Recursive Lattice zk-SNARKs for Instant Trustless Bootstrap: A 10-Millisecond Verification Path for Post-Quantum Blockchain Nodes

(v2 — with zk-STARK–attested transparent setup)

Quillon Graph Research
research@quillon.xyz

May 13, 2026

Abstract

We describe the design and phased implementation of a recursive zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for the Quillon Graph chain. The construction collapses the validity of the entire chain tip into a single constant-size proof verifiable in approximately five to ten milliseconds on commodity hardware. A fresh node fetches a 5–150 KB proof from any peer, verifies it locally, and immediately accepts the canonical state root — enabling mining and transactions within seconds rather than hours, without trusting a checkpoint signed by an operator. The state commitment is a depth-256 sparse Merkle tree over the wallet–balance map; the per-block transition is encoded as an R1CS circuit composing BLAKE3, ML-DSA (Dilithium5) signature verification, a Merkle-path gadget, and a number-theoretic transform (NTT)-based anchor election check; the recursion is realized via Nova folding in the first deployment and is scoped to migrate to a lattice IVC scheme (LatticeFold, LaBRADOR, or Greyhound) in 2028 so the entire trust chain becomes post-quantum. The Nova structured reference string is generated by a deterministic public algorithm and accompanied by a zero-knowledge STARK proof of correct generation, removing the need for a multi-party trusted-setup ceremony and aligning with the project's existing on-chain STARK infrastructure (4,469 LOC across `crates/q-zk-stark/`, plus a 173 KB genesis-time encryption-SRS STARK proof already on the live chain). We report the implementation status honestly: the sparse Merkle tree is shipped in shadow mode; the gadget library (BLAKE3, Poseidon, NTT, Dilithium5) totalling roughly 2,800 LOC is production-grade; the STARK infrastructure (AIR framework, prover, verifier, batch prover, polynomial/FRI commitments, blockchain-state circuit, wallet-privacy circuit) is approximately 70–80% shipped; the δ-circuit composition, Nova wrapper, lattice swap, and wire protocol are scoped for Phases 1–4 with public mandatory verification targeted for 2028.

Contents

1 Introduction **1**

2 Background and Preliminaries **3**
2.1 The Quillon Graph chain 3
2.2 Incrementally verifiable computation 3
2.3 Lattice succinct arguments 3
2.4 Why we will deploy Nova first and migrate later 4

3 System Architecture **4**
3.1 State commitment: `balance_root_v2` (sparse Merkle tree) 4
3.2 The δ-circuit: one R1CS proof per block transition 5
3.3 Recursion: Nova folding 6

4 Transparent Setup via zk-STARK Attestation **6**
4.1 The trusted-setup problem 6
4.2 The STARK-attested alternative 7
4.3 Existing infrastructure 7
4.4 What this changes operationally 8
4.5 What this does *not* change 8

5 Performance Analysis **8**
5.1 Constraint count breakdown 8
5.2 Prover throughput target 8
5.3 Verifier latency 9
5.4 Proof size 9

6 The Lattice Migration **9**
6.1 Why migrate 9
6.2 Setup options across the Phase 2–4 trajectory 9
6.3 What stays the same across the swap 10
6.4 Decision pathway 10

7 Beyond Setup: zk-STARK Throughout the Stack **11**
7.1 Wallet privacy proofs — shipped today 11
7.2 Batched Dilithium signature verification — near-term work 11
7.3 Light-client mobile path — near-term work 11
7.4 Cross-chain bridge attestations — future work 12
7.5 Activation-height attestations — future work 12
7.6 Encryption SRS proof — shipped today, foundational reference 12

8 Bootstrap Protocol and Progressive Archival **12**
8.1 Wire protocol 12
8.2 Bootstrap-mode CLI flag 13
8.3 Progressive archival UX 13

9 Security Analysis **14**
9.1 Soundness 14
9.2 Setup trust assumptions 14
9.3 Phased deployment with advisory window 15
9.4 Lattice parameter selection 15

10 Comparison to Related Systems **15**

11 Phased Rollout **16**

12 Implementation Status **16**

13 Conclusion **18**

---

1 Introduction

Joining a blockchain network for the first time is one of the worst experiences in decentralized infrastructure. A new Quillon Graph node today must either **(a)** replay the entire chain from genesis, which takes between five and twelve hours depending on disk throughput and peer bandwidth, or **(b)** accept a checkpoint snapshot signed by a bootstrap operator and trust that the signing key has not been compromised. Option (a) is genuinely trustless but glacially slow. Option (b) is fast but reintroduces a centralized trust assumption that the rest of the protocol works hard to eliminate.

We argue that neither tradeoff is necessary. Recent advances in incrementally verifiable computation (IVC) [1, 2], paired with a state commitment carefully chosen to be both lattice-friendly and hash-only (no elliptic-curve primitives buried in the consensus path), make it possible to compress the validity of an entire chain into a single constant-size proof that any node — including a browser wallet — can verify in milliseconds.

The contribution of this whitepaper is the concrete design of such a proof for Quillon Graph and a phased plan to deploy it without disrupting a live multi-billion-dollar mainnet:

(1) A **depth-256 sparse Merkle tree** (SMT) over the wallet–balance map, using BLAKE3 leaf and node hashes with explicit domain separation. The SMT is implemented today in `crates/q-storage/src/balance_smt.rs` (∼600 LOC, twelve test cases, shipped in shadow mode).

(2) A **single-block state-transition circuit** (δ-circuit) that composes the existing R1CS gadget library: BLAKE3 compression, ML-DSA (Dilithium5) signature verification, Poseidon, an NTT verifier, and a Merkle-path gadget over the SMT. Scoped for Phase 1; not yet implemented.

(3) A **Nova fold** of the δ-circuit producing a constant-size, constant-verification-cost proof π_n of the validity of the chain from genesis to height n. Scoped for Phase 2.

(4) A **lattice migration path** (Phase 4, target Q1 2028) that replaces Nova on BN254 with LatticeFold [5], LaBRADOR [4], or Greyhound [6], leaving the δ-circuit unchanged and removing the trusted setup. The decision among the three candidates is deferred until Phase 2–3 benchmark data is available.

(5) A **progressive-archival UX** (`GET /api/v1/status/archive` and `X-QNK-Archive-*` response headers) that ships independently of the SNARK work, so the user-facing promise of an instantly useful node is honest about which queries are answered locally and which are still being backfilled.

We are explicit throughout this document about what is implemented today and what is forthcoming. The SMT is in production binaries in shadow mode (validating parity with the legacy flat-hash `balance_root_v1` on every block). The 2,792 LOC of gadgets in `crates/q-ivc/` cover BLAKE3 verification, Poseidon, an NTT butterfly with the FIPS 204 negacyclic convention, and the full Dilithium5 verification primitive (`compute_az_minus_ct`, `high_bits`, `use_hint`, and the signed-norm helper). The δ-circuit composition, the Nova wrapper, the lattice swap, and the wire protocol are aspirational; none are released yet. Promising what does not exist is how cryptographic protocols quietly become unsafe in production.

