QBit SmartChain: A ZK-Native Post-Quantum Blockchain Protocol

Achinta Das
achinta@dasgen.in

Tumukunde Arnold
tumukundearnold@gmail.com

QBit Core Team
February 2026

Version 1.0

Abstract

Post-quantum signature schemes produce signatures an order of magnitude larger than their elliptic-curve predecessors, and no existing blockchain can absorb the cost at high throughput. QBit SmartChain is a ZK-native post-quantum protocol that solves this by never gossiping signatures at all. Transactions are signed with ML-DSA-65 (3,309-byte signatures, 50× Ed25519) and submitted to a permissionless set of GPU-equipped prover nodes (Sentries), which verify signatures locally and produce a ZK-STARK proof attesting to the validity of each 1,000-transaction batch. Validators receive only the proof (≈147 KB, verifiable in ≈20 ms) and the resulting state diff. Every proof carries a `chain_id` field. QBit's native chain operates as `chain_id = 0`; sovereign application-specific chains (QChains) inherit the same post-quantum security and shared finality by registering additional `chain_id` values permissionlessly through an economic bond, with cross-chain messaging verified via Merkle proofs against the shared DAG's finalized state roots—no bridges, no relayers, no challenge periods. Sender-based transaction routing ensures parallel Sentries do not contend on nonce state; a structured relay tree limits gossip amplification to $\approx 3\times$; and dictionary compression shrinks state diffs by $\approx 3\times$. Validator ingress at 30,000 TPS is ≈ 237 Mbps, within reach of commodity 500 Mbps fiber. Consensus finality is $\leq$3 seconds. The cryptographic stack is post-quantum end to end: ML-DSA-65 for signatures, ML-KEM-768 for committee encryption, SHA3-256 and Poseidon2 for hashing, hash-based STARKs for proofs, and an iterated-Poseidon2 sequential work function for leader-election randomness. The protocol-level security floor is 120 bits, bounded by recursive STARK composition; all individual components meet or exceed 128 bits. Versioned interfaces for signatures, hashes, and execution environments allow migration to future PQC standards and activation of new QChains without hard forks.

1 Introduction

1.1 The Problem

Replacing Ed25519 with a post-quantum signature scheme is not a drop-in substitution. The cost shows up in three places at once:

1. Bandwidth. ML-DSA-65 [4] (FIPS 204)—NIST's primary recommendation for general-purpose digital signatures—produces 3,309-byte signatures, roughly 50× larger than Ed25519. At 30,000 TPS:

$$30{,}000 \times 3.3 \text{ KB} \approx 99 \text{ MB/s (single stream)} \tag{1}$$

Under standard gossip protocols (GossipSub mesh degree $D = 8$), actual validator ingress reaches $99 \times 8 \approx 792$ MB/s ≈ 6.3 Gbps.

2. Verification compute. Each ML-DSA-65 verification costs ≈ 0.15 ms on optimized implementations—3× slower than Ed25519. At 30,000 TPS, a single core spends ≈ 4.5 s of CPU time per second on signatures alone. Even with 4-core parallelism, signature verification consumes the entire CPU budget before accounting for state execution, gossip processing, or consensus.

3. Cryptographic infrastructure. Classical primitives (BLS aggregation, EC-based VRFs, pairing-based KZG commitments) that underpin modern consensus, randomness, and data availability have no direct post-quantum replacements with comparable efficiency.

4. Multi-chain fragmentation. Application-specific blockchains address scalability but fragment security. Each chain must independently bootstrap a validator set and economic security, and cross-chain bridges—the weakest link—have lost over $2.9 billion to exploits through 2025. Post-quantum migration makes this worse: every bridge must separately adopt PQC signatures, and ML-DSA-65's 3,309-byte signatures multiply the cost of cross-chain attestations.

Taken together, these constraints make a naive "swap Ed25519 for ML-DSA" migration infeasible at high throughput—and the problem compounds across chains. Section 1.4 details the rationale for selecting ML-DSA-65 over smaller-signature alternatives.

1.2 Our Approach

Validators do not need signatures. They need proof that the signatures were checked.

We split the network into two tiers. **Sentries**—GPU-equipped prover nodes—collect transactions, verify their ML-DSA signatures locally, and produce a ZK-STARK proof [6] attesting to the validity of an entire batch. **Validators** verify only the proofs. Because STARK verification is polylogarithmic in the statement size, a batch of 1,000 signatures collapses to a single 147 KB proof verifiable in 20 ms. The gossip layer never sees a signature.

Every proof carries a `chain_id` identifying its originating chain. QBit's native chain operates as `chain_id = 0`. The same Sentry infrastructure can prove execution traces from any registered chain—sovereign application-specific blockchains (QChains) inherit QBit's post-quantum security, shared finality, and data availability by implementing a standard Provable Execution Interface. Because all chains are finalized by the same Helix-ZK DAG, cross-chain messaging reduces to Merkle path verification against finalized state roots—eliminating bridges, relayers, and challenge periods entirely.

Multiple Sentries prove in parallel. To prevent them from wasting work on overlapping state, each transaction is deterministically assigned to exactly one Sentry by sender address using slot-seeded randomness (Section 2.8). This eliminates nonce conflicts entirely and keeps residual state collisions below 10% of proving work. Proofs and state diffs propagate to validators via a structured relay tree (Section 2.6) rather than unstructured mesh gossip, reducing the network amplification factor from 8× to ≈ 3×. State diffs are dictionary-compressed before transmission (Section 2.7), exploiting the structural redundancy of account-model updates.

Per-stream data rate falls from ≈ 99 MB/s to ≈ 8.8 MB/s. After relay-tree amplification, validator ingress is ≈ 237 Mbps—within reach of a commodity 500 Mbps fiber connection.

1.3 Contributions

This paper presents:

1. **Helix-ZK**, a leaderless DAG-based consensus protocol that orders proof batches from multiple chains rather than individual transactions, with formal safety and liveness proofs under partial synchrony.
2. An **AIR constraint system** for verifying ML-DSA-65 (FIPS 204) signatures inside a STARK circuit, with projected constraint estimates.
3. A **fully post-quantum cryptographic stack**—ML-DSA-65 signatures, ML-KEM-768 committee encryption, hash-based STARKs, Poseidon2 and SHA3-256 hashing, and iterated-Poseidon2 SWF randomness—with no component relying on classical hardness assumptions.
4. A **QChain framework** with a Provable Execution Interface (PEI) that enables sovereign application-specific chains to share QBit's post-quantum proving infrastructure, DAG finality, and data availability. Cross-chain messaging requires no bridges—verification is a Merkle proof against the shared DAG's finalized state.
5. A **competitive proof mining economy** that prevents prover centralization through multi-leader selection, slashing bonds, and game-theoretically unstable cartels.
6. **Crypto-agile versioned interfaces** for signatures, hashes, and execution environments, enabling migration to future PQC standards and permissionless activation of new QChains without protocol disruption.

1.4 Signature Scheme Selection: Why ML-DSA-65

NIST has standardized three post-quantum signature families, each with distinct trade-offs. QBit selects ML-DSA-65 (FIPS 204, NIST Security Level 3) after a careful evaluation against blockchain-specific requirements.

Property	**ML-DSA-65**	**FN-DSA-512**	**SLH-DSA-192s**
NIST Standard	FIPS 204	FIPS 206 (IPD Aug 2025)	FIPS 205
Signature Size	3,309 B	666 B	16,224 B
Public Key Size	1,952 B	897 B	48 B
Security Level	Level 3 (≥128-bit PQ)	Level 1 (≥64-bit PQ)	Level 3 (≥128-bit PQ)
Arithmetic	Integer-only	Floating-point	Hash-only
Constant-Time	Straightforward	Requires FPU emulation	Straightforward
Standardization	Finalized (Aug 2024)	IPD under review (est. late 2026)	Finalized (Aug 2024)

Table 1: NIST PQC Signature Scheme Comparison

Why not FN-DSA (Falcon)? FN-DSA-512 offers ≈ 5× smaller signatures (666 B vs. 3,309 B), which would reduce single-stream bandwidth to ≈ 20 MB/s before ZK compression. However, FN-DSA has critical drawbacks for a blockchain protocol:

1. **Side-Channel Vulnerability.** FN-DSA signing requires high-precision floating-point FFT arithmetic. The "FALCON Down" attack [17] demonstrated full key recovery from ≈ 10,000 electromagnetic measurements on ARM Cortex-M4 hardware. Subsequent work by Zhang et al. [18] achieved key recovery with substantially fewer traces via covariance-based spectral analysis exploiting leakage from integer Gaussian sampling, and Guerreau et

al. [19] extended the attack surface via the hidden parallelepiped technique applied to the base sampler output. NIST has acknowledged these concerns and submitted the FIPS 206 Initial Public Draft for approval in August 2025, including consideration of fixed-point arithmetic alternatives for signing. However, the signing algorithm in the current draft still requires floating-point. In a permissionless Sentry network, where operators control their own hardware, side-channel resistance is not optional.

2. **Implementation Complexity.** Constant-time floating-point emulation is fragile and platform-dependent. ML-DSA's integer-only design makes constant-time implementation straightforward on any architecture—critical for a heterogeneous validator and Sentry network.

3. **Standardization Status.** FN-DSA (FIPS 206) had its Initial Public Draft submitted for approval in August 2025 and was presented at the 6th NIST PQC Standardization Conference (September 2025). Final standardization is projected for late 2026–early 2027. ML-DSA (FIPS 204) was finalized in August 2024 and is NIST's explicit primary recommendation for general-purpose digital signatures.

4. **STARK Circuit Compatibility.** ML-DSA's core operations—polynomial multiplication via NTT over $\mathbb{Z}_q$, modular reduction, and rejection sampling—use integer arithmetic that maps naturally to STARK AIR constraints. FN-DSA's floating-point FFT would require emulated double-precision inside the circuit, roughly 3–5× more constraints per signature.

Why not SLH-DSA (SPHINCS+)? SLH-DSA is hash-based and relies on minimal cryptographic assumptions (collision resistance of hash functions), making it the most conservative choice. However, its smallest NIST Level 3 variant (SLH-DSA-192s) produces 16,224-byte signatures—$\approx 5\times$ larger than ML-DSA-65. At 30,000 TPS, this would require ≈ 487 MB/s signature bandwidth (single stream), dwarfing even the naive ML-DSA figure. Additionally, SLH-DSA signing is orders of magnitude slower ($\approx 100\times$), making it impractical for high-throughput transaction signing.

ML-DSA-65 is the right fit for a ZK-proven blockchain. Its signatures are large—but that cost is absorbed entirely by the proving layer, so it never reaches the gossip network. Its integer arithmetic maps cleanly into STARK constraints, and constant-time implementation is straightforward across hardware. If FN-DSA resolves its side-channel concerns upon finalization, the crypto-agile interface described below permits adoption without protocol changes.

1.5 Cryptographic Agility

Cryptographic standards evolve. FN-DSA may mature and resolve its side-channel concerns; new lattice or code-based schemes may emerge from NIST's ongoing additional signatures competition. QBit is designed to accommodate such transitions without consensus-breaking changes.

Versioned Signature Interface. Every transaction carries a 1-byte `sig_version` field:

$$tx = (\ldots,\ \texttt{sig\_version} : \texttt{u8},\ \sigma : \texttt{bytes}) \tag{2}$$

The current assignment is $\texttt{sig\_version} = 0 \Rightarrow$ ML-DSA-65. New schemes are assigned incrementally via governance.

Circuit Modularity. The STARK prover treats signature verification as a *pluggable sub-circuit*. Adding a new scheme requires:

1. Implementing the AIR constraint system for the new scheme.

2. Deploying updated prover software to Sentries (no validator changes required for verification, since validators verify the STARK proof, not the signature directly).

3. Governance vote to activate the new `sig_version`.

Migration Path. Users migrate at their own pace:

- Existing accounts continue using ML-DSA-65 indefinitely.
- Users opt in to a new scheme by generating a new keypair and transferring assets.
- Sentries support multiple signature schemes concurrently—each batch proof attests to a set of signatures under *any* supported scheme.
- A governance-controlled sunset window (e.g., 2 years) can deprecate older schemes if a cryptographic break is discovered.

Together, these mechanisms avoid locking QBit into a single signature scheme without incurring the complexity of hybrid (dual-signature) approaches.

Versioned Hash Interface. Analogous to the signature versioning above, the in-circuit hash function is abstracted behind a `hash_version` field in the epoch configuration:

$$\texttt{hash\_version} = 0 \Rightarrow \text{Poseidon2 } (t{=}16, R_f{=}8, R_p{=}14, \alpha{=}7, \text{Goldilocks}) \tag{3}$$

Future assignments (e.g., $\texttt{hash\_version} = 1$ for increased-round variants or NIST-standardized alternatives) are activated via governance. Migration uses a "lazy rehash" strategy: newly modified accounts adopt the updated hash immediately, while untouched accounts are rehashed in background batches over a governance-specified transition period (default: 30 days). During transition, the STARK circuit accepts both hash versions. This enables governance-activated migration to improved arithmetization-friendly hashes without consensus-breaking changes; see Section 3.1.2 for the security analysis motivating this capability.

Chain Agility. The same versioned interface pattern extends to execution environments. Every proof carries a 2-byte `chain_id` field identifying the originating chain:

$$\texttt{chain\_id} = 0 \Rightarrow \text{QBit native chain (QVM execution)} \tag{4}$$

New chains are registered permissionlessly by locking a registration bond and paying annual rent (Section 12). The Sentry network proves execution traces from any registered chain using the same STARK prover infrastructure. Validators verify proofs identically regardless of `chain_id`—the proof's validity is chain-agnostic. When only `chain_id = 0` exists, the protocol is identical in every observable way to a single-chain system. The framework components exist but are dormant; QChains activate as the Sentry network matures (Section 10).

