed and independently complete (composition over inheritance per LP-200).

3.2 Dispatch flow

When the 0x012205 precompile is called:

1. Read the leading kind byte from calldata (offset 0).
2. Look up the registered P3QVerifier for that kind in the kind-keyed dispatch table.
3. Charge Verifier.GasCost() (per-kind, returned by the impl). The uniform per-byte adder is added separately by the precompile's RequiredGas.
4. Parse the wire format down to (sig, messageHash, groupPubKey) using the layout appropriate to the kind (the per-kind size constants are baked into the implementation; the framing – BE32 length fields, trailing 32-byte messageHash – is uniform).
5. Call Verifier.Verify(sig, messageHash, pk).
6. Return the boolean result encoded as a 32-byte EVM-ABI bool (abiTrue or abiFalse).

This is the single point of truth for the family. Three verifier implementations, one slot, one dispatch path, one ABI shape, one audit surface. Adding a fourth PQ-family kind (a hypothetical code-based threshold via HQC-Sig variants, for example) is one new P3QVerifier impl and one new kind byte registered in the same dispatch table; no new precompile slot, no new Solidity function shape, no new audit boundary at the EVM-side.

3.3 Per-kind gas table

kind	Name	Gas charge	Verify CPU
$0x01$	Pulsar	50 000	∼80 µs
$0x02$	Corona	5 000 000	∼1.6 ms
$0x03$	Magnetar	30 000	∼1.92 ms

Table 3: Per-kind gas table. Corona's 5 M charge is the ∼100× multiplier over Pulsar reflecting the CPU verify-cost ratio; Magnetar's 30 000 reflects the fact that the SLH-DSA hash-tree walk is memory-bandwidth-bound and parallelizes harder on modern CPUs than the lattice NTT inverse.

The asymmetry between Corona's 5 M and Magnetar's 30 000 – despite comparable CPU verify times – reflects two distinct calibrations:

- Corona's verify dominates a single core because the NTT inverse + polynomial multiplication mod q over the larger Ringtail-parameter coefficients is a tight cache-resident loop with limited ILP. 1.6 ms on a single core is ∼141× the Pulsar verify; the gas charge rounds to ∼100×.
- Magnetar's verify is a hash-tree walk: hash function evaluation is highly parallelizable, well-vectorized by modern CPUs, and memory-bandwidth-bound. 1.92 ms wall-clock is dominated by sequential dependencies in the WOTS+ structure but the per-operation cost is lower than the lattice multiplication. The gas charge reflects the modest per-call CPU cost.

This is the same calibration discipline LP-218 uses for Pulsar: gas reflects CPU-class verifier cost, not GPU-class. Operators with all-Blackwell validator sets may petition for discounted gas in a future amendment; LP-220 does not codify it.

3.4 Validator registration

Each P3Q-conformant validator MUST register a `P3QVerifier` for every kind it will accept on-chain. Misconfiguration – a validator missing the Corona verifier when Z-Chain receives a rollup-batch tx that calls `P3Q.verify(0x02, ...)` – is a configuration error, not a soundness error: the validator's EVM cannot validate the block, and consensus excludes that validator's vote on the block. LP-202 tier-degradation handles the recovery (the validator votes `nil` on the block; consensus continues with the remaining validators).

The reference `p3q.Run` ships KindPulsar wired and KindCorona / KindMagnetar reserved-but-not-wired; this is the genesis state. As the Corona and Magnetar verifier libraries land in subsequent versions of `luxfi/corona` and `luxfi/magnetar`, their P3Q dispatch entries are wired in, and the wired-state advances. The wire format and the Solidity ABI are stable across the wired/not-wired transition; only the verify behavior changes (from `false` to the actual cryptographic outcome).