The remainder of this paper is organized as follows. Section 2 reviews the necessary background: IVC, lattice succinct arguments, and the specifics of the Quillon Graph chain. Section 3 presents the SMT and the δ-circuit. Section 5 analyzes constraint counts, prover throughput, verifier latency, and proof size. Section 6 discusses the lattice migration. Section 8 specifies the wire protocol and the progressive archival UX. Sections 9 and 10 cover security analysis and a

comparison to related systems. Section 11 sets out the phased rollout. Section 12 states the present implementation status precisely.

2 Background and Preliminaries

2.1 The Quillon Graph chain

Quillon Graph is a UTXO-style account chain whose consensus layer combines a Narwhal-style DAG mempool [23] with the DAG-Knight anchor election [22]. Blocks are produced at approximately one block per second on mainnet. The consensus layer chooses an *anchor block* per round using a verifiable number-theoretic transform (NTT)-based randomness beacon. Transaction signatures use the ML-DSA scheme standardized as FIPS 204 [8] at parameter level 5 ("Dilithium5"). Block headers are serialized in postcard binary format and hashed with BLAKE3 [7]. The state commitment in the current header is `balance_root_v1`, a flat BLAKE3 hash over the sorted `(address, balance)` pairs:

```
let mut root_hasher = blake3::Hasher::new();
root_hasher.update(b"balance_root_v1");
for leaf in &leaf_hashes { // O(N) in N wallets
    root_hasher.update(leaf);
}
*root_hasher.finalize().as_bytes()
```

Listing 1: Legacy state commitment in `crates/q-storage/src/lib.rs`.

This commitment is sound and cheap to maintain off-circuit, but proving membership inside an R1CS circuit costs $\Theta(N)$ BLAKE3 compressions — infeasible at $N \approx 10^5$ wallets today and worse as the network grows.

2.2 Incrementally verifiable computation

Incrementally verifiable computation (IVC), introduced by Valiant in 2008 [1], lets a prover maintain a single proof π_n that $F^n(z_0) = z_n$ for some step function F, updating it incrementally as F is applied. The verifier checks π_n in time independent of n. The original construction was theoretical; folding-scheme approaches such as Nova [2] and its refinement HyperNova [3] have produced the first implementations efficient enough for production-scale recursion.

Nova works over a pair of elliptic curves (typically BN254 and a cycle partner). The folding step does not produce a SNARK at each step — it produces a *relaxed* R1CS instance. A final SNARK is generated only at the moment verification is required, by "decompressing" the relaxed instance through a Spartan-style proof. This is what gives Nova its key property: the per-step prover cost is linear in the step circuit, not in n, and the final verify cost is independent of n.

For blockchain bootstrap, this is exactly the property we want. The step function is "apply one block to the state root." A 12-month-old chain with $\approx$ 31,500,000 steps verifies in the same time as a one-hour-old chain.

2.3 Lattice succinct arguments

Pairing-based SNARKs — Groth16 [13], Marlin [12], the Pickles construction in Mina [10], Halo 2 [11] — rely on the hardness of the discrete log or pairing problems on elliptic curves. These assumptions fall to a sufficiently large quantum computer running Shor's algorithm [20]. This is a problem for a chain whose transaction signatures (Dilithium5) and key-encapsulation primitives (Kyber, FIPS 203 [9]) are already post-quantum: if the recursive proof itself rests on

a quantum-broken primitive, then the most critical piece of the trust chain — the proof that the entire history is valid — has the worst quantum exposure in the stack.

Lattice-based succinct arguments avoid this. LaBRADOR [4] produces compact proofs for R1CS instances based on the Module Short Integer Solution (Module-SIS) problem. LatticeFold [5] adapts the folding-scheme idea to lattices, providing IVC with Module-SIS hardness. Greyhound [6] provides fast lattice-based polynomial commitments and could be wrapped in a recursion harness.

All three are post-NIST PQC standardization era [21] and parameter-selectable to at least the NIST PQC level-1 security target. The trade-off is implementation maturity: pairing-based recursive SNARKs have production deployments going back years (Mina, Aleo); lattice IVC deployments do not yet exist outside research code as of mid-2026.

2.4 Why we will deploy Nova first and migrate later

We choose to ship a pairing-based recursive SNARK (Nova on BN254) in Phases 1–3, accepting a transient non-post-quantum dependency in the recursive layer, and then migrate to lattice IVC in Phase 4. The reasoning is operational rather than cryptographic:

- The δ-circuit is the same in both flavors. The proof system swap is a localized change of about 800 LOC in `crates/q-ivc/src/recursion/`; everything else — the SMT, the gadgets, the API surface, the bootstrap flow — stays put.
- Debugging a 440-million-constraint composition circuit *and* a brand-new lattice IVC implementation simultaneously is a recipe for soundness errata. Production-mature Nova lets us validate the circuit first.
- The recursive proof during Phases 1–3 is advisory — blocks are still validated block-by-block on every node. A quantum attacker forging a Nova proof during this window would not compromise the chain; the proof simply would not be accepted when re-verified against block-level rules.
- By the Phase 4 target (Q1 2028), at least one of the three lattice candidates will have a vetted reference implementation suitable for production. We pick based on benchmark data at that time, not now.

3 System Architecture

3.1 State commitment: `balance_root_v2` (sparse Merkle tree)

The first structural change is the migration from the flat `balance_root_v1` commitment to a sparse Merkle tree (`balance_root_v2`). The SMT is keyed by the 32-byte BLAKE3 of the public key (the wallet address); leaves carry the 16-byte little-endian `u128` balance. The tree has depth 256, matching the address length. Internal nodes use BLAKE3 with explicit domain separation:

$$\begin{aligned} \mathrm{leaf}(a, b) &= \mathrm{BLAKE3}(\text{"}\texttt{smt_leaf_v2}\text{"} \,\|\, a \,\|\, b_{\mathrm{LE}}) \\ \mathrm{node}(L, R) &= \mathrm{BLAKE3}(\text{"}\texttt{smt_node_v2}\text{"} \,\|\, L \,\|\, R) \end{aligned}$$

The empty-subtree hashes $E_d = \mathrm{node}(E_{d+1}, E_{d+1})$ are precomputed for all depths $d = 0, \ldots, 256$, anchored by $E_{256} = \mathrm{leaf}(\mathbf{0}, 0)$. When a sibling along a path equals E_d the storage layer omits it from the database; on the wire it is signaled by a single bit in a 256-bit empty-subtree bitmap. This is the standard "Compact Sparse Merkle Tree" construction in proof-friendly form.