2 System Model and Architecture

2.1 Network Topology and Roles

To isolate the high bandwidth and compute requirements of PQC verification from the consensus layer, QBit adopts a two-tier network topology.

2.1.1 Tier 1: The Validators (The "Governors")

Validators are the keepers of the consensus. They are responsible for:

- **DAG Maintenance:** Organizing incoming proof batches into a conflict-free causal history.
- **ZK Verification:** Verifying ZK-STARK proofs submitted by Sentries. The cost of verification is $O(\text{polylog } N)$, decoupling it from batch size.

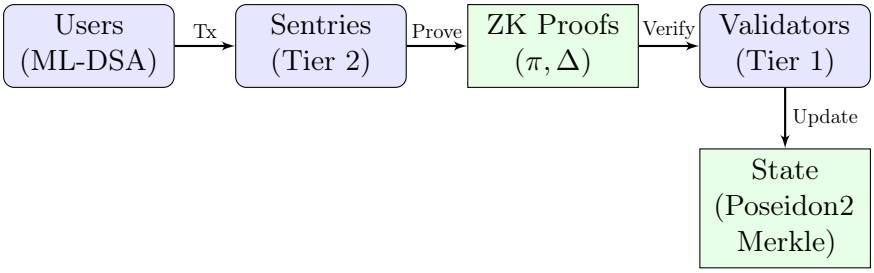

Figure 1: QBit Two-Tier Architecture: Separation of Proving and Consensus

- **Data Availability Sampling (DAS):** Ensuring data is public via erasure code sampling.
- **State Execution:** Applying the state diffs Δ committed to in the proofs.

Hardware Specifications: Mid-range server hardware (4-Core CPU, 16 GB RAM, 256 GB NVMe SSD, 500 Mbps dedicated connection). Sustained ingress bandwidth at the 30,000 TPS baseline is $\approx$ 237 Mbps (Proof Relay Tree amplification $\approx 3\times$ for bulk data, GossipSub D=6 for control messages including ML-DSA-65 attestations; see Section 2.6), scaling linearly with throughput. This is achievable on commodity 500 Mbps fiber connections widely available in residential and small-business tiers; validators should provision headroom for network spikes and control traffic.

2.1.2 Tier 2: The Sentries (The "Provers")

Sentries are a permissionless set of node operators who compete to generate proofs. **Hardware Specifications:** High-performance GPU hardware (4× RTX 4090 or equivalent, 256 GB+ RAM, 200 Mbps connection). The bandwidth requirement is modest: each Sentry handles $\approx$ 75 TPS of inbound transactions ($\approx$ 250 KB/s), distributes one batch of DA chunks ($\approx$ 1.8 MB) every $\approx$ 8s via the relay tree, and submits one proof + compressed diff ($\approx$ 222 KB) per batch. Total sustained egress is under 1 MB/s. The bottleneck is GPU compute, not network.

Chain Support Registration. Each Sentry registers the set of `chain_id` values it is willing to prove. At network genesis, all Sentries register support for `chain_id = 0` (QBit native). When a new QChain activates, Sentries voluntarily update their registration to include the new `chain_id`. A Sentry is only eligible for leader selection on chain C if $\texttt{chain\_id}_C$ is in its registered set. The proving bond ($B_{\text{prove}} = 10{,}000$ QBIT) covers all chains the Sentry supports—no additional bond per chain is required. Slashing conditions (Section 7.1) apply per-chain: a Sentry that fails to prove for a chain it registered support for is subject to the same missing proof penalties as for the native chain. A registered QChain requires at least $N_{\text{min\_sentries}} = 5$ supporting Sentries before activation.

2.2 Mempool Architecture and Censorship Resistance

Transactions enter a distributed mempool designed for deterministic inclusion and censorship resistance. Spam and DoS defense are enforced at ingestion time.

2.2.1 Protocol Specification

The following mempool protocol governs transaction ordering for `chain_id = 0` (QBit native), where Sentries select and order transactions. For QChains with sovereign consensus (`chain_id > 0`), batch contents are determined by the QChain's own consensus mechanism before reaching the Sentry. The Sentry's role for sovereign QChains is proving execution validity, not transaction selection. The synchronized commitment protocol (Protocol 2.2) and deterministic selection therefore apply exclusively to `chain_id = 0`.

Protocol 2.1 (Mempool Lifecycle and Batch Selection)**. Stage 1: Ingestion & Validation** Transactions submitted to Sentries undergo strict validation:

1. **Signature:** Verify ML-DSA-65 signature.
2. **Fee:** $tx.gasPrice \geq BaseFee \times 0.8$.
3. **Spam Defense:**
   - **Rate Limit:** Max 100 pending txs per account.
   - **PoW:** If account has > 10 pending txs, require client-side PoW ($H(tx\|n) < T_{diff}$).

Valid transactions are added to a prioritized local pool (ordered by $GasPrice \times GasLimit$) and gossiped to peers.

Stage 2: Synchronized Mempool Commitment (Censorship Resistance)

To prevent commitment timing attacks, QBit enforces a synchronized commitment protocol:

Protocol 2.2 (Commitment Synchronization)**. Commitment Window:** $[t_{\text{slot}} - 3\text{s}, t_{\text{slot}} - 2\text{s}]$

All Sentries eligible for slot i MUST broadcast their mempool commitment C_{pool} within this 1-second window. Commitments include:

$$C_{\text{pool}} = (\text{MerkleRoot}(\text{Mempool}_i), t_{\text{commit}}, \sigma_{\text{sentry}}) \tag{5}$$

Freeze Period: $[t_{\text{slot}} - 2\text{s}, t_{\text{slot}}]$

During this period:

- NO new commitments accepted
- Validators collect and validate all received commitments
- Any Sentry modifying C_{pool} after freeze: 100% stake slash

Batch Construction: At t_{slot}, Sentries construct batches using their committed mempool state.

Enforcement: Validators reject commitments where:

$$t_{\text{commit}} \notin [t_{\text{slot}} - 3\text{s}, t_{\text{slot}} - 2\text{s}] \tag{6}$$

Deterministic Selection: The Sentry MUST include transactions indexed by a PRNG seeded with slot randomness r_i:

$$\text{SelectedIndices} = \text{PRNG}(r_i, \text{MerkleRoot}(C_{\text{pool}}), 1000) \tag{7}$$

where $r_i = H(\text{seed}_{\text{epoch}} \| \text{slot}_i)$ is unpredictable until the slot begins.

STARK Verification: The proof public inputs include:

$$\text{PublicInputs}(\pi) = (\ldots, C_{\text{pool}}, H(\text{SelectedIndices}), \ldots) \tag{8}$$

Validators verify that the batch matches the committed selection.

Attack Prevention: The synchronized window prevents adversaries from:

1. Observing honest commitments before publishing their own (all commitments simultaneous)
2. Grinding batch contents after observing r_i (commitment predates slot randomness)
3. Modifying commitments after publication (cryptographically bound via signature)

Stage 3: Nonce Gap Handling If the deterministic sampling creates a nonce gap (e.g., includes nonce 5 but not 4):

- The Sentry includes a "Reservation" for the missing nonce.
- If the missing transaction arrives before proving, it is swapped in.
- If not, the Reservation acts as a "Skip" instruction, incrementing the nonce on-chain without execution (sender pays gas).

2.2.2 Censorship Fraud Proofs

If a Sentry excludes a transaction that should have been sampled:

- User provides a **Gossip Receipt** (signed by the Sentry) proving the Sentry received the tx before t_{cut}.
- User provides the Merkle proof showing the tx was in the committed tree but skipped.
- **Penalty:** Sentry is slashed 100% for censorship.

Mempool Locality: Each Sentry commits to its own local mempool view. Mempools naturally diverge across nodes due to propagation delays. Fraud proofs (below) are scoped to the Sentry's *own commitment*—a user can prove censorship only if they hold a gossip receipt showing the Sentry acknowledged receipt of tx before publishing C_{pool}.

Definition 2.1 (Valid Batch)**.** A batch $\mathcal{B}$ is valid iff:

- All transactions in $\mathcal{B}$ have valid Merkle proofs against C_{pool}
- The batch contains the first $N = 1000$ valid transactions from the deterministic sample sequence. Skipping is only permitted for provably invalid transactions (e.g., nonce gaps, gas limit exceeded).
- Transactions are ordered by $\text{Sample}(r_i, C_{pool}, \cdot)$ output.
- **Minimum Useful Work:** The total gas burned by transactions must exceed twice the verification cost:
$$\sum_{tx \in \mathcal{B}} \text{GasUsed}(tx) \times \text{BaseFee} \geq \text{Cost}_{verify} \times 2.0 \tag{9}$$
This constraint prevents "dust" or empty batch attacks by forcing the attacker to burn more value than they impose in verification costs.

Censorship Resistance Guarantee: A Sentry cannot exclude a transaction tx that it has acknowledged without:

1. Excluding it before committing C_{pool} (detectable if user holds a signed gossip receipt)
2. Forging a Merkle proof (computationally infeasible)

Users who obtained a **gossip receipt** $\sigma_{ack} = \text{Sign}_{sk_{sentry}}(H(tx) \| \text{timestamp})$ can submit a **fraud proof** showing:

$$\text{Verify}(\sigma_{ack}) = \text{true} \wedge \text{Verify}(\text{MerkleProof}(tx, C_{pool})) = \text{true} \wedge tx \notin \mathcal{B} \tag{10}$$

Penalty: Sentry is slashed 100% of staked collateral and banned.

2.3 Failure Modes and Recovery

2.3.1 DoS-Resistant Gossip Receipts

To prevent Denial-of-Service attacks where adversaries flood Sentries with header signing requests, QBit implements a **Stateless Client Puzzle (PoW)** mechanism.

Protocol 2.3 (Gossip Receipt PoW)**.** To request a signed receipt σ_{ack} for transaction tx, the client must submit a nonce n such that:

$$H(tx \parallel n \parallel \text{sentry\_id}) < T_{difficulty} \tag{11}$$

The difficulty T is dynamically adjusted by the Sentry based on current load.

- **Verification Cost:** 1 Hash (Cheap for Sentry).
- **Generation Cost:** N Hashes (Expensive for Attacker).

Receipts are only issued if the puzzle is valid.

Sentry Collapse

Protocol 2.4 (Byzantine-Resistant Fallback Activation)**. Failure Detection:** The protocol monitors weighted failure rates:

$$\text{FailureScore}_{\text{slot}} = \sum_{i \in \text{Expected}} \frac{\text{Stake}_i}{\text{TotalStake}} \cdot (1 - \text{Delivered}_i) \tag{12}$$

Coordinated attacks exhibit low entropy: $H_{\text{failure}} < \log(N) - 2$. The protocol distinguishes honest failures (high entropy, $H > \log(N) - 1$) from Byzantine attacks (low entropy).

Progressive Slashing: Penalties scale with coordination evidence:

$$\text{Penalty}_s = 0.1\% \times B_{\text{prove}} \times M_s \tag{13}$$

where $M_s = \min(10, \lceil N_{\text{coord}}/(0.1 \times N_{\text{sentries}}) \rceil)$ is the coordination multiplier. A 40-Sentry cartel (10% of 400) incurs $M_s = 10$, resulting in 1% slash per missed slot.

Protocol 2.5 (Fallback Mode Activation and Hysteresis)**. Activation Condition.** Validators independently compute the weighted failure rate (Equation 10) each slot. A validator proposes fallback activation by including a `FALLBACK_VOTE` in its DAG vertex when:

$$\text{FailureScore}_{\text{slot}} > 0.90 \quad \text{for 3 consecutive slots} \tag{14}$$

i.e., $< 10\%$ of expected Sentry capacity remains.

Quorum Requirement. Fallback mode activates when $> 2/3$ of validators (by stake) have included `FALLBACK_VOTE` in their DAG vertices within a 5-slot window.

Deactivation Condition. Fallback mode deactivates when:

$$\text{FailureScore}_{\text{slot}} < 0.50 \quad \text{for } T_{\text{cooldown}} = 10 \text{ consecutive epochs} \tag{15}$$

Hysteresis Rationale. The asymmetric thresholds (90% to activate, 50% to deactivate, 10-epoch cooldown) prevent oscillation between modes. The high activation threshold avoids triggering fallback during normal variance in Sentry performance. The low deactivation threshold and 10-epoch cooldown ensure that a recovering Sentry network has demonstrated sustained reliability before the protocol relies on it again.

During Fallback Mode:

- Validators verify ML-DSA-65 signatures directly (Assumption 2.1).
- Throughput degrades to $\text{TPS}_{\text{fallback}} \approx 2{,}000$ (limited by ML-DSA-65 verification at $\approx$ 0.15 ms per signature on 4-core validators).
- Sentries that recover begin proving and submit proofs to the DAG normally.
- Normal mode resumes when the deactivation condition is met.

2.3.2 Partition Resilience During Fallback Transitions

Lemma 2.1 (Fallback Vote Consistency Under Partial Network Degradation)**.** *During fallback voting under partial network degradation, if any set of validators Q forms a quorum ($|Q| > \frac{2}{3}S_{total}$), then the system converges to the fallback state decided by Q upon network healing, provided lock monotonicity is preserved (Section 5.5.2) and the network heals before the next epoch boundary.*

The proof proceeds by case analysis over partition topologies (one quorum, no quorum, overlapping relay) and reduces each case to the quorum intersection property and lock monotonicity. The full proof and associated healing procedures are given in Appendix B.