3.5 Failure modes

4 Composition With Sibling LPs

4.1 LP-217 cert profile modes and the P3Q kinds

The LP-217 cert-profile-mode ladder is the operator-facing naming for cross-family PQ posture. Each mode declares a required leg set; the P3Q kinds supply the on-chain machinery that makes each mode's leg set callable from a rollup contract:

Failure	Trigger	Effect
Unknown `kind` byte	caller passes `kind` not in $\{0x01, 0x02, 0x03\}$	precompile returns `abiFalse`; caller observes `false`
Reserved-but-not-wired kind	caller passes `kind=0x02` or `kind=0x03` before verifier lands	precompile returns `abiFalse`; caller observes `false`
Malformed wire layout	caller passes truncated/oversized fields	precompile returns `abiFalse`
Sig-length mismatch (Pulsar)	`sigLen` $\neq$ FIPS-204 expected size for the mode byte	precompile returns `abiFalse`
FIPS verifier reject	sig genuinely does not verify under the group pubkey	precompile returns `abiFalse`
Out-of-gas	caller supplies `gas` $<$ `RequiredGas(input)`	precompile returns `contract.ErrOutOfGas`; caller transaction reverts
Group pubkey rotation skew	sequencer rotated keys without updating the on-chain group pubkey	precompile returns `abiFalse` until rotation tx lands

Table 4: Failure modes. Every cryptographic failure is `abiFalse`; only out-of-gas reverts. This is the LP-218 "no revert on cryptographic failure" contract – the Solidity caller decides whether `false` is fatal to its batch.

Mode	Required legs (P3Q calls)	Per-batch gas	Use case
`PQ-off`	BLS only (no P3Q)	$\sim$5000	non-financial; gaming
`PQ-fast`	BLS + P3Q($0x01$)	$\sim$55 000	payments, DEX, gaming
`PQ-strict`	BLS + P3Q($0x01$) + P3Q($0x02$)	$\sim$5 055 000	custody, treasury, securities
`PQ-heavy`	BLS + P3Q($0x01$) + P3Q($0x02$) + P3Q($0x03$)	$\sim$5 085 000	sovereign anchors, bridges

Table 5: LP-217 modes mapped to P3Q kind calls. The four cert profile modes correspond to subsets of the P3Q kind set; PQ-heavy calls all three. The $\sim$5 M-gas cost cliff between PQ-fast and PQ-strict is the price of intra-lattice parameter-diversity insurance via the Corona kind.

The composition is **additive**: a PQ-heavy rollup contract calls the BLS precompile and three `P3Q.verify(kind, ...)` calls at slot `0x012205` in sequence inside its `validateBatch` function. If any of the four returns `false`, the batch is rejected.

4.1.1 Sample rollup-contract composition (Solidity)

```
function validateBatch(
    bytes32 prevRoot,
    bytes32 newRoot,
    bytes32 batchHash,
    bytes calldata blsSig,
    bytes calldata pulsarSig,
    bytes calldata coronaSig,    // omit for PQ-strict
    bytes calldata magnetarSig,  // omit for PQ-strict/fast
    bytes calldata groupPk
) external returns (bool) {
    bytes32 h = sha256(abi.encodePacked(prevRoot, newRoot, batchHash));

    // Classical leg
    if (!BLS.verify(blsSig, h, blsGroupPk)) return false;
```

```
    // P3Q Pulsar leg (PQ-fast and above)
    if (!P3Q.verify(0x01, pulsarSig, h, groupPk)) return false;

    if (mode >= PQ_STRICT) {
        // P3Q Corona leg (Module-LWE family independence)
        if (!P3Q.verify(0x02, coronaSig, h, groupPk)) return false;
    }

    if (mode == PQ_HEAVY) {
        // P3Q Magnetar leg (hash-based family independence)
        if (!P3Q.verify(0x03, magnetarSig, h, groupPk)) return false;
    }

    return true;
}
```

The same precompile address (`0x012205`) is called three times with three different kind bytes. The rollup contract's mode field is set at rollup-genesis time via `CertModeFloor` (LP-218); the contract enforces it on every batch.

4.2 LP-218 rollup-batch pattern

LP-218 specifies the rollup-batch envelope and the sequencer→Z-Chain submission pattern; LP-220 specifies the per-kind verifier mechanics. The two LPs share the same slot (`0x012205`), the same ABI shape (`P3Q.verify(kind, sig, h, pk)`), and the same `abiFalse`-on-cryptographic-failure contract.

LP-218 § "Precompile wire format" documents the structural format; LP-220 sections the per-kind LPs LP-218, LP-220, and LP-222 document the per-kind specifics on top of that uniform skeleton. The intent is clean cross-reference rather than duplication: a reader who wants the rollup-batch pattern reads LP-218; a reader who wants per-kind verifier mechanics reads LP-220; a reader who wants both reads both.