Implementation status: `crates/q-storage/src/balance_smt.rs` is approximately 743 LOC and is shipped in shadow mode in v10.9.x: every block updates both `balance_root_v1` and the SMT in the same RocksDB write batch and asserts identity between the two roots (so any divergence is immediately visible in mainnet logs). The mandatory switch from v1 to v2 in the block header is gated on a future activation height `BALANCE_ROOT_V2_HEIGHT`, which is not yet set.

```
pub struct BalanceSmt { /* ... */ }

impl BalanceSmt {
    pub fn new(db: Arc<DB>) -> Result<Self>;
    pub async fn rebuild_from_balances(
        &self, balances: &HashMap<[u8;32], u128>
    ) -> Result<[u8;32]>;
    pub async fn update(
        &self, addr: &[u8;32], balance: u128
    ) -> Result<[u8;32]>;
    pub async fn batch_update(
        &self, updates: &[([u8;32], u128)]
    ) -> Result<[u8;32]>;
    pub async fn prove(&self, addr: &[u8;32]) -> Result<SmtProof>;
    pub fn root(&self) -> [u8;32];
}

pub struct SmtProof {
    pub addr: [u8;32],
    pub balance: u128,
    pub siblings: [[u8;32]; 256],
    pub empty_bitmap: [u8; 32], // bit i = 1 iff siblings[i] == E_i
}

impl SmtProof {
    pub fn verify(&self, expected_root: &[u8;32]) -> bool;
}
```

Listing 2: Public API of the shipped sparse Merkle tree, `crates/q-storage/src/balance_smt.rs`.

3.2 The δ-circuit: one R1CS proof per block transition

The δ-circuit is the heart of the construction. It takes as *public inputs* the previous and next state roots, the block header hash, and the block height. It takes as *private witnesses* the entire block body, the Merkle paths into both state roots for every touched wallet, the Dilithium5 signature verification witnesses, and the NTT anchor witness. The circuit is satisfied if and only if the block validly transitions `state_root_prev` $\rightarrow$ `state_root_next`.

Definition 1 (δ-circuit)**.** *Let s_n denote the SMT root at height n and B_{n+1} the block produced at height $n+1$. The δ-circuit is an R1CS predicate*

$$\delta(s_n,\, B_{n+1},\, s_{n+1}) \in \{0, 1\}$$

satisfied if and only if B_{n+1} is a valid block under all consensus rules and applying its transactions, coinbase, and emission to the wallet–balance map committed by s_n yields the map committed by s_{n+1}.

The predicate composes the existing gadgets:

1. $h_{\text{hdr}} = \text{BLAKE3}(\text{header_bytes})$, enforced via `Blake3Gadget::verify_hash`.

Algorithm 1 Proof-bootstrap of a fresh node

Require: Peer URL p, expected genesis state root s_0

1: $(s_t, \pi_t, H_t) \leftarrow$ HTTP_GET(p, `/api/v1/proof/tip`)
2: valid $\leftarrow$ VerifyRecursive(π_t, s_t, H_t.height)
3: **if** $\neg$valid **then**
4: **retry** with a different peer
5: **end if**
6: **assert** H_t.state_root $= s_t$
7: Accept s_t as canonical; enable mining and transaction acceptance
8: Begin archive backfill in background (not on critical path)

2. For each transaction tx $= (\text{from}, \text{to}, a, f, n, \sigma, \text{pk})$:

   2.(a) $\text{Verify}_{\text{Dilithium5}}(\text{pk}, \text{msg}(\text{tx}), \sigma)$, via `DilithiumVerifierGadget::verify`.

   2.(b) Membership of $(\text{from}, b_{\text{from}})$ in s_n, via `MerklePathGadget::enforce_membership`.

   2.(c) Membership of $(\text{to}, b_{\text{to}})$ in s_n, same gadget.

   2.(d) Balance sufficiency $b_{\text{from}} \geq a+f$ (range check on $b_{\text{from}}-a-f$ via `enforce_norm_bound`).

   2.(e) Updated $(\text{from}, b_{\text{from}} - a - f)$ membership in an intermediate root.

   2.(f) Updated $(\text{to}, b_{\text{to}} + a)$ membership in the root that follows the from-side update.

3. Coinbase emission satisfies the era-step schedule $e \leq R(\text{height})$, where R is implemented as a piecewise lookup over the four-year halving boundaries.

4. NTT-based anchor election validates the block producer's claim of being the anchor for this round, via `NttVerifierGadget`.

5. After all transactions and the coinbase have been applied sequentially, the resulting current root must equal the public `state_root_next`.

Remark 1. *The δ-circuit is large but not enormous. We accept a high per-block prover cost (Section 5) because the prover lives on dedicated infrastructure and runs once per block; the verifier runs on every wallet and every new node, and we want it fast.*

3.3 Recursion: Nova folding

With δ defined as a step circuit, the recursive proof is constructed by Nova's fold:

$$\pi_{n+1} = \text{Nova.Fold}(\delta,\ s_n,\ B_{n+1},\ \pi_n), \qquad \pi_0 = \mathbf{1}.$$

Each fold consumes the previous proof and the new block's witness and produces a fresh proof of size and verification cost independent of n. Genesis nodes (Epsilon, Beta, Gamma, Delta) maintain (s_n, π_n) alongside the block database and serve $(s_n, \pi_n, \text{header}_n)$ over the wire protocol described in Section 8.

A new node executes the algorithm in Algorithm 1.

Steps 1–7 complete in well under one second; the verify call (line 2) is the binding constraint at approximately 5–10 ms.

4 Transparent Setup via zk-STARK Attestation

4.1 The trusted-setup problem

Nova and Groth16 both consume a structured reference string (SRS) at proving and verification time. The SRS is generated by sampling a secret trapdoor τ and emitting public group elements derived from τ. Anyone who knows τ can forge proofs. The standard mitigation is a multi-party computation (MPC) ceremony: N participants each contribute randomness, mix it into the SRS, and discard their share. Soundness requires that at least one participant genuinely discarded their share. The Powers of Tau ceremonies behind [13] and most Zcash deployments adopted this protocol with hundreds of participants. The ceremony is correct in principle but operationally unattractive: it is slow to coordinate, must be repeated for every new circuit, and requires real-world trust in the participant pool.

4.2 The STARK-attested alternative

We avoid the ceremony entirely. The SRS is generated by a deterministic public algorithm $\mathsf{gen} : \mathsf{seed} \mapsto \mathsf{SRS}$, where seed is a public unpredictable quantity (in our case, a future block hash determined after the SRS-generation transaction is mined — a RANDAO-style construction). A zero-knowledge STARK proof π_{setup} attests that SRS was produced by running gen on seed. Any validator can re-run gen locally and verify π_{setup}; if both check, the SRS is accepted.

The trust assumption collapses to:

1. **STARK soundness** (random-oracle model on BLAKE3 [7, 15]), and
2. **Seed unpredictability** (the seed is committed before any party who could influence it learns its value).