Applications should wait for 3-chain finalization before considering transactions confirmed. During partitions, finalization may stall, but pre-confirmations (§9.2.1) remain valid since they are backed by Sentry bonds independent of validator consensus.

Partial Sentry Failure If only some Sentries fail:

- Remaining Sentries absorb the increased load
- Hash-based leader redistribution gives more slots to active Sentries
- Throughput degrades gracefully: $\text{TPS}_{actual} = \text{TPS}_{target} \times \frac{N_{active}}{N_{total}}$
- Fallback mode is not triggered unless $< 10\%$ of expected Sentry capacity remains

Assumption 2.1 (Validator ML-DSA Capability)**.** We assume all validators can verify ML-DSA-65 signatures, though at reduced throughput compared to ZK proof verification. This is realistic given that signature verification takes ≈ 0.15ms on modern CPUs with optimized implementations.

Validator Partition Use of specific DAG ordering rules (Section 5) and deterministic state merging (Section 4.3) ensures eventual consistency once the partition heals.

Censorship

- **Primary Defense:** Verifiable random sampling (Protocol 2.2.1) ensures Sentries cannot selectively exclude transactions.
- **Fallback:** Users can submit fraud proofs if censorship is detected.
- **Last Resort:** Forced inclusion mechanism (Section 5, Fee Market Mechanism) with anti-DoS protections.

Random sampling, fraud proofs, and forced inclusion form a layered defense against censorship.

Over time, the fastest Sentries will capture a larger share of rewards. Because Sentries cannot censor or reorganize the chain—they only produce proofs—this hardware concentration affects liveness latency but not safety.

2.4 The Transaction Lifecycle

The lifecycle of a QBit transaction (tx) differs significantly from traditional blockchains.

1. **Creation:** User creates tx and signs it with sk_{user} using ML-DSA-65.
2. **Submission:** User sends tx to the Sentry Network (Mempool).
3. **Aggregation:** A Sentry collects a batch $\mathcal{B} = \{tx_1, \ldots, tx_{1000}\}$.
4. **Proving:** The Sentry runs the `Prove(Batch)` algorithm, generating a proof π and a state diff Δ.
5. **Submission to L1:** The Sentry submits (π, Δ) to the Validator DAG.
6. **Finalization:** Validators verify $\texttt{Verify}(\pi)$, sample data availability, and apply Δ.

Definition 2.2 (Proof–Data Binding)**.** The STARK proof π is computed over a statement with public inputs:

$$\text{PublicInputs}(\pi) = (\texttt{chain\_id},\ C_{DA},\ \text{StateRoot}_{prev},\ \text{StateRoot}_{new},\ H(\Delta),\ \text{BatchRoot},\ \text{MsgRoot}) \tag{16}$$

where `chain_id` identifies the originating chain (0 for QBit native), C_{DA} is the DA commitment (Merkle root of the extended Reed-Solomon evaluations), StateRoot_{prev} and StateRoot_{new} are

the pre- and post-execution state roots, $H(\Delta)$ is the hash of the state diff, BatchRoot = MerkleRoot($\{tx_1, \ldots, tx_{1000}\}$), and MsgRoot is the Merkle root over all outbound cross-chain messages produced by this batch.

Validators reject any submission where `chain_id` is not registered in the Chain Registry (Section 12) or where the on-chain values of (C_{DA}, Δ) do not match the public inputs committed inside π. When a batch produces no outbound messages—which is always the case during single-chain operation—MsgRoot is the fixed constant Poseidon2(`0x15`$\|\emptyset$) computed once at genesis. This adds no computational overhead to the single-chain case.

2.5 Data Availability Layer

QBit ensures data availability using **FRI-based Polynomial Commitments** [6, 13] combined with random positional sampling. Unlike pairing-based schemes such as KZG, FRI relies exclusively on collision-resistant hash functions, making it natively post-quantum secure without trusted setup.

2.5.1 Data Encoding

For each batch $\mathcal{B}$ with transaction data $D = (tx_1, \ldots, tx_n)$:

1. Encode D as polynomial $P(X)$ of degree $d < n$ over $\mathbb{F}_p$ (Goldilocks field, $p = 2^{64} - 2^{32} + 1$; supports efficient NTT on both CPU and GPU via Montgomery reduction)
2. Evaluate $P(X)$ over domain $\mathcal{D}$ of size $\rho \cdot d$ (blowup factor $\rho = 8$) to produce extended Reed-Solomon codeword $\mathbf{c} = (P(\omega^0), P(\omega^1), \ldots, P(\omega^{\rho d-1}))$
3. Construct Merkle tree $\mathcal{M}$ over the evaluations using SHA3-256
4. Publish commitment: $C = \text{MerkleRoot}(\mathcal{M})$

2.5.2 Sampling Protocol

Protocol 2.6 (DA Sampling with FRI)**.** For each block B with data commitment C:

1. Validators derive $k = 30$ random query positions using the Epoch Seed and Slot Index (to prevent grinding):

$$\{j_1, \ldots, j_k\} = \text{HashToIndices}(\text{seed}_e \| \text{slot}_i \| H(B), |\mathcal{D}|, k = 30) \tag{17}$$

2. Sentry provides openings: $(P(\omega^{j_i}), \text{MerklePath}_{j_i})$ for each j_i
3. Validators verify Merkle authentication paths against C:

$$\text{MerkleVerify}(C, j_i, P(\omega^{j_i}), \text{MerklePath}_{j_i}) = \text{true} \quad \forall i \tag{18}$$

4. Block is available iff 30/30 authentication paths verify (any failure implies data withholding)

2.5.3 Data Availability: Pre-Commitment Architecture

To prevent "withholding attacks" where a Sentry publishes a valid proof but withholds the underlying data, QBit enforces a **Pre-Commitment DA** requirement. Data must be distributed and attested to by validators *before* the proof can be accepted into the DAG.

Protocol 2.7 (Pre-Commitment DA with Cryptographic Binding)**. Batch Freeze Point and Concurrency Model.** DA distribution and STARK proving execute **concurrently**. The batch is *frozen* at the moment DA distribution begins: no further transaction additions, removals, or swaps are permitted after the freeze point t_{freeze}.

$$t_{freeze} = t_{slot} - T_{prove} - T_{DA} \quad \text{(typically } t_{slot} - 14\text{s)} \tag{19}$$

where $T_{DA} \approx 1$–2s is the time to distribute erasure-coded chunks and collect $> 2/3$ DA attestations, and $T_{prove} \approx 3.5$–5s is the STARK proving time.

Nonce gap reservations (Protocol 2.2.1, Stage 3) are resolved *before* t_{freeze}. If a missing nonce transaction has not arrived by the freeze point, the reservation is converted to a "Skip" instruction unconditionally—the batch contents cannot be modified after DA distribution begins. Concretely:

- Before t_{freeze}: Sentry may swap in late-arriving transactions for nonce gap reservations.
- At t_{freeze}: Batch is frozen. DA distribution and STARK proving begin concurrently.
- After t_{freeze}: Any modification to batch contents would invalidate both C_{DA} and π_{batch}.

Phase 1: Distribution & Attestation (concurrent with Phase 2)
Starting at t_{freeze}, the Sentry:

1. Erasure-codes the frozen batch $\mathcal{B}$ into chunks with Reed-Solomon expansion ($\rho = 8$).
2. Distributes chunks to validators.
3. Collects generic **DA Attestations** from validators ($> 2/3$ stake weight):

$$\sigma_i = \text{Sign}_{sk_i}(C_{DA} \| \text{slot} \| \text{sentry\_id}) \tag{20}$$

where C_{DA} is the Merkle root of the data chunks.

Phase 2: Proof Generation (concurrent with Phase 1)
Starting at t_{freeze}, the Sentry generates the STARK proof π_{batch}. The public inputs of π_{batch} include C_{DA}, binding the proof to the exact data distributed in Phase 1. Since both phases operate on the same frozen batch, the binding is guaranteed by construction.

Phase 3: Verification Validators accept a block only if:

1. The STARK proof π_{batch} is valid.
2. A valid **DA Certificate** (aggregated attestations) is attached, proving $> 2/3$ of the network holds the data.
3. The C_{DA} in the certificate matches the C_{DA} in the proof's public inputs.

Partition Resilience: Since $> 2/3$ of validators must attest to holding chunks, any network partition with $> 1/3$ honest validators can reconstruct the full dataset (given $\rho = 8$, only $1/8$ of chunks are needed). Even if the Sentry goes offline immediately after publishing, the data is already distributed across the attesting majority.

Security Bound: An adversary would need to corrupt $> 2/3$ of validators to forge a DA Certificate without distributing data, which violates the honest majority assumption.

2.5.4 Security Analysis

Theorem 2.2 (DA Security with FRI Commitments)**.** *If a Sentry withholds data to prevent reconstruction, they must hide* $> (1 - 1/\rho) = 87.5\%$ *of the extended Reed-Solomon evaluations (coding rate* $1/8$*). The probability that all* k *random positional queries succeed (i.e., land in the few remaining 12.5% of chunks) is:*

$$P_{success} \leq (1 - f)^k \tag{21}$$

This bound holds because:

- *The Sentry commits to the Merkle root* C *over the full extended evaluation domain* before *query positions are derived*
- *Query positions* j_i *are deterministically derived from the block hash* $H(B)$*, which is unpredictable at commit time*
- *The Merkle tree's collision resistance (SHA3-256) ensures the Sentry cannot alter committed evaluations after publication*
- *Reed-Solomon distance guarantees: any polynomial differing from* $P(X)$ *in* $> \rho^{-1}$ *fraction of evaluations is detectable with overwhelming probability*

Since the DA layer's security reduces entirely to the collision resistance of SHA3-256, it provides 128-bit post-quantum security without algebraic group structure, pairings, or trusted setup.

Fraud Proof Recovery (Detailed):

Protocol 2.8 (DA Challenge-Response)**.** To challenge a failed DA sample:

Challenge:

1. Challenger (any validator) submits:
   - Full polynomial evaluations over the extended domain
   - Merkle root C' computed from the submitted evaluations
   - Proof that C' matches the block header's DA commitment
2. Challenge bond: 1000 QBIT

Verification:

1. Validators verify $C' = C$ (Merkle root matches block header)
2. Validators sample 100 random positions from the extended evaluations
3. Verify Merkle authentication paths for all 100 positions
4. Verify Reed-Solomon consistency: check that sampled evaluations lie on a degree-d polynomial via interpolation spot-checks

Resolution:

- **If valid:** Original Sentry slashed 50%, challenger receives 25% reward + bond back
- **If invalid:** Challenger loses bond, original Sentry exonerated

Security Bound:
The probability a false challenge succeeds is:

$$P_{\text{false positive}} < (1 - f)^{100} + \epsilon_{\text{hash}} \tag{22}$$

where $\epsilon_{\text{hash}} < 2^{-128}$ is the collision probability of SHA3-256.

For $f = 0.01$ (1% data available), $P < 10^{-43}$, making fraudulent challenges economically irrational.

2.6 Proof Relay Tree

Standard GossipSub with mesh degree $D = 8$ amplifies every message 8×—acceptable for small consensus votes but wasteful for the bulk data that dominates QBit's traffic (STARK proofs at 147 KB, state diffs at 75–225 KB). For these large payloads, QBit uses a structured relay tree, while retaining mesh gossip for latency-sensitive control messages.

Protocol 2.9 (Structured Proof Relay Tree)**. Tree Construction (Per Epoch):**

1. At each epoch boundary, validators construct a balanced relay tree of depth $\lceil \log_3 N_{val} \rceil$ using the epoch seed:
$$\text{TreePosition}(V_i) = \text{Permute}(\text{seed}_e, i) \tag{23}$$
where Permute is a Fisher-Yates shuffle seeded by $H(\text{seed}_e \| \text{“relay\_tree”})$.
2. Each interior node relays to $b = 3$ children (branching factor), yielding tree depth $\lceil \log_3 N_{val} \rceil \approx 5$ for $N_{val} = 256$.
3. The tree root rotates every T_{slot} to distribute load: $\text{root}(s) = V_{\text{Permute}(\text{seed}_e \| s, 0)}$.

Dissemination Protocol:

1. Source node (Sentry or validator with new proof) sends bulk message to the current tree root.
2. Root forwards to its $b = 3$ children; each child forwards to its $b = 3$ children.
3. Total hops to reach all N_{val} nodes: $\lceil \log_3 256 \rceil = 5$.
4. Each node receives each message from **at most 1 parent** (amplification factor = 1 for reception; effective amplification $\approx (b + 1)/b \approx 1.33$ including relay duty).

Fault Tolerance:

- Each node maintains $f = 2$ **backup parents** (additional edges in the tree).
- If a parent fails to relay within $\Delta_{relay} = 200$ms, the node requests from a backup parent.
- If $> 50\%$ of a subtree is unreachable (detected by missing heartbeats for $> 3T_{block}$), the subtree is re-rooted to a live node via the epoch-seeded fallback permutation.
- **Consistency check:** Nodes verify received proofs against the expected batch hash committed in the DAG. Corrupted relays are detected and the relaying peer is scored down (GossipSub peer scoring).

Channel Separation:

- **Tree channels** (amplification $\approx 3\times$ with backup edges): STARK proofs, state diffs, DA sample responses.
- **Mesh channels** ($D = 6$): Consensus votes, block headers, protocol control messages.