4.3 LP-300 schema registry

The P3Q family lives at one slot, but the wire envelopes – the sequencer→Z-Chain `RollupBatchTx` payload, the inner-batch Pulsar/Corona/Magnetar signature blobs – are schema-versioned per LP-300. The registry entry for the P3Q family is:

```
slot:    0x012205
kinds:   { 0x01: pulsar, 0x02: corona, 0x03: magnetar }
domain:  lux-evm-precompile-p3q-v1   (FIPS 204 ctx)
abi:     P3Q.verify(uint8, bytes, bytes32, bytes) -> bool
```

Schema bumps within a kind (e.g. a future Corona parameter set revision) advance the mode byte without changing the slot or the kind byte. Schema bumps across the family (a fourth kind, say) advance the kind byte without changing the slot.

4.4 Cross-references

4.5 Backwards compatibility

None. The pre-Quasar Edition Lux network had no P3Q precompile family and is a separate network out of scope. The genesis state of the new final Lux network (activation `2025-12-25T16:20:00-08:00`) ships the unified-slot kind-byte-dispatch ABI as the only ABI; no shim exists for the sibling-slots draft.

LP	Relationship
LP-073	Pulsar threshold signature – P3Q kind $0x01$ verifies
LP-181	Magnetar threshold SLH-DSA – P3Q kind $0x03$ verifies
LP-182	QuasarCert – the cert legs whose verify kinds LP-220 supplies
LP-200	ZAP Stack umbrella – the "one and only one way" principle the unified slot honors
LP-202	Cert tier degradation – the per-kind verify path participates in the tier ladder
LP-203	GPU-native verify – batched Corona / Magnetar kernels (projected)
LP-217	Cert profile modes – the operator-facing names that determine which P3Q kinds a rollup contract calls
LP-218	ZAP-native PQ rollups – specifies the unified slot and kind $0x01$ Pulsar; LP-220 specifies kinds $0x02$ Corona and $0x03$ Magnetar on the same slot
LP-219	Rollup-to-rollup atomic – cross-rollup batches verify via the P3Q family at each participating rollup's chosen kind set
LP-0120	Quasar mainnet defaults – owns slots `0x012204`, `0x012206`, `0x012207` for raw-primitive verifiers (the non-rollup-batch use case); LP-218/LP-220 P3Q family lives exclusively at `0x012205`
LP-300	Schema registry – entry for the P3Q family kind set
FIPS 204	ML-DSA standard – Pulsar-kind verifier conformant
FIPS 205	SLH-DSA standard – Magnetar-kind verifier conformant
Ringtail eprint 2024/1113	2-round lattice threshold reference – Corona-kind verifier conformant

Table 6: LP-220 composition references.

4.6 Future work

- **Corona/Magnetar verifier wiring.** The reference `p3q.Run` returns `abiFalse` for kinds $0x02$ and $0x03$ today. The follow-up is to wire `luxfi/corona/sign.Verify` and `luxfi/magnetar/verify` into the dispatch table. The ABI and the wire format are stable across this transition; only verify behavior changes from `false` to the cryptographic outcome.
- **GPU dispatch boundary closure.** Install `lux-lattice.pc` on Spark and expose `lux_slhdsa_*` / `lux_corona_*` entry points in `lux/accel/c_api.h`. Once both close, the Blackwell verify projections (LP-220 and LP-222) become measurements.
- **Fourth PQ family.** If a code-based signature scheme (HQC-Sig variants, McEliece-Sig) becomes operationally relevant, it lands as one new `P3QVerifier` impl and one new `kind` byte (provisionally $0x04$) in the dispatch table. No new precompile slot allocation, no new Solidity function shape.
- **Blackwell-discounted Corona gas.** Operators with all-Blackwell validator sets may petition for a discounted Corona gas charge reflecting GPU-class verify cost rather than CPU-class; LP-220 does not codify this. A future amendment would tie the discount to a validator-hardware attestation surface.

References

[1] Lux Industries. Lp-200: The lux zap stack, 2026.