Both assumptions are standard, well-studied, and require zero coordination between human participants.

```
public seed s > gen(s)  > SRS

                > STARK_Prove(gen, s, SRS) > _setup

validator: re-run gen(s), check SRS matches,
           verify _setup with public verifier  trust SRS
```

4.3 Existing infrastructure

We are not adding zk-STARK to the project for this work. The pattern is already in production:

- `crates/q-zk-stark/` (4,469 lines of Rust) ships: `air.rs` (Algebraic Intermediate Representation, 429 LOC), `stark_prover.rs` (442 LOC), `stark_verifier.rs` (548 LOC), `batch_prover.rs` (662 LOC, batched STARK proofs over multiple statements), `polynomials.rs` (549 LOC, FRI-style polynomial commitments), `blockchain_state_circuit.rs` (410 LOC, a STARK circuit over blockchain state transitions), `wallet_privacy_stark.rs` (530 LOC, deployed wallet-privacy proofs), and a `gpu/` subtree for hardware acceleration.
- `crates/q-storage/src/encryption_zkstark.rs` (562 LOC) exposes `EncryptionKeyProof` — a STARK proof binding the encryption key schedule to a public seed. The proof is generated once per chain at genesis, validated by every participating node.
- The artifact `data-mainnet-genesis/encryption.zkstark` (176,832 bytes) is a production STARK proof currently committed to the chain. The transparent-setup pattern has been

live since `v1.0.43-beta`.

The Nova-SRS attestation reuses the same AIR framework and the same prover/verifier pair. The circuit that the STARK proves is the Nova-SRS generator `gen` itself: a deterministic sequence of field-multiplications and group exponentiations. No new cryptographic constructions are required — only a new AIR description of `gen` and a one-time prover invocation.

4.4 What this changes operationally

The Phase 2 deployment (Section **??**) ships the Nova-attested SRS from day one. There is no separate MPC-ceremony phase. The on-chain artifact `data-mainnet-genesis/nova-srs.zkstark` (analogous to the existing `encryption.zkstark`) is generated by the Quillon Graph maintainers, verified by every validator and every fresh-bootstrap node, and remains auditable indefinitely. If a flaw is later found in the chosen lattice scheme or in Nova itself, the STARK-attested SRS for the next generation can be re-generated and re-attested without re-running a coordinated MPC ceremony.

4.5 What this does *not* change

The STARK attestation removes the ceremony but does not make the underlying proof system post-quantum. Nova on BN254 still depends on the elliptic-curve discrete logarithm assumption, which Shor's algorithm [20] can break given a sufficiently capable quantum computer. Phase 4 (Section **??**) addresses post-quantum by swapping the underlying scheme; the STARK-attested setup pattern carries over unchanged to any lattice scheme that requires public parameters.

5 Performance Analysis

5.1 Constraint count breakdown

The cost of a single δ-circuit step dominates the prover budget. Table 1 breaks down the constraint count for a block carrying $K = 100$ transactions and one coinbase, in line with current mainnet block content.

Component	**Per-instance**	**Block total ($K = 100$)**
Block-header BLAKE3 hash	50,000	50,000
Per-tx Dilithium5 signature verify	1,500,000	150,000,000
Per-tx Merkle paths (4 × depth-256, BLAKE3)	2,360,000	236,000,000
Per-tx balance arithmetic + range	10,000	1,000,000
Coinbase Merkle path + emission lookup	5,000,000	5,000,000
NTT anchor verification	50,000,000	50,000,000
Total		**≈ 442,050,000**

Table 1: R1CS constraint count breakdown for one δ-circuit step, $K = 100$ transactions. The dominant terms are Dilithium5 signature verification and the four Merkle paths per transaction. Numbers are upper-bound estimates derived from the existing `Blake3Gadget` per-compression cost (approximately 2,300 constraints) and from published Dilithium R1CS analyses.

Two notes on the table. First, the four Merkle paths per transaction (from_prev, to_prev, from_next, to_next) at 256 × BLAKE3-compression each costs about 590,000 constraints per path. Second, the BLAKE3 cost dominates the Merkle gadget; replacing it with a SNARK-friendly hash such as Poseidon [14] would compress this column by roughly two orders of magnitude but would also make the SMT root no longer reproducible by a BLAKE3-only light

client. We have chosen to stay BLAKE3 for v2 and to revisit a Poseidon-rooted "v3" state commitment in 2027 if Phase 2–3 benchmarks demand it.

5.2 Prover throughput target

To keep up with mainnet's one-block-per-second cadence, the fold step must complete within one wall-clock second per block on the genesis prover hardware. Nova's per-step prover cost scales roughly linearly with the step circuit's constraint count; a 442 M constraint step on a 48-core Epsilon-class machine is on the edge of feasibility. We have three optimization levers in reserve:

1. **Batched signature verification.** Verify b signatures in a batched gadget rather than b independent gadget invocations, reusing the same Number-Theoretic Transform tables. Saves on the order of 40% per signature for $b = 4$.
2. **Multi-block folds.** Fold m blocks into one Nova step (still valid; Nova does not require step granularity to match block granularity). This amortizes setup overhead.
3. **Distributed proving.** Phase 3 onwards, distribute the per-block fold across multiple validator nodes so no single machine has to keep up with mainnet alone.

If none of these are sufficient, we fall back to producing π every m blocks rather than every block, with $m \leq 10$. A bootstrap proof that lags the tip by ten seconds is still acceptable user experience.

5.3 Verifier latency

Verifier cost is the metric that ultimately matters for the user-facing promise. Table 2 summarizes the targets across deployment surfaces.

Surface	Target latency	Notes
Desktop (Intel/AMD x86-64, 8 cores)	≤ 5 ms	Native arkworks verifier
Laptop (M-series ARM)	≤ 10 ms	Native arkworks verifier
Browser WASM (mobile or desktop)	≤ 250 ms	wasm-bindgen + IndexedDB cache
Embedded (Raspberry Pi 5)	≤ 50 ms	Native arkworks verifier

Table 2: Verifier latency targets across deployment surfaces. Targets are conservative; published Nova-on-BN254 benchmarks consistently report sub-5-ms verify times on commodity desktop hardware.

The advertised "10-millisecond verification" in the title of this paper refers to the laptop target. The desktop number is faster; the browser number is slower but still well within the latency budget of a wallet's first-load operation.

5.4 Proof size

Nova proofs over BN254 are approximately 5 KB when finalized through a Spartan compression step. Lattice proofs are larger: LaBRADOR's published figures put R1CS proofs in the 50–150 KB range depending on the constraint count. Both fit comfortably in a single HTTP response and well below the gossipsub message-size limits we have configured.

6 The Lattice Migration

6.1 Why migrate

Once Phases 1–3 are stable, the only non-post-quantum primitive remaining in the consensus path is the elliptic-curve folding scheme in Nova. This is acceptable transient state but not acceptable long-term. The 2028 lattice migration is what closes that gap.