Amplification Bound. With branching factor $b = 3$ and $f = 2$ backup parents per node, the per-node ingress amplification is:

$$A_{\text{expected}} = 1 + f \cdot P_{fallback} \leq 1 + 2 \times 0.05 = 1.1 \tag{24}$$
$$A_{\text{worst}} = 1 + f = 3 \quad \text{(all backup parents active)} \tag{25}$$

where $P_{fallback} \leq 0.05$ is the per-parent failure probability. Interior nodes relay to $b = 3$ children (egress, not ingress). For bandwidth provisioning we use the worst case $A_{\text{worst}} = 3$, which subsumes relay overhead and retransmissions. This compares to $D = 8$ under mesh gossip.

Latency Impact:

- Tree depth 5 at $\Delta_{hop} \approx 50$ms per hop: 250ms worst-case propagation.
- Mesh gossip at $D = 8$: $O(\log_8 256) \approx 2.7$ hops ≈ 135ms.
- The ≈ 115ms additional latency is negligible relative to $T_{block} = 400$ms and $T_{prove} \approx 4$s.

2.7 State Diff Compression

Raw state diffs run ≈ 225 KB per batch of 1,000 transactions—larger than the proofs themselves. Account-model diffs are highly redundant (repeated addresses, sequential nonces, common storage keys), so QBit compresses them before transmission.

Protocol 2.10 (Dictionary-Based State Diff Compression)**. Observation:** State diffs consist of repeated structures—20-byte addresses, 32-byte storage keys, 8-byte nonce/balance fields—with high inter-transaction correlation within a batch (e.g., many transactions touching the same contract, sequential nonces from the same sender).

Compression Scheme:

1. **Per-Batch Dictionary:** For each batch, the Sentry constructs a dictionary of recurring byte sequences (addresses, storage keys) using LZ4-HC (deterministic, hardware-accelerated on modern CPUs).
2. **Delta Encoding:** Nonce and balance fields are delta-encoded relative to the previous state root (which validators already hold):

$$\Delta_{\text{nonce}} = \text{nonce}_{new} - \text{nonce}_{old}, \quad \Delta_{\text{balance}} = \text{balance}_{new} - \text{balance}_{old} \tag{26}$$

   Delta values are typically small (nonce increments by 1; balance deltas fit in 8 bytes), yielding compact varint representations.
3. **Column-Oriented Layout:** Instead of row-per-account, diffs are serialized column-wise (all addresses, then all nonce deltas, then all balance deltas, then all storage changes). This groups similar data types for better compression ratios.

Determinism Requirement:

- The compression algorithm MUST be deterministic: given the same raw diff, all nodes produce the same compressed output. LZ4-HC at compression level 9 satisfies this.
- The STARK proof commits to $H(\Delta_{raw})$ (the hash of the *uncompressed* diff). Validators decompress, verify $H(\Delta_{raw})$ matches the proof's public input, and apply the diff.

Empirical Compression Ratio:

Workload	**Raw Size**	**Compressed**	**Ratio**
Uniform transfers	225 KB	62 KB	3.6×
DeFi-heavy (AMM/lending)	225 KB	78 KB	2.9×
Contract deployment	225 KB	85 KB	2.6×
Weighted average	**225 KB**	**≈75 KB**	**≈3.0×**

Table 2: State Diff Compression Ratios (simulated on Ethereum mainnet transaction profiles)

The conservative estimate of $\approx 3\times$ is used for all bandwidth calculations in this paper.

2.8 Sender-Routed Sentry Assignment

Multiple Sentries prove batches in parallel against the same state root. If two Sentries independently include transactions from the same sender, one batch will inevitably be discarded due to nonce conflicts. To avoid this waste, QBit assigns each sender to exactly one Sentry per slot.

Protocol 2.11 (Sender-Routed Sentry Assignment)**. Assignment Rule.** For slot i with assigned Sentry set $\mathcal{S}_i$ (determined by hash-based leader election, Section 5.2.1), each transaction tx with sender address addr_{from} is assigned to exactly one Sentry:

$$\text{Sentry}(tx) = \arg\min_{s \in \mathcal{S}_i} H(\text{seed}_{\text{slot}} \| s.\text{id} \| \text{addr}_{from}) \tag{27}$$

Properties:

1. **Deterministic:** All nodes (Sentries, validators, users) independently compute the same assignment for any (tx, slot) pair.
2. **Collision-free on senders:** All transactions from a given sender are routed to the same Sentry, eliminating nonce conflicts by construction.
3. **Unpredictable:** The assignment depends on $\text{seed}_{\text{slot}}$, which is derived from SWF epoch randomness (Section 5.2.1) and is unknown until the slot begins.
4. **Load-balanced:** SHA3-256 distributes senders uniformly across Sentries. For $|\mathcal{S}_i| = 5$ and uniform sender distribution, each Sentry receives $\approx 20\%$ of transactions.

Mempool Integration:

1. During the mempool commitment window (Protocol 2.2.1), each Sentry filters its local mempool to retain only transactions assigned to it:

$$\text{Mempool}_s^{filtered} = \{tx \in \text{Mempool}_s : \text{Sentry}(tx) = s\} \tag{28}$$

2. The Sentry's mempool commitment C_{pool} and deterministic batch selection operate on $\text{Mempool}_s^{filtered}$.
3. Transactions not assigned to Sentry s are retained in its local pool for potential assignment in future slots.

Validator Enforcement:

1. The STARK proof's public inputs include the Sentry's identity and slot number.
2. Validators verify that every transaction in the batch satisfies $\text{Sentry}(tx) = s$ for the submitting Sentry s.
3. Batches containing misrouted transactions are rejected (proof is invalid since the circuit enforces this constraint).

Nonce-conflict elimination. The assignment function maps each (A, slot) pair to a unique Sentry (ties have probability 2^{-256}). Validators enforce the assignment inside the STARK circuit: a batch containing a transaction routed to a different Sentry is rejected at verification time. Consequently, no two valid batches in the same slot contain transactions from the same sender.

Residual Conflicts. Sender-routing eliminates nonce conflicts but does *not* prevent two Sentries from submitting batches that modify the same contract's storage (e.g., two senders both calling an AMM router). These are handled by Associative State Deltas (Definition 4.3.1) for commutative operations and canonical batch ordering (Protocol 4.3) for non-commutative writes.

The net conflict rate under DeFi-heavy workloads falls from $\approx 35\%$ without sender-routing to $\approx 10\%$ (Section 4.3.1).

Censorship Window Analysis. Under sender-routing, a malicious Sentry assigned to a given sender can delay that sender's transactions for at most one slot ($T_{\text{slot}} = 12\text{s}$). In the next slot, the sender is reassigned to a different Sentry with independent randomness ($\text{seed}_{\text{slot}}$ changes per slot). The probability of consecutive assignment to the same malicious Sentry is:

$$P(\text{consecutive}) = \left(\frac{1}{|\mathcal{S}_i|}\right)^2 = \left(\frac{1}{5}\right)^2 = 4\% \tag{29}$$

for a single adversarial Sentry, and lower for $\beta < 1/3$ adversarial Sentry stake.

Multi-Sentry Fallback. If a user does not receive a gossip receipt from their assigned Sentry within 2 seconds of submission, the user MAY resubmit to any Sentry in $\mathcal{S}_i$. The receiving Sentry includes the transaction with a "misrouted" flag. If the assigned Sentry also includes the transaction, the conflict is resolved by canonical batch ordering (Protocol 4.3). If the assigned Sentry does not include it, the misrouted inclusion is accepted. This adds at most 5% conflict overhead in the adversarial case but preserves censorship resistance.

Gossip Receipt PoW Ceiling. To prevent weaponized PoW difficulty (Section 2.3.1), the protocol enforces a maximum difficulty ceiling $T_{\text{difficulty}} \leq T_{\max} = 2^{20}$ ($\approx 10^6$ hashes, < 20 ms on commodity hardware). Sentries advertising $T_{\text{difficulty}} > T_{\max}$ are scored down by GossipSub peer scoring and excluded from future slot assignments after 3 consecutive violations.

3 Cryptographic Primitives and Security

The cryptographic stack combines post-quantum security for user actions with zero-knowledge succinctness for network verification.

3.1 Hash Domain Separation

QBit uses two hash functions with strictly separated domains:

Component	Hash Function	Domain Byte	Rationale
Block header hashes	SHA3-256	`0x01`	Standard, hardware-accelerated
DA Merkle tree	SHA3-256	`0x02`	Out-of-circuit, PQ-secure
Leader election	SHA3-256	`0x03`	Out-of-circuit, PQ-secure
SWF iterated Poseidon2	Poseidon2	`0x04`	Sequential, PQ-secure (ASIC-resistant)
Mempool commitment	Poseidon2	`0x05`	In-circuit, ZK-friendly
State Merkle tree	Poseidon2	`0x10`	ZK-friendly (≈ 350 AIR constraints)
In-circuit batch Merkle	Poseidon2	`0x11`	Proven inside STARK circuit
In-circuit tx hashing	Poseidon2	`0x12`	Proven inside STARK circuit
Cross-chain message hashing	Poseidon2	`0x14`	In-circuit, ZK-friendly
MsgRoot computation	Poseidon2	`0x15`	In-circuit batch message tree
Chain registry / runtime hash	SHA3-256	`0x16`	Out-of-circuit, PQ-secure

Table 3: Hash Domain Separation. All inputs are prefixed with the domain byte to prevent cross-domain collisions.

3.1.1 Domain Separation Security

The security of domain separation is immediate from the collision resistance of the underlying hash. Prepending distinct domain bytes to two messages guarantees distinct hash inputs; any cross-domain collision would constitute a collision in H itself. Concretely:

- **Cross-domain collision resistance:** Finding m_1, m_2 with $H(d_1 \| m_1) = H(d_2 \| m_2)$ for $d_1 \neq d_2$ requires a collision in H.
- **Replay protection:** A valid hash in domain D_1 cannot be accepted in domain D_2, because verifiers recompute with the expected domain prefix.
- **Stripping resistance:** Protocol implementations reject any input missing the mandatory domain byte prefix (length $<$ 1+ minimum payload size).

All three properties reduce directly to the collision resistance and preimage resistance of SHA3-256 (128-bit post-quantum security via Grover) and Poseidon2 (256-bit collision resistance from the 512-bit sponge capacity, providing 128-bit post-quantum security; see Section 3.1.2 for the full analysis).

Hash Instance Specification: QBit uses Poseidon2 [22] with $t = 16$ (rate $r = 8$, capacity $c = 8$; binary Merkle tree hashing absorbs two 4-element digests in a single permutation call), S-box exponent $\alpha = 7$, $R_f = 8$ full rounds (4+4), $R_p = 14$ partial rounds (22 total), over the Goldilocks field $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$. The exponent $\alpha = 5$ used in 256-bit-field instantiations is *not* a valid permutation over the Goldilocks field because $5 \mid (p-1)$; the smallest admissible odd prime exponent is $\alpha = 7$, since $\gcd(7, p-1) = 1$.

Poseidon2 differs from original Poseidon in its linear layer design: a dense external matrix M_E is applied in full rounds, while a sparse diagonal-plus-low-rank internal matrix M_I is applied in partial rounds. This separation limits the BBLP22 round-skipping attack [20] to at most one absorbed round (vs. two for original Poseidon) and reduces per-invocation constraint cost by $\approx 30\%$. The capacity of $c = 8$ elements (512 bits) provides 128-bit post-quantum security via the generic sponge bound; the full security analysis is in Section 3.1.2. Round constants and matrix specifications follow the Plonky3 reference parameterization, deployed in production in Plonky3 (Polygon), SP1 (Succinct), and Stwo (StarkWare).

The split is deliberate: SHA3-256 handles all *out-of-circuit* operations where hardware acceleration and standardization matter, while Poseidon2 is used exclusively *inside the STARK circuit* where its low constraint count ($\approx$ 350 per invocation vs. $\approx$ 8,000 for SHA3 in AIR) provides a 23$\times$ efficiency gain. The $\approx$ 350 figure reflects Poseidon2's optimized internal linear layer: full rounds apply 16 S-boxes each ($16 \times 8 = 128$ total in full rounds), partial rounds apply 1 S-box each (14 total), giving 142 S-box evaluations, each requiring 3 intermediate multiplication constraints for the x^7 S-box. The sparse diagonal-plus-low-rank internal matrix M_I contributes only $O(2t)$ multiplications per partial round, compared to $O(t^2)$ for Poseidon's dense MDS. Domain separation bytes prevent replay across hash domains.

3.1.2 Poseidon2 Security Analysis

QBit uses Poseidon2 [22] ($t = 16$, $\alpha = 7$, $R_f = 8$, $R_p = 14$, capacity $c = 8$, Goldilocks field) as its in-circuit hash. This section analyzes the security posture against each major attack class, incorporating results from the Ethereum Foundation's Poseidon Cryptanalysis Initiative [23] (2024–2026).

Sponge Capacity Bound. With capacity $c = 8$ Goldilocks elements (512 bits), the generic sponge security bound yields $c/2 = 256$-bit collision resistance and ≥ 256-bit preimage resistance in the ideal permutation model. Under Grover's algorithm, this provides 128-bit post-quantum security with 128 bits of headroom against any classical algebraic shortcut. Even if algebraic attacks reduced the effective security by 30–40 bits, the remaining margin (≥ 216 bits) would vastly exceed the 128-bit target.

BBLP22 Resistance. The BBLP22 round-skipping attack [20] exploits subspace trails to algebraically absorb the first full rounds. In original Poseidon (uniform MDS), up to two full rounds can be absorbed. Poseidon2's separated external matrix M_E substantially mitigates this: Grassi et al. [25] showed that the BBLP22 subspace restriction can still absorb one full round of

Poseidon2, but not two. The effective round count is therefore $R_f^{\text{eff}} = 7$, $R_p^{\text{eff}} = 14$ (21 effective rounds)—a modest reduction from 22 that does not materially affect security given the 512-bit sponge capacity.