6.2 Setup options across the Phase 2–4 trajectory

Combining the Nova/BN254 choice (Phases 1–3) with the STARK-attested setup (Section 4) and the candidate lattice schemes for Phase 4 gives four operationally distinct configurations. Table 3 compares them. The middle two rows describe what we actually ship in Phases 2 and 3; the bottom two are the post-quantum migration targets.

Scheme	**Verify**	**Proof size**	**Setup**	**PQ-safe**
Nova/BN254 (MPC ceremony)	∼5 ms	∼5 KB	MPC, N participants	No
Nova/BN254 + STARK-attested SRS (Phase 2–3)	∼5 ms + STARK verify	∼5 KB + ∼50 KB STARK	Transparent (public algorithm + STARK)	No
LatticeFold (Phase 4 candidate A)	∼50 ms (target 10 ms)	50–100 KB	None	Yes
LaBRADOR-IVC + wrapper	∼30 ms (target 10 ms)	50–150 KB	None	Yes
Greyhound (poly commit + adapt.)	∼30 ms (target 10 ms)	100 KB+	None	Yes

Table 3: Setup and verification trade-offs across deployment phases. The boldface row is what Phase 2–3 ship. The bottom three rows are the Phase 4 candidates, all of which avoid setup entirely by virtue of their algebraic construction (no SRS, no trapdoor). The decision among the three lattice candidates is deferred until Phase 2–3 benchmark data is collected.

The Nova-on-BN254 path is not post-quantum, but the STARK attestation of its setup is. Once Phase 4 swaps to a lattice scheme, the entire chain becomes post-quantum end-to-end, and the STARK attestation is no longer needed for that proof system (the lattice schemes require no setup at all). The STARK pattern itself remains useful for other places in the stack (Section 7).

6.3 What stays the same across the swap

The decision to migrate is meaningful only because the swap surface is narrow. The δ-circuit definition in `crates/q-ivc/src/circuits/delta_block.rs` is unchanged. `balance_root_v2` is unchanged. The wire protocol shape (Section 8) is unchanged — only the `proof_version` string and the byte encoding of `proof_b64` differ. The mandatory verification block-height gate logic is unchanged. The advisory-vs-mandatory rollout sequence is unchanged.

What changes is approximately 800 LOC in `crates/q-ivc/src/recursion/`: the `StepCircuit`

implementation is re-emitted against the lattice scheme's API; the folder orchestrator is re-implemented; the verifier is re-implemented; the public parameters file is regenerated.

6.4 Decision pathway

We commit explicitly to the following sequence:

1. **Late 2026:** prototype each of the three candidates as a non-IVC lattice argument of single-block validity. Measure prover time, verifier time, and proof size.
2. **Q1 2027:** decide. Publish a short technical report committing to one of the three.
3. **Q1–Q3 2027:** build the lattice IVC harness around the chosen scheme.
4. **Q4 2027:** side-by-side testnet running Nova and lattice in parallel; identical δ-circuit; compare proof times and verifier times under real block flow.
5. **Q1 2028:** production swap.

7 Beyond Setup: zk-STARK Throughout the Stack

The STARK-attested SRS (Section 4) is the first place we apply zk-STARK in the recursive-bootstrap stack, but it is not the last. The same `crates/q-zk-stark/` infrastructure composes naturally with several other parts of the architecture. We list them here in order of deployment plausibility and label each as *shipped today*, *near-term work*, or *future research*.

7.1 Wallet privacy proofs — shipped today

`crates/q-zk-stark/src/wallet_privacy_stark.rs` (530 LOC) is production code today. A wallet can prove "I own at least X QUG" without revealing the exact balance. The proof is a STARK over a range-membership AIR, verifiable in pure native Rust on any node and in WebAssembly via the same bindings as the recursive SNARK verifier (Section 8). This existing capability is independent of the SRS-attestation pattern but uses the same underlying AIR framework. The relevance to recursive SNARK: it demonstrates that mature, deployed STARK verification in Quillon Graph is not theoretical — nodes already verify STARKs on the consensus path.

7.2 Batched Dilithium signature verification — near-term work

The dominant per-block cost in the δ-circuit is signature verification. Each Dilithium5 signature requires approximately 1.5 M R1CS constraints; a 100-transaction block accumulates 150 M constraints from signatures alone (Section 5). A natural optimization: replace in-circuit signature verification with a *batched STARK proof* that asserts "these K signatures all verify correctly," consumed by the δ-circuit as a single attestation. The reduction is dramatic: STARK signature verification scales nearly linearly in K outside the circuit (a few hundred microseconds per signature on dedicated hardware) and the δ-circuit swallows the result as a constant-size constraint.

The infrastructure is already partly in place: `crates/q-zk-stark/src/batch_prover.rs` (662 LOC) implements batched STARK proving over multiple statements with shared trace columns. We expect this to reduce δ-circuit constraint count by 30–50%, bringing the Nova prover budget per block from $\sim$60 s toward the target of 1 s with commodity hardware.

7.3 Light-client mobile path — near-term work

The recursive SNARK proof verifier targets ≤ 250 ms in WebAssembly (Section 5). For extremely resource-constrained clients — low-end mobile devices, embedded peers, IoT clients — even 250 ms can be too much. A STARK proof of "the most recent N block headers are valid" is verifiable in pure JavaScript (no WASM) in tens of milliseconds, at the cost of trusting only the last N blocks rather than the entire chain history. This is strictly weaker than the recursive SNARK guarantee but acceptable for a light wallet that only needs to know the current state, not the historical chain.

The pattern parallels `wallet_privacy_stark.rs`: a small, mobile-friendly STARK proof, generated by genesis nodes, served alongside the recursive proof for clients that opt into the lighter mode.

7.4 Cross-chain bridge attestations — future work

Recursive SNARK proofs over BN254 are awkward to verify on chains that use different elliptic curves. Verifying a Nova proof on Ethereum, for example, requires implementing BN254 pairing arithmetic in Solidity — expensive in gas and a notable attack surface. STARK proofs use only hash functions and are agnostic to the destination chain's curve choice. A STARK proof of "the Quillon Graph state root at height H is R" can be verified on Ethereum (via the Cairo-style verifier pattern), Bitcoin (via Tapscript with appropriate hash gadgets), or any chain that supports hash evaluation in its scripting language.

This is the natural way to bridge Quillon Graph state to other chains without trusting a multi-sig committee: the destination chain verifies a STARK over the bridged state root and accepts it as canonical. We anticipate this becoming relevant in 2027 when bridging questions arise in production. `crates/q-zk-stark/src/blockchain_state_circuit.rs` (410 LOC) is the existing STARK circuit over blockchain state transitions that would underpin this.

7.5 Activation-height attestations — future work

When a consensus rule activates at a known block height (e.g., `BALANCE_ROOT_V2_HEIGHT`), a STARK proof attesting "the state transition at this height correctly applies the rule activation" provides an independent audit trail. Adversarial activations (a malicious miner producing a block at the activation boundary with inconsistent semantics) are detectable by any client that verifies the activation STARK. The pattern reuses `blockchain_state_circuit.rs`.

7.6 Encryption SRS proof — shipped today, foundational reference

We close this section by emphasizing that the transparent-setup pattern of Section 4 is not new infrastructure. The encryption SRS at chain genesis was produced by precisely this pattern and lives in production today as a 176,832-byte STARK proof at `data-mainnet-genesis/encryption.zkstark`. Every full node verifies it at startup. The Nova-SRS attestation is the same engineering applied to a different generator circuit.

8 Bootstrap Protocol and Progressive Archival

8.1 Wire protocol

The bootstrap surface is three additions to the existing HTTP and gossipsub layers.

`GET /api/v1/proof/tip`. Returns the latest folded proof and the block header it commits to. No authentication — the proof is publicly verifiable, so the API exposes no authority.

```
{
  "tip_height": 11400000,
  "state_root": "0xabcd...",
  "block_header": {
    "height": 11400000,
    "parent_hash": "0x...",
    "tx_root": "0x...",
    "state_root": "0xabcd...",
    "timestamp": 1771761600,
    "producer_id": 123,
    "anchor_validator": "..."
  },
  "proof_version": "nova-bn254-v1",
  "proof_size_bytes": 51232,
  "proof_b64": "<base64-encoded proof bytes>"
}
```

Listing 3: `GET /api/v1/proof/tip` response body.

`POST /api/v1/proof/verify`. Optional helper for clients without an arkworks verifier compiled in (notably browser wallets that prefer not to bundle WASM crypto). Submits a proof and returns a boolean plus a verify-time measurement. This endpoint is a convenience; the trustless path is for the client to verify locally.

Gossipsub topic `/qnk/mainnet-genesis/recursive-proof`. Genesis nodes publish $(s_n, \pi_n, \text{header}_n)$ on a new gossipsub topic with one message per block. Peers maintain "latest seen proof" and forward only higher-height proofs. Rate-limited to one message per peer per block.

8.2 Bootstrap-mode CLI flag

The Quillon Graph node binary gains a single new flag:

```
--bootstrap-mode=proof # Download proof, verify, start immediately
--bootstrap-mode=checkpoint # Existing behavior: trust a signed snapshot
--bootstrap-mode=genesis # Existing behavior: full replay from height 1
```

`proof` is opt-in throughout Phase 3 (the advisory window) and becomes the default in Phase 5 (mandatory verification).

8.3 Progressive archival UX

The proof-bootstrap delivers a verified state root, but it does not deliver historical blocks. An archive backfill runs in parallel after bootstrap. Historical-block queries against blocks not yet backfilled need a graceful UX.

We add a status endpoint and three response headers:

```
{
  "tip_height": 11400000,
  "lowest_indexed_height": 4321001,
  "archive_complete": false,
  "archive_progress_pct": 37.9,
  "archive_eta_seconds": 64800,
  "blocks_per_sec_recent": 175,
  "verified_proof_height": null
}
```

Listing 4: `GET /api/v1/status/archive` response body.

Every historical-lookup endpoint (`/blocks/h`, `/tx/id`, `/receipts/h`) carries three response headers: `X-QNK-Archive-Status: complete|backfilling`, `X-QNK-Archive-Progress: <indexed>/<tip>`, and `X-QNK-Archive-Lowest-Indexed: <height>`. Queries for heights below `lowest_indexed_height` return HTTP 202 (Accepted) with a body describing where the data will be available, instead of HTTP 404. Wallet and explorer UIs render this as "Block X not yet indexed, ETA T" with a clearly marked opt-in proxy to a fully-indexed peer.

This UX ships in v10.9.16 *independently of the SNARK work*: the existing two-phase backfill already produces `lowest_indexed_height`, so the endpoint and headers light up immediately and are useful even before the proof bootstrap exists.

9 Security Analysis

9.1 Soundness

The soundness of the bootstrap argument decomposes into three layers.

Layer 1: the underlying proof system. Nova on BN254 is sound under the discrete-logarithm assumption on the BN254 curve and the Random Oracle Model. We will rely on the published Nova literature through Phase 3 and on the published lattice scheme analyses in Phase 4. We commit to not modifying the proof system internals beyond the published reference implementation.

Layer 2: the δ-circuit's correctness. The circuit is sound if and only if it faithfully encodes the block-validity predicate. This is the layer where bugs can be introduced and is the principal soundness risk. Mitigation: an adversarial-witness test suite covering all five well-defined fail modes (invalid signature, insufficient balance, state-root mismatch, replay-nonce reuse, over-emission coinbase) is mandatory before mandatory verification at activation.

Layer 3: the wire path. The new node fetches a proof from an arbitrary peer. The peer can return whatever bytes it likes. A forged proof is rejected by the local verifier; a valid proof for a different state root than the claimed one is rejected by the post-verification consistency check at line 6 of Algorithm 1. A network-level adversary who controls all peers can stall the bootstrap but cannot induce the new node to accept a wrong state root.

9.2 Setup trust assumptions

Section 4 described the transparent-setup pattern in constructive detail. Here we make its trust assumptions precise and contrast them with the alternatives:

- **Naive Nova (rejected).** A single dealer generates the SRS, keeping the trapdoor τ. Trust assumption: "the dealer discarded τ." Operationally untenable for a $2 billion mainnet.
- **Nova + MPC ceremony (industry standard, rejected here).** N contributors mix randomness. Trust assumption: "at least one of N participants is honest *and* actually discarded their contribution." Logistically expensive; requires coordinated public ceremony; must be re-run if the circuit changes.
- **Nova + STARK-attested transparent setup (what we ship).** Trust assumption: "STARK soundness in the Random Oracle Model on BLAKE3 [7, 15] *and* the public seed is unpredictable at SRS-generation time." No coordination required; the proof is reproducible; mid-chain re-attestation is a single transaction.

- **LatticeFold / LaBRADOR / Greyhound (Phase 4).** No setup at all. Trust assumption reduces to the chosen scheme's hardness assumption (Module-SIS / RLWE).

The STARK assumption stack is non-circular: STARK soundness only needs BLAKE3 to behave as a random oracle. We already rely on BLAKE3 throughout the chain for state roots, block hashes, and SMT leaves; the STARK attestation does not introduce a new cryptographic primitive into the trust model.

The STARK-attested pattern is already in production for a related setup: the genesis encryption key schedule (`data-mainnet-genesis/encryption.zkstark`, 176,832 bytes, generated by `crates/q-storage/src/encryption_zkstark.rs`). The Nova-SRS attestation is the same pattern applied to a different generator circuit, and reuses every component of `crates/q-zk-stark/`.