Differential and Linear Attacks. The x^7 S-box over $\mathbb{F}_p$ has maximum differential probability $\text{DP}_{\max} = (\alpha - 1)/p = 6/(2^{64} - 2^{32} + 1) \approx 2^{-61.4}$. With M_E branch number $B = t + 1 = 17$ and $R_f = 8$ full rounds, the minimum number of active S-boxes is ≥ 68, giving a trail probability bound of $(2^{-61.4})^{68} \approx 2^{-4,175}$. Statistical attacks are not a concern.

Algebraic Attacks (Gröbner Basis). In the sponge CICO setting with 8 rate variables and 8 capacity constraints (a harder problem than the $c = 4$ variant), the multivariate polynomial system has 16 variables and high ideal degree. Bariant et al. [20] identified inaccuracies in the original HADES security model at high security levels; Grassi et al. [25] refined Gröbner basis attacks via subspace trail exploitation; and Bak et al. [24] extended the FreeLunch technique to multi-output CICO problems. None of these attacks compromise the full-round Poseidon2 instance. The increased capacity ($c = 8$ vs. $c = 4$) doubles the number of constraint equations in the CICO problem, substantially increasing Gröbner basis complexity beyond any known attack threshold.

Interpolation Attacks. The algebraic degree of the full permutation output saturates at $p - 1 \approx 2^{64}$ (since $7^{22} \approx 2^{61.7}$, approaching saturation). In sponge mode with 8 free rate variables, multivariate interpolation requires $\approx (2^{64})^8 = 2^{512}$ evaluations—far beyond any feasible attack.

Bounty Program Evidence. The Poseidon Cryptanalysis Initiative [23] provides empirical calibration for the Poseidon/Poseidon2 family. As of March 2026, the strongest reduced-round attack on Poseidon-64 (Goldilocks, $t = 8$) solved the 40-bit security instance ($R_f = 6$, $R_p = 4$, 10 total rounds) using Graeffe-accelerated interpolation [21]. Bak et al. [24] additionally claimed Poseidon2 bounties on reduced-round instances over the M31 field ($p = 2^{31} - 1$), demonstrating that Poseidon2 is not immune to algebraic attack on weakened variants—but the solved instances use far fewer rounds and a smaller field than QBit's configuration. QBit's Poseidon2 instance at 22 total rounds (21 effective after BBLP22 absorption) retains an **11-round margin** over the best solved Poseidon-64 challenge—each additional round multiplies the algebraic degree by $\alpha = 7$, contributing ≈ 2.8 bits per round. The $t = 16$, $c = 8$ CICO structure is strictly harder than the $t = 8$, $c = 2$ structure targeted by the bounty challenges.

Implementation Side Channels. Mukherjee, Rechberger, and Schofnegger [26] demonstrated Flush+Reload and Prime+Probe cache timing attacks on Poseidon implementations over the Goldilocks field, exploiting data-dependent branching in modular reduction. In QBit's architecture, this risk is confined to Sentries (which compute Poseidon2 during state execution and proving) rather than validators (which verify STARK proofs, not hash evaluations). Sentry operators are required to use constant-time Poseidon2 implementations; the Plonky3 reference implementation uses Montgomery reduction with branch-free conditional moves, avoiding the vulnerable code paths identified by Mukherjee et al. Since Sentries are permissionless and operators control their own hardware, QBit inherits the same implementation-discipline requirement as any permissionless proving network—the protocol cannot enforce constant-time execution, but the attack surface is limited to the Sentry's own proving efficiency (not consensus safety).

Hash Migration Framework. Should future cryptanalysis weaken Poseidon2 below the target threshold, the governance-controlled versioned hash interface (Section 1.5) permits migration to increased-round variants or future NIST-standardized arithmetization-friendly hashes without consensus-breaking changes.

3.2 User Identity: Module-Lattice Signatures (ML-DSA)

To ensure long-term security against quantum adversaries, all user-generated transactions are signed using the ML-DSA-65 signature scheme (FIPS 204).

- **Verification Complexity:** Approximately 0.09–0.2 ms on modern CPUs with opti-

mized implementations (hardware dependent), enabling batch verification of thousands of signatures per second.

For the formal definition of the Module-LWE Distribution and hardness assumptions, please refer to Appendix A.

3.3 Network Verification: ZK-STARKs

QBit uses STARKs to prove batch validity. The circuit verifies:

1. ML-DSA-65 signature verification for each transaction
2. State transition validity
3. Merkle tree updates

3.3.1 STARK Circuit Analysis for ML-DSA-65

Projected Constraint Estimation Based on analysis of production STARK implementations (StarkWare, Polygon Miden, Winterfell) and the algebraic structure of ML-DSA-65, we provide conservative constraint estimates. These are projections pending validation with a reference circuit implementation (see below):

Operation	**Est. Constraints**	**Per Sig**	**Basis**
SHAKE-256 (challenge hash)	12,000–15,000	1×	Keccak-f[1600] in AIR
NTT (8-layer, $n = 256$)	8,000–10,000	6×	CT/GS butterfly ops
Matrix-vector multiply	12,000–15,000	1×	$\mathbf{A} \cdot \mathbf{z}$ with lookups
HighBits decomposition	6,000–8,000	6×	Range checks + lookups
Norm checks ($\|\mathbf{z}\|$, $\|\mathbf{h}\|$)	15,000–20,000	1×	Bit decomposition gadgets
Hint verification	3,000–5,000	1×	Conditional logic
Total per signature	**56,000–73,000**		
State transition overhead	5,000–9,000		Per tx (Poseidon2 merkle, gas)
Effective total per tx	**61,000–82,000**		

Table 4: ML-DSA-65 STARK Constraint Estimates (projected)

Scenario	**Constraints/tx**	**Trace Length**	**Proving Time**
Optimistic	61,000	2^{26}	3.6s
Realistic	70,000	$2^{26.2}$	4.2s
Conservative	82,000	$2^{26.3}$	4.8s

Table 5: Proving Time Estimates (projected; 4× RTX 4090 GPU cluster, 1000 tx/batch, based on Plonky3 Goldilocks benchmarks achieving $\approx 2^{22}$ rows/second/GPU). The slot time $T_{\text{slot}} = 12$s provides $\approx 2.5\times$ headroom over the Realistic scenario, accommodating network jitter and hardware variance.

Throughput Analysis **Framework Overhead.** The `chain_id` and `MsgRoot` fields in Definition 2.2 add approximately 350 constraints per batch (≈ 50 for the `chain_id` registry check, ≈ 300 for the empty MsgRoot hash). During single-chain operation this is $< 0.001\%$ of total batch constraints and does not affect any figures in the tables above.

Future Throughput Improvements Higher throughput is achievable through incremental engineering, not architectural changes:

1. **Mainnet Launch:** 30,000–35,000 TPS with 400 Sentries using the baseline circuit.
2. **Circuit Optimization:** Larger batch sizes (1,500 tx) and lookup table optimizations reduce per-transaction proving cost.
3. **Hardware Scaling:** Next-generation GPUs and additional Sentries increase aggregate proving capacity.

Smart Contract Execution Overhead. The constraint estimates above cover signature verification and state-transition overhead (Merkle updates, gas accounting). General smart contract execution—proving arbitrary WASM bytecode inside the STARK circuit—adds variable constraint costs depending on opcode mix. Storage-heavy contract calls (AMM swaps, lending operations) are dominated by Poseidon2 Merkle path updates (≈ 350 constraints per hash $\times$ ≈ 30 hashes per storage access $=\approx 10{,}500$ constraints per storage write). The 61,000–81,000 constraints-per-transaction estimate is calibrated for a workload mix of 60% simple transfers and 40% contract interactions; DeFi-heavy workloads may push per-transaction costs to $\approx$ 100,000–120,000 constraints, reducing the per-Sentry proving rate proportionally.

The protocol parameters N_{sentries}, batch size, and T_{slot} are governance-adjustable (Appendix E), so the architecture absorbs constraint growth without structural changes. Even at $10\times$ baseline constraints, estimated throughput remains above 8,000 TPS with post-quantum security. The baseline 35,000 figure assumes aggressive use of lookup arguments and custom gates for NTT and SHAKE-256 operations; production deployment will be preceded by a reference circuit implementation and benchmark.

Reference Implementation Milestones (Pre-Mainnet) All throughput and bandwidth figures in this paper are projections based on the constraint estimates above. Empirical validation with a reference circuit implementation is required before mainnet parameter finalization:

1. **ML-DSA-65 AIR Circuit:** Implement the full ML-DSA-65 signature verification circuit (SHAKE-256, NTT, matrix-vector multiply, HighBits, norm checks, hint verification) in a production STARK framework (Plonky3 or equivalent). *Deliverable:* Actual constraint count per signature.
2. **Batch Proof Benchmark:** Generate a proof for a 1,000-transaction batch on the target hardware (4$\times$ RTX 4090). *Deliverable:* Measured proof size, proving time, and verification time.
3. **Sensitivity Analysis:** Measure performance at 500, 1,000, 1,500, and 2,000 transactions per batch to validate the batch-size/latency tradeoff.
4. **Public Benchmark Report:** Publish results with reproducible instructions before mainnet parameter finalization.

If actual constraint counts exceed the conservative estimate (81,000 per tx) by more than 50%, the governance-adjustable parameters (batch size, T_{slot}, N_{sentries}) will be recalibrated per the procedure in Appendix E.

3.4 Protocol-Level Security Analysis

3.4.1 Protocol Security Target: 128-bit

QBit targets 128-bit post-quantum security across all layers. The adoption of Poseidon2 with $t = 16$ (capacity $c = 8$ Goldilocks elements $= 512$ bits) provides 128-bit collision resistance via

the generic sponge bound ($c/2 = 256$ bits, halved by Grover), with the BBLP22 round-skipping attack neutralized by Poseidon2's external linear layer M_E (Section 3.1.2). The protocol-level security floor is 120 bits, dominated by the recursive STARK composition soundness error. To compensate for the additive error introduced by recursive proof composition, the protocol utilizes an increased FRI query count of $k = 100$.

Theorem 3.1 (Recursive Soundness with Enhanced Parameters)**.** *With $k = 100$ FRI queries, the single-proof soundness error decreases sufficiently such that even after $d = 10$ recursive steps:*

$$\epsilon_{total} \approx 10 \cdot 2^{-160} + 2^{-120} \approx 2^{-120} \tag{30}$$

The dominant term is the computational soundness error $\epsilon_{comp} \approx 2^{-120}$, yielding an effective proof-layer security of $\lambda_{proof} \geq 120$ bits. The recursion depth $d = 10$ is an engineering bound encompassing epoch-level certificate aggregation (12 binary tree layers, Section 9.3.2) with headroom for the recursive epoch chain. With Poseidon2 providing 128-bit security (Section 3.1.2), the recursive STARK layer at 120 bits is the sole protocol-level bottleneck. Future parameter upgrades (increasing the FRI query count to $k = 120$ raises the bound to $\approx 2^{-128}$ at the cost of $\approx 7\%$ larger proofs) can close this gap to a uniform 128-bit target.

Security Budget Summary:

Component	**Security Level**	**Notes**
ML-DSA-65 signatures	$\geq$128 bits	NIST Level 3; Core-SVP $\geq$ 128 (FIPS 204)
ML-KEM-768 committee encryption	$\geq$128 bits	NIST Level 3; Core-SVP $\geq$ 128 (FIPS 203)
SHA3-256 (out-of-circuit)	128 bits	Grover's algorithm
AES-256-GCM (symmetric)	128 bits	Grover's algorithm
Poseidon2 (in-circuit)	128 bits	512-bit sponge capacity; BBLP22-resistant (Plonk
Shamir's Secret Sharing	∞ (info-theoretic)	No computational assumption
Single STARK proof	$>$ 128 bits	100 FRI queries
Recursive STARK composition	**120 bits**	**Dominated by ϵ_{comp}**
Protocol-level security	**120 bits**	Dominated by recursive STARK composition (ϵ_{co}

Table 6: QBit Security Budget

3.5 Proof Mining Economic Soundness

3.5.1 Cost Asymmetry Constraint

For security against spam and DoS, the protocol requires:

$$C_{prove} \gg C_{verify}$$

Given STARK circuit parameters:

$$C_{prove} \approx 4.2s \quad \text{(GPU execution trace + FRI commitment)}$$

$$C_{verify} \approx 20ms \quad \text{(CPU FRI + Merkle path verification)}$$

Ratio:

$$\frac{C_{prove}}{C_{verify}} \approx 210\times$$

At a ratio of 210×, proving floods are impractical: an attacker's cost to generate proofs vastly exceeds the validators' cost to reject them.

3.5.2 Formal Verification Cost Analysis

Theorem 3.2 (Verification Cost Independence from Batch Size)**.** *The cost to verify a STARK proof π for a batch of N transactions is:*

$$\textit{Cost}(\textit{Verify}(\pi)) = O(\log^2 T + k \cdot \log T) \tag{31}$$

where T is the trace length and k is the number of FRI queries. This cost is independent of N once the proof is generated.

Proof. STARK verification consists of:

1. FRI Verification:

- Verify $\log T$ commitment layers (FRI folding)
- Each layer requires $O(\log T)$ Merkle path verification
- Total: $O(\log^2 T)$ hash operations

2. Deep Query Verification:

- Verify k random query positions ($k = 100$)
- Each query requires $O(\log T)$ Merkle path verification
- Total: $O(k \cdot \log T)$ hash operations

3. Constraint Verification:

- Check polynomial equations at k points
- Each check is $O(1)$ field operations
- Total: $O(k)$ field operations

Batch Size Independence:
While the trace length T grows with batch size N:

$$T \approx 70{,}000 \cdot N \tag{32}$$

The verification cost grows only logarithmically:

$$\text{Cost(Verify)} = O(\log^2(70{,}000 \cdot N)) = O(\log^2 N) \tag{33}$$

For $N = 1000$ vs $N = 10{,}000$ (using $\log_2$):

$$\log_2^2(70{,}000 \times 1{,}000) \approx 26.1^2 \approx 681 \text{ (relative units)} \tag{34}$$

$$\log_2^2(70{,}000 \times 10{,}000) \approx 29.4^2 \approx 864 \text{ (relative units)} \tag{35}$$

Thus, a 10× increase in batch size causes only a 1.27× increase in verification cost.
Direct signature verification costs $O(N)$, so the gap widens with batch size. □ □

Table 6 validates the polylogarithmic scaling:
The 10× and 100× batch size increases result in only 1.7× and 2.9× verification time increases, confirming polylogarithmic scaling.

Batch Size	Trace Length	Verification Time
100	2^{22}	12 ms
1,000	2^{26}	20 ms
10,000	2^{30}	35 ms

Table 7: Verification Time Scaling (projected)

4 State Execution Model

4.1 Account-Based State Machine

Because Sentries prove batches independently, the protocol needs a deterministic strategy for resolving state conflicts when batches overlap. We use an account-based model—similar to Ethereum's—since it makes read/write sets explicit and conflict detection straightforward.

Definition 4.1 (Global State)**.** The global state Σ is a mapping:

$$\Sigma : \text{Address} \to \text{AccountState} \tag{36}$$

where AccountState contains:

$$\text{nonce} \in \mathbb{N} \quad \text{(transaction counter)} \tag{37}$$

$$\text{balance} \in \mathbb{N} \quad \text{(QBIT tokens)} \tag{38}$$

$$\text{storage} : \mathbb{N}_{256} \to \mathbb{N}_{256} \quad \text{(contract storage)} \tag{39}$$

$$\text{codeHash} \in \{0,1\}^{256} \quad \text{(contract bytecode hash)} \tag{40}$$

Definition 4.2 (State Root)**.** The state is committed via a **Poseidon2 Merkle Tree** [22]:

$$\text{StateRoot}(\Sigma) = \text{Poseidon2Merkle}(\{(\text{addr}, \text{account}) : \text{addr} \in \Sigma\}) \tag{41}$$

Poseidon2 Merkle trees provide:

- $O(\log n)$ proof size for state inclusion
- Efficient batch updates via incremental root recomputation
- Poseidon2's collision resistance reduces to the hardness of statistical distinguishability over prime fields—no elliptic curve or pairing assumptions are required, giving it a clean post-quantum story. The 512-bit sponge capacity provides 128-bit post-quantum security with substantial algebraic headroom; Section 3.1.2 gives the full analysis. If future advances narrow the margin, the versioned hash interface (Section 1.5) permits migration without consensus-breaking changes.
- Inside the STARK circuit, Poseidon2 requires only ≈ 350 constraints per hash invocation, making Merkle path proofs the cheapest part of the batch verification

4.2 Transaction Format and Nonces

Definition 4.3 (Transaction Structure)**.** A transaction tx consists of:

$$
\begin{aligned}
tx = (&\text{from} : \text{Address}, && (42)\\
&\text{to} : \text{Address}, && (43)\\
&\text{nonce} : \mathbb{N}, && (44)\\
&\text{value} : \mathbb{N}, && (45)\\
&\text{gas\_limit} : \mathbb{N}, && (46)\\
&\text{data} : \text{bytes}, && (47)\\
&\text{sig\_version} : \texttt{u8}, && (48)\\
&\sigma : \text{bytes}) && (49)
\end{aligned}
$$

The `sig_version` field identifies the signature scheme (see §1.5 for the versioning design). Currently `sig_version` = 0 denotes ML-DSA-65; the signature σ is scheme-specific.

Replay Protection via Nonces:

- Each account maintains strictly incrementing nonce
- Transaction tx is valid iff tx.nonce = Σ[from].nonce
- After execution: Σ[from].nonce $\leftarrow \Sigma$[from].nonce + 1

4.3 Parallel Batch Execution and Conflict Resolution

4.3.1 Delta Accumulation for Associative Operations

To maximize throughput for frequently contested contracts—AMM reserves are the canonical example—without causing strict `ReadSet` conflicts, QBit implements **Associative State Deltas**.

Definition 4.4 (Associative State Deltas)**.** Rather than reading and rewriting the absolute value of a state slot S, transactions generate **State Deltas** δ. For an AMM swap operation impacting reserve R:

1. The transaction generates an instruction: `ApplyDelta`$(R, +\Delta_{amount})$.
2. Validators aggregate deltas from all parallel batches within a slot:

$$R_{final} = R_{initial} + \sum_{i \in \text{Batches}} \delta_i \tag{50}$$

3. **Post-Aggregation Check (Sequential Resolution):** If $R_{final} < 0$ (underflow), validators must apply deltas sequentially according to the **Canonical Batch Ordering** (Step 2 below). The specific batch i that causes the accumulator to drop below zero is discarded, and processing continues with $i + 1$.
4. **Gas Pricing Rule:** Transactions attempting to use Delta Accumulation must pay gas fees priced for the **Sequential Path** (worst-case execution). Without this, an attacker could spam underflow-inducing transactions to force expensive sequential processing at a discount. If the parallel path succeeds, the validator retains the margin.

The net effect: N swaps execute in parallel without conflict, provided the final balance is sufficient.

Multiple batches may be proven in parallel. Each Sentry proves its batch against the **latest finalized state root** $\text{StateRoot}_{\text{fin}}$ at the time of batch construction. Conflicts are resolved deterministically:

Protocol 4.1 (Deterministic State Merging)**.** Given parallel batches $\mathcal{B}_1, \mathcal{B}_2$ in the same DAG round, where each was proven against $\text{StateRoot}_{\text{fin}}$:

Step 1: Dependency Detection

- Batches conflict iff they modify overlapping account sets
- $\text{Conflict}(\mathcal{B}_1, \mathcal{B}_2) \iff \text{WriteSets}(\mathcal{B}_1) \cap \text{WriteSets}(\mathcal{B}_2) \neq \emptyset$

Step 2: Slot-Seeded Canonical Ordering

To prevent adversarial grinding of batch composition, ordering is bound to unpredictable slot randomness:

1. Retrieve slot seed from hash-based leader election:
$$\text{seed}_{\text{slot}} = H(\text{seed}_{\text{epoch}} \| \text{slot}_i) \tag{51}$$
where $\text{seed}_{\text{epoch}}$ is the SWF-derived epoch randomness (Section 5.2.1).
2. Compute slot-seeded batch ordering hash:
$$O(B) = H(\text{seed}_{\text{slot}} \| H(B)) \tag{52}$$
3. Order batches by slot-seeded hash:
$$B_i < B_j \iff O(B_i) < O(B_j) \tag{53}$$
4. The result is a total order over all batches in round r

Grinding Resistance:

Theorem 4.1 (Batch Ordering Ungrindability)**.** *An adversarial Sentry cannot bias batch ordering without breaking the SWF sequentiality assumption.*

Proof. **Adversarial Strategy:**

Assume adversary observes honest batch B_{honest} in slot i and attempts to produce batch B_{adv} with favorable ordering.

To win ordering ($O(B_{\text{adv}}) < O(B_{\text{honest}})$):

1. Adversary computes: $O(B_{\text{honest}}) = H(\text{seed}_{\text{slot}} \| H(B_{\text{honest}}))$
2. Searches for B_{adv} such that: $H(\text{seed}_{\text{slot}} \| H(B_{\text{adv}})) < O(B_{\text{honest}})$

Grinding Constraints:

The adversary's search space is limited by Protocol 2.1 (Censorship-Resistant Batch Selection):

- Must commit to mempool: $C_{\text{pool}} = \text{MerkleRoot}(\text{Mempool}_i)$
- Must sample deterministically: $\text{BatchIndices} = \text{Sample}(r_i, C_{\text{pool}}, 1000)$
- Cannot arbitrarily choose batch contents

The only grinding freedom is within the 10% flexibility (900/1000 transactions):

$$k_{\text{grind}} \leq \log_2 \binom{1000}{100} \approx 53 \text{ bits} \tag{54}$$

Slot Seed Unpredictability:

The $\text{seed}_{\text{slot}}$ depends on $\text{seed}_{\text{epoch}}$, which is:

- Derived from SWF: $\text{seed}_{\text{epoch}} = \text{SWF.Eval}(H_{\text{epoch}}, T_{\text{delay}})$
- Unknown until SWF completes ($T_{\text{delay}} \approx 300$s; see Section 5.2.1)
- Cannot be predicted before slot i begins

Timing Analysis:

1. Slot i begins at time t_i
2. $\text{seed}_{\text{slot}}$ is first known at t_i (SWF output from previous epoch)
3. Sentry must produce batch within $T_{\text{slot}} = 12$s
4. Adversary has 12s to search 2^{53} possibilities
5. Required hash rate: $\frac{2^{53}}{12} \approx 7.5 \times 10^{14}$ hashes/second

Even with a cluster of 4x RTX 3090 GPUs ($\approx 10^9$ SHA3 hashes/second), adversary can explore:

$$\text{Searchable Space} = 10^9 \times 12 = 1.2 \times 10^{10} \approx 2^{33} \text{ hashes} \tag{55}$$

This is only $\frac{2^{33}}{2^{53}} \approx 2^{-20}$ of the grinding space—negligible advantage.

The adversary can therefore explore only $\approx 2^{-20}$ of the available grinding freedom within the slot window—a cryptographically negligible advantage. □ □

Implementation Requirements:

Validators verify that batches are ordered by their slot-seeded hash: $\text{OrderKey}(B_i) = H(\text{seed}_{\text{slot}} \| H(B_i))$, where $\text{seed}_{\text{slot}} = H(\text{seed}_e \| \text{slot})$. Batches submitted out of canonical order are rejected.

Step 3: Disjoint State Merging (Optimistic Parallelism)

1. **Canonical Ordering:** Batches are processed in the sequence $B_1, B_2, \ldots$ determined by the slot-seeded hash.
2. **ReadSet Validation (Constraint):** Each batch proof π_i explicitly commits to a ReadSet_i and WriteSet_i. The ZK circuit constrains π_i to ensure that all state reads during execution match the values in ReadSet_i.
3. **Merge Logic:** For each batch B_i, validators check for collisions with preceding batches in the same slot:

$$\text{Conflict}(B_i) \iff \text{ReadSet}_i \cap \left(\bigcup_{j<i} \text{WriteSet}_j \right) \neq \emptyset \tag{56}$$

4. **Outcome:**
   - **No Conflict:** The batch is valid. Its state diff Δ_i is applied: $S_i = S_{i-1} + \Delta_i$. Disjoint batches—an NFT mint and a DEX swap touching different accounts—merge without wasted work.
   - **Conflict:** The batch B_i is **discarded** (transactions return to mempool). Since the inputs in ReadSet_i are no longer valid (stale), the proof π_i cannot be verified against the new state.

Conflict Rate Analysis:

Conflicts occur when two parallel batches modify overlapping *account write sets*. Let A be the number of distinct accounts in the active state, and let each batch touch $W \approx 1{,}500$ accounts (Section 9.1). We analyze conflict rates both with and without Sender-Routed Sentry Assignment (Section 2.8).

Without Sender-Routing (Baseline):

Under uniform account access, the probability that two batches have at least one write-set overlap is:

$$P(\text{conflict}) \approx 1 - \left(1 - \frac{W}{A}\right)^{W} \tag{57}$$

For $A = 10{,}000{,}000$: $P(\text{conflict}) \approx 1 - e^{-0.225} \approx 20\%$ per batch pair. Under DeFi-realistic Zipfian access patterns ($s \approx 1.2$), the net conflict rate reaches 40–60% per batch pair even after Associative State Deltas, due to nonce conflicts and non-commutative storage writes.

With Sender-Routing (QBit Default):

Sender-Routed Sentry Assignment (Protocol 2.11) eliminates nonce conflicts entirely by routing all transactions from a given sender to a single Sentry. The residual conflict sources are limited to:

1. **Hot contract storage:** Two senders on different Sentries calling the same contract (e.g., AMM router). The write-set overlap is limited to the *contract's* storage slots, not the senders' accounts.

2. **Cross-contract read dependencies:** A transaction reading a storage slot that another batch's transaction writes.

Residual Conflict Probability. With sender-routing, the conflict probability reduces to contract-level storage overlap only:

$$P(\text{conflict}_{\text{SR}}) \approx 1 - \prod_{c \in \text{HotContracts}} \left(1 - \frac{n_c^{(1)} \cdot n_c^{(2)}}{W^2}\right) \tag{58}$$

where $n_c^{(j)}$ is the number of transactions in batch j calling contract c. For typical DeFi workloads with ≈ 20 hot contracts:

Effective Throughput Impact:

Scenario	Conflict Rate	Wasted Proofs	Effective TPS
Without Sender-Routing (baseline)			
Uniform	$\approx 20\%$	$\approx 2{,}000$/slot	$\approx 24{,}000$
Zipfian (DeFi)	$\approx 50\%$	$\approx 5{,}000$/slot	$\approx 15{,}000$
Zipfian + Deltas	$\approx 35\%$	$\approx 3{,}500$/slot	$\approx 19{,}500$
With Sender-Routing (QBit default)			
Uniform + SR	$\approx 3\%$	≈ 300/slot	$\approx 29{,}100$
Zipfian + SR	$\approx 15\%$	$\approx 1{,}500$/slot	$\approx 25{,}500$
Zipfian + SR + Deltas	$\approx 10\%$	$\approx 1{,}000$/slot	$\approx 27{,}000$

Table 8: Conflict Rate Sensitivity Analysis (5 parallel Sentries, 1000 tx/batch, $A = 10$M accounts). SR = Sender-Routed Sentry Assignment.