9.3 Phased deployment with advisory window

The single most important mitigation against δ-circuit bugs is operational: the proof is **advisory** for at least six months on mainnet before any node makes a consensus-affecting decision based on the proof's validity. During the advisory window:

- Every block is still validated block-by-block on every node.
- The proof is published and verified, but its acceptance or rejection does not influence consensus.
- Mismatches between block-by-block validation and proof verification are logged as alerts; any non-zero rate of mismatches *must* block the transition to mandatory verification.

Only after the advisory window completes with zero soundness discrepancies does the chain activate mandatory verification at a publicly announced block height.

9.4 Lattice parameter selection

When Phase 4 lands, parameters are chosen to meet at least NIST PQC level 1 security against both classical and quantum adversaries, per [21]. Specific parameter sets will be published as part of the Phase 4 technical report once a candidate is chosen.

10 Comparison to Related Systems

Several deployed and proposed systems exploit recursive SNARKs for blockchain bootstrap or for compact verification. We summarize the relevant ones in Table 4.

Mina's Pickles construction is the closest precedent: a recursive SNARK that compresses arbitrary computation, deployed in a live blockchain, with a constant-size proof of the entire history. Mina's verify time is slower than ours (the construction is tuned for absolute proof size rather than verify time) and it is not post-quantum. Halo 2 and Marlin similarly trade verify time against proof size in pairing land. Filecoin's Groth16 is single-purpose (proof of replication / proof of spacetime); it is not a general consensus bootstrap.

The contribution of the Quillon Graph plan, relative to these predecessors, is the explicit post-quantum migration plan combined with sub-10-ms verification and the operational UX integration described in Section 8.

11 Phased Rollout

A summary of phases is given in Table 5. We restate the cadence in prose here to emphasize the dependencies.

System	Proof system	Proof size	Verify time	PQ?
Mina	Pickles (Halo 2-style on Pasta)	∼200 B	∼200 ms	No
Zcash (Halo 2)	Halo 2 (Pasta)	∼3 KB	∼10 ms	No
Aleo	Marlin (BLS12-377/BW6-761)	∼1 KB	∼20 ms	No
Filecoin	Groth16 (BLS12-381)	192 B	∼2 ms	No
Quillon Graph (this work)	**Nova/BN254 → Lattice IVC**	**5–150 KB**	**5–10 ms**	**Yes (Phase 4)**

Table 4: Recursive SNARK deployments at the time of writing. The distinguishing property of the Quillon Graph plan is end-to-end post-quantum security *after Phase 4*, with the entire signature, key-encapsulation, hash, and (eventually) recursion path PQ-secure. Verify-time numbers are published benchmarks for the respective systems; proof sizes are nominal full-finalized proof sizes.

Phase	Target	Deliverable
Phase 0	2026-Q2 (in flight)	Gadget library complete; SMT shadow-mode; archive-status UX live
Phase 1	2026-Q3	δ-circuit composed, single-block proof on testnet
Phase 2	2026-Q4	Nova IVC wrapper; folded proofs on testnet
Phase 3	2027-Q3	Advisory mainnet integration; `-bootstrap-mode=proof` opt-in
Phase 4	2028-Q1	Lattice migration; trusted setup eliminated
Phase 5	2028-Q2/Q3	Mandatory verification at announced activation height

Table 5: Phased rollout. Phases 0–1 are funded engineering work against scoped specifications; Phase 4 requires R&D that is scoped but not committed-pathway. Phase 5 is gated on at least six months of advisory-mode soak with zero soundness discrepancies.

Phase 0 dependencies. Phase 0 is the only phase that is partially complete today. The gadget library (BLAKE3, Poseidon, NTT, Dilithium5) ships in `crates/q-ivc/`; the SMT ships in `crates/q-storage/src/balance_smt.rs`; the archive-status endpoint ships in `crates/q-api-server/src/har` What remains in Phase 0 is the Merkle-path gadget over BLAKE3 (∼600 LOC, three engineering weeks); a small range-check wrapper around `enforce_norm_bound` (∼3 days); and three test-fixture corrections in the Dilithium gadget tests (∼1.5 days).

Phase 1 dependencies. The δ-circuit composition requires Phase 0 complete plus an additional ∼1,500 LOC of glue connecting the existing gadgets through the per-transaction loop and the final state-root equality check.

Phase 2 dependencies. The Nova wrapper requires Phase 1 complete, plus selection between the `nova-snark` (Microsoft) and the `arkworks-rs/nova` (community) crate. We expect to spend approximately three engineering days prototyping both on a toy step circuit before committing.

Phase 3 dependencies. Integration into the production API server, gossipsub topic registration, opt-in CLI flag, and a six-month advisory soak before any further consensus-affecting decision can be made.

Phase 4 dependencies. A live, instrumented, advisory-mode Nova deployment from Phase 3 provides the benchmark data needed to choose among the three lattice candidates. We do not bind to a candidate before that data is collected.

Phase 5 dependencies. Six months of clean advisory soak. A publicly announced activation height at least 20,000 blocks (∼5.5 hours, but in practice we will announce on the order of weeks) in the future. Old binaries continue to validate via the block-by-block path until they choose to upgrade.

12 Implementation Status

We close with an honest inventory of what is shipped, what is drafted, and what is not yet started.

Shipped, in production (shadow mode where applicable):

- `crates/q-storage/src/balance_smt.rs` (743 LOC). Depth-256 sparse Merkle tree with BLAKE3 leaf and node hashes, empty-subtree precompute, RocksDB-backed persistence, atomic batch updates. Twelve unit tests covering empty tree, single-leaf insertion, random-tree updates, batch-vs-sequential equivalence, and adversarial sibling tampering.
- `crates/q-ivc/src/gadgets/blake3.rs` (551 LOC). Single-block BLAKE3 compression gadget with the `verify_hash` entry point. Multi-block extension is the "Path A" handoff to an external implementer and is the principal blocker for the Merkle-path gadget at depth 256.
- `crates/q-ivc/src/gadgets/poseidon.rs` (329 LOC). Poseidon hash gadget; reserved for a potential v3 state commitment.
- `crates/q-ivc/src/gadgets/ntt.rs` (769 LOC). Cooley–Tukey decimation-in-time NTT butterfly. The roots$[m + i] = \omega^{i \cdot n/(2m)}$ negacyclic convention compatible with FIPS 204 is implemented and tested; the BLAKE3 `FpVar`↔`UInt32` bridge is in place.
- `crates/q-ivc/src/gadgets/dilithium.rs` (902 LOC). Dilithium5 verification primitives: `compute_az_minus_ct` in both standard cyclic and negacyclic forms, `enforce_norm_bound`, `enforce_signed_norm_bound` (positive-only with documented CAVEAT pending an upstream arkworks `FpVar::is_cmp` fix), `high_bits` per FIPS 204 §5.4, and `use_hint` per FIPS 204 §6.5.2.
- `crates/q-zk-stark/` (4,469 LOC across nine source files). Full zk-STARK toolkit: `air.rs` (429 LOC) for AIR construction, `stark_prover.rs` (442 LOC), `stark_verifier.rs` (548 LOC), `batch_prover.rs` (662 LOC), `polynomials.rs` (549 LOC) for FRI-style polynomial commitments, `blockchain_state_circuit.rs` (410 LOC), `wallet_privacy_stark.rs` (530 LOC, deployed), `performance.rs` (591 LOC) for benchmark harness, and a `gpu/` subtree for hardware acceleration. Approximately 70–80% of the STARK surface is shipped; what is missing is the specific AIR for the Nova-SRS generator (Section 4) and the AIR for batched Dilithium verification (Section 7).
- `crates/q-storage/src/encryption_zkstark.rs` (562 LOC). `EncryptionKeyProof` for the transparent-setup attestation of the chain's encryption SRS. In production. The artifact `data-mainnet-genesis/encryption.zkstark` (176,832 bytes) is the proof itself, committed to genesis state.