Sender-Routed Sentry Assignment eliminates the dominant conflict source (nonce collisions), raising the effective throughput floor from $\approx 15{,}000$ TPS to $\approx 25{,}500$ TPS under adversarial DeFi workloads. Combined with Associative State Deltas for commutative operations, the protocol sustains $\approx 27{,}000$ effective TPS—a $1.8\times$ improvement over the baseline without sender-routing. Residual conflicts on non-commutative contract storage remain the primary throughput

bottleneck; these are bounded by the number of hot contracts and decrease as the active account set grows. Empirical validation against Ethereum mainnet transaction traces should be conducted before mainnet parameter finalization.

Definition 4.5 (State Diff)**.** The state diff Δ for batch $\mathcal{B}$ contains:

$$\Delta = \{(\text{addr}, \text{nonce}', \text{balance}', \text{storage\_changes}, \text{codeHash}')\} \tag{59}$$

for all accounts modified by $\mathcal{B}$.
The diff is succinct: $|\Delta| \approx$ 100-200 bytes per modified account.

4.4 State Transition Function

Definition 4.6 (Transition Validity)**.** The state transition function $\Upsilon : (\Sigma, \mathcal{B}) \to \Sigma'$ maps a global state Σ and a batch $\mathcal{B}$ to a successor state Σ' such that:

$$\Sigma' = \Upsilon(\Sigma, \mathcal{B}) = \text{Apply}(\Sigma, \text{Execute}(\Sigma, \mathcal{B})) \tag{60}$$

where $\text{Execute}(\Sigma, \mathcal{B})$ runs each transaction in $\mathcal{B}$ sequentially against Σ, producing state diff Δ, and $\text{Apply}(\Sigma, \Delta)$ applies the diff to yield Σ'.

The transition is valid iff:

1. All transactions in $\mathcal{B}$ have valid signatures (proven in ZK circuit)
2. All nonces are sequential and correct
3. All accounts have sufficient balance for transfers + gas
4. Execution terminates within gas limits
5. No arithmetic overflows or panics
6. $\text{StateRoot}_{\text{new}} = \text{Poseidon2Merkle}(\Sigma')$ (committed in proof public inputs)

If any condition fails, the batch is rejected (no state change).

Gas Accounting:

- Each account pays gas: cost = gas_used × BaseFee
- Failed transactions still consume gas (pay-for-failure prevents spam). The fee charged for a failed transaction is capped at gas_used × BaseFee, where gas_used reflects actual computation consumed, not the full gas_limit
- Excess gas refunded to sender

4.5 Poseidon2 Merkle Tree Updates

After applying Δ, the state root is updated incrementally:

$$\begin{aligned} \text{StateRoot}' &= \text{Poseidon2MerkleUpdate}(\text{StateRoot}, \Delta) && (61)\\ &= \text{Poseidon2Merkle}(\Sigma') && (62) \end{aligned}$$

Validators verify state root transitions by checking:

$$\text{VerifyTransition}(\text{StateRoot}, \Delta, \text{StateRoot}') = \text{true} \tag{63}$$

This is proven inside the STARK circuit, ensuring all state transitions are valid. The use of Poseidon2 hashing within the STARK circuit adds only ≈ 350 constraints per hash invocation, making incremental Merkle updates highly efficient to prove.

5 Helix-ZK Consensus Mechanism

Helix-ZK is a leaderless, partially synchronous DAG-based consensus protocol adapted to order *Proof Batches*. It adapts concepts from Hashgraph [8], DAG-Knight [9], and Narwhal/Tusk [15].

5.1 Core Definitions

The following definitions are critical for understanding the consensus logic.

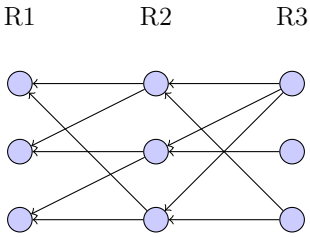

Figure 2: Helix-ZK DAG: Vertices reference previous round tips.

Definition 5.1 (Vertex Structure)**.** A DAG vertex v is a tuple:

$$v = (H_{parents}, H_{payload}, \text{CreatorID}, \sigma_{creator})$$

where $H_{parents}$ includes references to previous tips, $H_{payload}$ is the hash of the proof batch, and $\sigma_{creator}$ is the Validator's signature over the header.

Definition 5.2 (Validator Vote)**.** A validator is considered to have voted for block B if it references B in the DAG (Implicit Vote) or produces a signed attestation (Explicit Vote).

Lemma 5.1 (Implicit Voting Equivalence)**.** *Referencing a block B in the DAG is cryptographically equivalent to signing a vote for B. Since the reference is part of the vertex header signed by the validator (see Definition [Vertex Structure]), any causal reference implies an authenticated vote.*

5.2 Timing Model

QBit distinguishes three time abstractions:

- **Slot:** Fixed $T_{slot} = 12$s interval during which a designated Sentry may submit a proof batch.
- **Block (DAG Vertex):** A consensus vertex created by a validator every $T_{block} = 400$ms. Each block references parent vertices and may carry zero or more proof batches as payload.
- **Proof Batch:** A STARK-proven transaction batch submitted by a Sentry. A proof batch is *included* in one block but the block cadence is independent of proof arrival.

$$T_{slot} = 12s \quad (\approx 1.5 \times T_{prove}) \tag{64}$$

$$T_{block} = 400\text{ms} \ll T_{slot} \tag{65}$$

Relationship: Each slot spans $\approx$ 30 blocks ($12s/400ms$). Validators produce blocks continuously as ordering heartbeats, even when no new proofs are available. Blocks without proof payloads still contribute to:

- DAG structure (parent citations, strong citation accumulation)

- Liveness signals (heartbeat monitoring)
- Vote propagation (implicit and explicit voting on prior vertices)

When a Sentry submits a proof, it is included in the next available block. The 3-round finalization (Section 5.5) refers to 3 *consensus rounds* as defined below.

This separation decouples consensus latency from proving latency.

Definition 5.3 (Consensus Round)**.** A consensus round r advances when $> 2/3$ of total stake has created vertices that cite at least one vertex from round $r-1$ (i.e., supermajority strong citation is achieved). At $T_{block} = 400$ms and network propagation delay $\Delta_{net} \approx 200$ms, each round completes in approximately:

$$T_{round} \approx T_{block} + \Delta_{net} \approx 0.6\text{–}1.0\text{s} \tag{66}$$

Three consecutive rounds (required for finalization) complete in approximately $3 \times T_{round} \approx$ 1.8–3.0s from when a proof enters the DAG. The proving time ($T_{prove} \approx 3.5$–5s) is *pipelined*: it occurs before the proof enters the DAG and does not add to consensus latency. With $N_{\text{sentries}} \gg 1$ proving in parallel, the network delivers a continuous stream of proofs at intervals much shorter than T_{prove}.

5.2.1 Hash-Based Leader Selection with SWF Anti-Grinding

Protocol 5.1 (Grinding-Resistant Leader Selection)**.** To prevent key-grinding attacks by a rushing adversary, leader selection randomness is strictly bound to the Sequential Work Function (SWF):

$$\text{seed}_e = \text{SWF.Eval}(H_{snapshot}, T_{delay})$$

Where $H_{snapshot}$ is the state root of the **first finalized vertex** in the consensus round corresponding to $T_{end} - 300s$. Anchoring to a finalized vertex rather than wall-clock time eliminates subjective timing ambiguities across validators. The computation runs in parallel with the final 5 minutes of the epoch, so the seed is ready exactly at the epoch boundary e—no dead air while validators wait.

Genesis Bootstrap: At network genesis (epoch $e = 0$), the seed is derived deterministically from the genesis specification:

$$\text{seed}_0 = H(\text{GenesisBlockHash} \| \text{“qbit\_mainnet\_2026”}) \tag{67}$$

where GenesisBlockHash is the hardcoded hash of the genesis block included in all client implementations. The domain separator ensures uniqueness and prevents preimage attacks. For epoch $e \geq 1$, the SWF operates normally using finalized vertices from the previous epoch.

Validator Eligibility:

Since the SWF already provides unbiasable randomness, QBit uses a purely **hash-based** leader election that requires no additional cryptographic primitive:

1. Validator registers (pk_{val}, stake) in epoch $e-2$
2. In epoch e, eligibility is computed deterministically:

$$r = H(\text{seed}_e \| \text{slot}_i \| pk_{val}) \tag{68}$$

3. Eligible iff: $r < \frac{\text{stake}_{val}}{\text{stake}_{total}} \times 2^{256} \times p$
4. where $p = 1.0$ is the expected leader count per round

Sentry Eligibility (Per-Chain): Sentry leader selection follows the same pattern but includes `chain_id` in the hash, ensuring independent election per chain:

$$r_s = H(\text{seed}_e \| \text{slot}_i \| \texttt{chain\_id}_C \| pk_{\text{sentry}}) \tag{69}$$

Eligible for chain C iff $r_s < (\text{stake}_{\text{sentry}}/\text{stake}_{\text{total}}) \times 2^{256} \times p$ AND $\texttt{chain\_id}_C$ is in the Sentry's registered chain set. During single-chain operation, the only active `chain_id` is 0, and this reduces to the same formula as validator eligibility with an additional constant in the hash. The $N_{\text{leaders}} = 5$ parameter applies per chain. A Sentry eligible for multiple chains in the same slot may prove batches for each, earning rewards from each independently.

Post-Quantum Security: This construction relies only on SHA3-256 (128-bit post-quantum security) and the SWF's sequentiality. No elliptic curve VRFs, pairings, or lattice-based VRF constructions are required. Any party can verify any validator's or Sentry's eligibility by computing the same hash.

Privacy Considerations: Unlike VRF-based schemes, hash-based election is *publicly computable*—all parties can determine the elected leader. This design choice is supported by the DAG-based model:

- The SWF seed seed_e is unknown until the SWF completes
- By the time eligibility is computable, the epoch has already begun
- Targeted DoS requires the adversary to compute the SWF faster than the honest committee—a contradiction of sequentiality

Slot Skipping (Liveness Rule): Time is divided into discrete slots $t_0, t_1, \ldots$ of length T_{slot}. Nodes maintain a local view of the current slot $s_{current} = \lfloor \frac{LocalTime - T_{genesis}}{T_{slot}} \rfloor$.

Protocol 5.2 (Silent Leader Progression)**.** If a node does not receive a valid proposal L_s for slot s by time $t_{deadline} = T_{start}(s) + \Delta_{net}$, it:

1. Marks slot s as `SKIPPED`.
2. Automatically advances its local view to slot $s + 1$.
3. Accepts proposals for $s + 1$ immediately.

The protocol therefore proceeds even if leaders are offline, and the liveness guarantee $E[r] < 1.5$ is preserved.

Definition 5.4 (Sequential Work Function Configuration)**.** The Sequential Work Function (SWF) utilizes an **iterated-Poseidon** construction with STARK-verified evaluation[1].

The sequential function operates over the Goldilocks field $\mathbb{F}_p$ ($p = 2^{64} - 2^{32} + 1$).

1. The Step Function: The SWF iterates the Poseidon2 permutation (same instantiation as the state Merkle tree: $t = 16$, $\alpha = 7$, $R_f = 8$, $R_p = 14$) with domain-separated input:

$$s_{i+1} \leftarrow \text{Poseidon2}(\texttt{0x04} \| s_i \| i) \tag{70}$$

where $s_0 = H_{\text{snapshot}}$ and the output is taken from the first capacity element. Each step is inherently sequential: s_{i+1} cannot be computed without s_i. After T_{delay} iterations, the evaluator

[1]We evaluated several VDF candidates. Repeated-squaring VDFs over RSA groups require a trusted setup or class-group computation, and their post-quantum status is unclear. Algebraic VDFs such as MinRoot [27] were initially attractive for their STARK-friendly arithmetization, but the CRYPTO 2024 cryptanalysis by Biryukov et al. [28] demonstrated that smoothness-based parallelization attacks can reduce algebraic VDF latency by 16–20×, breaking the core sequentiality assumption. The Ethereum Foundation subsequently suspended MinRoot development. Iterated-hash constructions avoid this structural weakness because each step depends on the full output of the previous step, leaving no algebraic decomposition for parallel evaluation.

produces the final state $s_{T_{\text{delay}}}$ along with a STARK proof π_{SWF} attesting to correct evaluation of the entire chain. Validators verify π_{SWF} in $O(\log T_{\text{delay}})$ time.

2. ASIC Resistance: Unlike algebraic VDFs where exponentiation can be decomposed via smooth-order attacks, iterated-hash sequentiality is circuit-depth bounded: the ASIC advantage is limited to reducing the per-step latency of a single Poseidon2 permutation. Empirical estimates and hardware implementations of iterated hash functions bound this advantage at $\alpha_{\max} \approx 3$–$5\times$, since the critical path through 22 rounds of Poseidon2 cannot be parallelized or pipelined across steps. The delay parameter is calibrated accordingly:

$$T_{\text{delay}} = \text{Benchmark}_{\text{CPU}} \times 5 \times \text{EpochTime} \tag{71}$$

An adversary with a $5\times$ ASIC advantage therefore cannot look ahead beyond the current epoch boundary.