Drafted, not wired in:

- `crates/q-ivc/src/circuits/epoch_transition.rs` (208 LOC). An `EpochTransitionCircuit` skeleton with `ValidatorSignatureInput` and `EpochTransitionInputs` types; the composition logic through the per-transaction loop is not yet written.

Not yet started:

- `crates/q-ivc/src/gadgets/merkle.rs`. The Merkle-path gadget at depth 256 over BLAKE3. Blocked on the multi-block BLAKE3 extension (above).
- `crates/q-ivc/src/circuits/delta_block.rs`. The full δ-circuit composition. Scoped at $\sim$1,500 LOC.
- `crates/q-ivc/src/recursion/`. The Nova wrapper. Scoped at $\sim$800 LOC.
- `crates/q-api-server/src/handlers.rs` additions for `/api/v1/proof/tip` and `/api/v1/proof/verify`. Scoped at one engineering week.
- The Phase 4 lattice IVC implementation.
- The specific AIR for the Nova-SRS generator inside `crates/q-zk-stark/`, plus the corresponding generation ceremony tooling. Scoped at $\sim$300–500 LOC of AIR code atop the shipped `crates/q-zk-stark/` infrastructure.

Test debt explicitly tracked: three Dilithium gadget tests still use the pre-fix `roots[1] = -1` convention rather than the post-fix `roots[1] = 1` convention for $n = 2$; `test_use_hint_with_bias` has an off-by-one in its expected value; `enforce_signed_norm_bound` is positive-only with a documented CAVEAT. None affect the soundness of the deployed gadgets; all are book-keeping fixes scheduled for the next `q-ivc` task pass.

13 Conclusion

A user joining the Quillon Graph network today waits hours, or accepts a signed checkpoint and waits seconds with a trust caveat. The construction described in this paper closes that gap. A fresh node fetches a constant-size recursive proof, verifies it locally in about 5–10 ms, and is immediately able to mine, transact, and serve current-state queries — with cryptographic certainty in the state root, not the trust of a checkpoint operator. Historical blocks backfill in the background, exposed through a deliberate UX surface (`X-QNK-Archive-*` headers, HTTP 202 for not-yet-indexed heights) so users see honest information about which queries are answered locally and which are still being indexed.

The plan is phased deliberately. Phase 0 ships today: gadgets, SMT, and progressive-archival UX. Phases 1–3 will roll the δ-circuit, the Nova fold, and the advisory mainnet integration through 2026 and into 2027. Phase 4 swaps Nova for a lattice IVC scheme in early 2028, eliminating the trusted setup and the last classical-only cryptographic dependency in the consensus path. Mandatory verification at an announced activation height is targeted for mid-to-late 2028 — only after at least six months of advisory soak with zero soundness discrepancies. We are not in a rush. A multi-billion-dollar mainnet does not need fast cryptography; it needs cryptography that has been debugged in advisory mode for long enough that no one is surprised when it becomes mandatory.

What we can promise today is that the foundations are in place: the state commitment is shipped, the gadget library is shipped, the zk-STARK toolkit (`crates/q-zk-stark/`, 4,469 LOC) and a production transparent-setup proof at chain genesis are shipped, the deployment cadence is published, and the architectural surface is small enough that a Phase 4 swap of the proof system underneath does not require touching any of the higher layers. The promise of an instantly trustless bootstrap is no longer hypothetical — it is a finite, scoped engineering effort against a codebase that is already mostly there.

We do not invent zk-STARK infrastructure for this work. We extend the project's existing production STARK stack — the encryption-SRS proof at genesis, the wallet-privacy proofs, the blockchain-state circuit, the batched prover, the AIR framework, the FRI-style polynomial commitments — to also attest the recursive SNARK's setup. The same engineering composes to batched signature verification, mobile light-client paths, cross-chain bridge attestations, and activation-height proofs. Recursive SNARK and recursive STARK are not competitors here: they are layered, complementary, and live in the same crate hierarchy.

References

[1] Paul Valiant. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. *TCC*, 2008.

[2] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. Cryptology ePrint Archive, Paper 2021/370, 2021.

[3] Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023.

[4] Ward Beullens and Gregor Seiler. LaBRADOR: Compact Proofs for R1CS from Module-SIS. Cryptology ePrint Archive, Paper 2023/1119, 2023.

[5] Dan Boneh, Binyi Chen, and Akshay Tairi. LatticeFold: A Lattice-Based Folding Scheme and its Applications to Succinct Proof Systems. Cryptology ePrint Archive, Paper 2024/257, 2024.

[6] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast Polynomial Commitments from Lattices. Cryptology ePrint Archive, Paper 2024/1293, 2024.

[7] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One Function, Fast Everywhere. Specification, BLAKE3 Team, 2020.

[8] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. FIPS PUB 204, 2024.

[9] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS PUB 203, 2024.

[10] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized Cryptocurrency at Scale (Mina Protocol Technical Whitepaper). O(1) Labs, 2020.

[11] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments. Cryptology ePrint Archive, Paper 2021/1536, 2021.

[12] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. Cryptology ePrint Archive, Paper 2019/1047, 2019.

[13] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. *EUROCRYPT*, 2016.

[14] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. Cryptology ePrint Archive, Paper 2019/458, 2019.

[15] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018.

[16] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed–Solomon Interactive Oracle Proofs of Proximity. *ICALP*, 2018. Cryptology ePrint Archive Paper 2017/134.

[17] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo — a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive, Paper 2021/1063, 2021.

[18] Polygon Zero Team. Plonky2: Fast Recursive Arguments with PLONK and FRI. *Polygon Zero*, 2022. Available at `github.com/0xPolygonZero/plonky2`.

[19] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proc. ACM CCS*, 1993.

[20] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proc. 35th Annual Symposium on Foundations of Computer Science*, 1994.

[21] National Institute of Standards and Technology. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. NISTIR 8413, 2022.

[22] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. DAG-KNIGHT: A Parameterless Generalization of the GHOSTDAG Protocol. Cryptology ePrint Archive, Paper 2022/1494, 2022.

[23] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. *EuroSys*, 2022.