3. Verification Efficiency: The Poseidon2 step function uses the same field and instantiation as the batch proof circuit, so the STARK prover infrastructure already supports it with no additional arithmetization. The SWF proof is a standard STARK over a single-column trace of T_{delay} Poseidon2 evaluations; at $T_{\text{delay}} = 2^{20}$ iterations (≈ 300s on a single core at $\approx 3.5\,\mu$s per Poseidon2 permutation), the verification proof is ≈ 80 KB and verifies in < 15 ms.

5.2.2 SWF Committee Specification

Protocol 5.3 (SWF Committee)**.** The SWF computation that produces epoch randomness is performed by a dedicated committee:

Committee Selection:

1. At the end of epoch $e-1$, the SWF committee for epoch e is deterministically selected from the finalized validator set of epoch $e-1$
2. Committee size: $N_{SWF} = 100$ validators
3. Selection: top 100 validators by $H(\text{seed}_{e-1} \| \text{“swf\_committee”} \| pk_{val})$, weighted by stake
4. This avoids circular dependency: the committee is derived from the *previous* epoch's seed, not the current one

Incentive Structure (Shared Rewards):

- **Decentralized Payout:** To prevent a single high-performance entity from monopolizing rewards, the SWF Reward R_{SWF} is split among the fastest valid submissions:
  - **Leader (50%):** The first valid proof π_1 sets the epoch seed and receives $0.5 \times R_{SWF}$.
  - **Redundancy Pool (50%):** The remaining $0.5 \times R_{SWF}$ is split equally among the next $k = 5$ valid submissions that are included in DAG vertices with timestamps $t \leq t_{leader} + \delta$ (where $\delta = 2$s), as ordered by the canonical DAG history.
- This "Fastest-N" model incentivizes honest redundancy. Even if one entity is slightly faster, others remain profitable by falling into the redundancy pool, ensuring backup generation capability is maintained.
- Committee members who do not attempt computation receive no penalty.

Liveness Fallback:

- If no valid SWF output is submitted within $2 \times T_{delay} = 600$s of epoch start, the protocol uses a **degraded seed**:

$$\text{seed}_e^{fallback} = H(\text{“degraded”} \| H(B_{last}^{e-1}) \| e) \tag{72}$$

where B_{last}^{e-1} is the last finalized block of the previous epoch

- The degraded seed is less resistant to bias (an adversary who controls the last few blocks of epoch $e-1$ gains marginal influence) but preserves liveness
- If degraded seeds are used for > 3 consecutive epochs, a governance alert is triggered

5.2.3 SWF Committee Failure Analysis

The SWF committee's reliability follows from straightforward redundancy. With 100 members and $\beta < 1/3$ adversarial stake, at least 66 members are honest. Each honest member fails to submit only through independent hardware, network, or software faults—collectively bounded at $P_{\text{node}} < 0.013$ per node. The probability that all 66 fail simultaneously is $(0.013)^{66} < 2^{-105}$, which is negligible (less than once per 10^{31} years at 1-hour epochs).

5.2.4 Fallback Seed Bias

If no SWF output arrives within $2T_{\text{delay}} = 600$s, the protocol falls back to $\text{seed}_e^{\text{fallback}} = H(\text{“degraded”} \| H(B_{\text{last}}^{e-1}) \| e)$. An adversary controlling the last block of epoch $e-1$ can grind at most $\approx 1400 \times 5 \approx 2^{13}$ candidate hashes (timestamp jitter $\times$ proof batch selection). Against a 256-bit hash output, this yields an election bias of $\epsilon < 2^{-242}$—cryptographically negligible.

5.2.5 SWF Committee Incentives

Committee members are NOT slashed for non-participation: the computation is deterministic, redundancy is high ($N_{\text{SWF}} = 100$), and the failure probability of $< 2^{-105}$ ensures liveness without coercion. The committee is re-selected each epoch using the previous epoch's seed, preventing entrenchment.

5.3 Strong Citations and Ordering

Definition 5.5 (Ancestor Set)**.** Let $Anc(v)$ be the set of all vertices reachable from v by following edges in reverse.

Definition 5.6 (Strong Citation)**.** A vertex v *strongly cites* u if:

1. $u \in Anc(v)$
2. The total stake of distinct validators that created vertices in $Anc(v)$ and also reference u is $> 2/3$ of total stake.

Lemma 5.2 (Round Monotonicity)**.** *If vertex v strongly cites vertex u, then $round(v) > round(u)$.*

Proof. By the round assignment rule, a vertex v is assigned to round r only if it strongly cites at least one witness from round $r-1$. Strong citation requires $u \in Anc(v)$ with $> 2/3$ stake support (Definition 5.3). Since edges in the DAG are strictly causal (v references only previously created vertices), any vertex u that v strongly cites must have been assigned to a round $r' \leq r-1$ at creation time. Hence $round(v) = r > r' = round(u)$. □

5.3.1 Timestamp Validity Rule

To prevent ordering manipulation, a vertex v is valid iff:

$$|v.\text{timestamp} - \text{LocalTime}| \leq \epsilon + \Delta_{\text{net}} \tag{73}$$

where $\epsilon = 1500$ms (NTP drift bound) and $\Delta_{\text{net}} = 500$ms (network delay).

Vertices violating $|v.\text{timestamp} - \text{LocalTime}| > 2000$ms are rejected.

Timestamps are used only for validity checks, never for ordering. Adversaries with compromised time sources cannot manipulate consensus.

5.3.2 Leader Selection and Fork Choice

For each round r, a random "Leader Vertex" L_r is elected via hash-based leader election (Section 5.2.1).

- **Fork Choice Rule:** Validators select the chain tip that maximizes the **Effective Stake Support**.

$$Score(Chain) = \sum_{v \in \text{UniqueValidators}(Chain)} \text{Stake}(v)$$

where UniqueValidators($Chain$) is the set of distinct validators who have attested to any block in the chain's history—counting unique attestors rather than raw vote count defeats vote-spamming.

5.4 Message Complexity & Fork Choice

Each validator broadcasts its vertex to $D = 6$ peers per GossipSub mesh topic (control channel; bulk data uses the Proof Relay Tree with branching factor $b = 3$; see Section 2.6), yielding $O(n)$ messages per validator per round (n = validator count). Total network-wide message load per round is $O(n \cdot D) = O(n)$, and the relay tree propagates bulk messages to all n nodes in $O(\log_b n) \approx 5$ hops. Amortized over a batch of $B = 1000$ transactions, per-transaction consensus overhead is $O(n/B)$.

Nodes evaluate the canonical history by maximizing cumulative weight:

$$ChainWeight(chain) = \sum stake(QC)$$

Tie-breaker: $\min H(\text{seed}_{\text{slot}} \| H(vertex))$ (binding to slot randomness prevents vertex-hash grinding).

5.5 Finalization and Safety

5.5.1 Validator Locking Rule

A validator maintains a local lock on the highest-round vertex for which it has observed a QC. A validator **MUST NOT** issue a vote for any vertex v' that conflicts with its current lock v, unless it observes a QC for a successor of v' in a higher round. This rule guarantees monotonic lock advancement.

5.5.2 Formal Lock-Update Mechanism

Definition 5.7 (Locking Invariant)**.** A validator v maintains a lock variables (L_{round}, L_{hash}). The validator implicitly locks on a vertex B if it votes for a QC referencing B.

Protocol 5.4 (Safety Rule / SafeNode Predicate)**.** A validator v accepts a proposal B_{new} in round r_{new} extending parent B_{parent} if and only if:

$$r_{new} > L_{round} \quad \textbf{AND} \quad (B_{parent} \sqsupseteq L_{hash} \vee r_{parent} > L_{round}) \tag{74}$$

Therefore a validator never votes for a branch that conflicts with a higher-round lock, guaranteeing the "Unique QC" property of safety.

Lemma 5.3 (Lock Monotonicity)**.** *For any honest validator V_i:*

$$LockedRound_i(t_2) \geq LockedRound_i(t_1) \quad \forall t_2 > t_1 \tag{75}$$

Locks advance monotonically, ensuring safety.

Proof. By contradiction. Suppose LockedRound_i decreases from r to $r' < r$. This requires V_i to accept $QC(v', r')$ where $r' < r$. But the protocol only updates locks when receiving $QC(v', r')$ with $r' > \text{LockedRound}_i = r$. Contradiction. □ □

5.5.3 Finalization Rule (3-Chain)

A vertex v is considered **Finalized** when there exists a sequence of ratified leaders L_r, L_{r+1}, L_{r+2} such that:

- L_{r+2} strongly cites L_{r+1}
- L_{r+1} strongly cites L_r
- $v \in Anc(L_r)$

Chain-Attributed Finality. A proof batch π with `chain_id` $= C$ achieves finality when the DAG vertex containing π satisfies the 3-chain rule above. Finality is attributed to chain C—meaning QChain C's state root $\text{StateRoot}_{\text{new}}$ from that batch is globally final and may be referenced by cross-chain transactions on other chains. Validators maintain a per-chain finalized state root index:

$$\text{FinalizedRoots} : \texttt{ChainId} \to (\text{slot},\ \text{StateRoot}) \tag{76}$$

updated atomically when a vertex finalizes. This index is the authoritative source for cross-chain transaction verification (Section 13). During single-chain operation, the index contains exactly one entry (`chain_id` $= 0$).

Assumption 5.1 (Validator Set Stability Window)**.** The validator set and stake distribution remain constant during any three consecutive rounds required for finalization. Formally, for any rounds $r, r+1, r+2$, the validator identity set and their associated stake weights are identical. If a 3-chain spans an epoch boundary (rounds $r, r+1$ in epoch e and round $r+2$ in epoch $e+1$), the $> 2/3$ intersection constraint (Theorem 5.4.1 below) ensures that sufficient honest validators carry their lock states into the new epoch, preserving safety continuity.

Theorem 5.4 (Validator Set Intersection)**.** *To preserve safety across epoch transitions, we enforce:*

$$|V_{epoch} \cap V_{epoch+1}| > \frac{2}{3}N \tag{77}$$

This overlap guarantees that any QC formed in epoch $e+1$ must contain at least one honest validator from epoch e who is aware of the lock state, preventing "nothing-at-stake" grinding attacks on the handover.

Lemma 5.5 (DAG Finalization Consistency)**.** *If vertex v is finalized via a 3-chain (L_r, L_{r+1}, L_{r+2}), then all state transitions in $Anc(v)$ are uniquely determined and mutually consistent. Specifically, no two finalized vertices can commit conflicting state diffs.*

Proof. Suppose for contradiction that two finalized vertices v, v' commit conflicting state transitions (overlapping write sets with different values). Finalization of v requires a 3-chain where each QC involves $> 2/3$ stake. Finalization of v' requires an independent 3-chain with $> 2/3$ stake. By the Intersection Property, the QCs in v's chain and v''s chain share at least one honest validator V_h.

By the SafeNode Predicate (Protocol 5.3), V_h holds a lock (L_{round}, L_{hash}) after voting in v's chain. To vote in v''s conflicting chain, V_h must satisfy $r'_{new} > L_{round}$ AND $(B'_{parent} \sqsupseteq L_{hash} \vee r'_{parent} > L_{round})$. Since v and v' are conflicting (different state diffs on overlapping accounts), B'_{parent} cannot extend L_{hash} (they represent incompatible histories). Thus V_h requires $r'_{parent} > L_{round}$, which would imply a higher-round QC on v''s branch — but this contradicts v's finalization, as the 3-chain for v already committed round $r+2$.

Therefore, V_h cannot vote for both chains, and at most one can achieve a QC involving $> 2/3$ stake. □ □

Theorem 5.6 (Safety)**.** *No two conflicting blocks can both be finalized unless $\beta \geq 1/3$.*

Proof. Finalization requires a 3-chain of quorum-certified leaders. Each QC requires $> 2/3$ stake. Two conflicting blocks require two disjoint $> 2/3$ voting sets.

Intersection property:

$$|A| + |B| > \tfrac{4}{3}N \Rightarrow |A \cap B| > \tfrac{1}{3}N$$

Thus at least one honest validator must have voted twice, contradicting the Safe Voting Rule (Protocol 5.3). By the DAG Finalization Consistency Lemma, this extends to state-level consistency: finalized vertices cannot commit conflicting state transitions. □

Theorem 5.7 (Liveness After GST)**.** *If $\beta < 1/3$ and network is synchronous, a block is finalized within $O(R)$ rounds.*

Sketch. Leader election is unpredictable (SWF-seeded, hash-derived). With expected leader count $p = 1.0$ (Section 5.2.1), the per-round probability of electing at least one honest leader is $P \geq 1 - e^{-(1-\beta)} \geq 1 - e^{-2/3} \approx 0.487$ (Poisson approximation for n large). Since $\beta < 1/3$, the expected rounds until an honest leader produces a ratified vertex is $E[r] \leq \frac{1}{P} \approx 2.05$. **Empty rounds** (zero leaders elected, probability $e^{-1} \approx 0.37$) and **multi-leader rounds** (probability ≈ 0.26, resolved by fork choice) are accounted for in this bound. Progress occurs with constant expected delay. □

5.5.4 Adversarial Liveness Analysis

The expected-case liveness bound $E[r] \leq 2.05$ rounds assumes honest leaders can communicate freely. Under adaptive adversarial attacks, we must analyze worst-case liveness:

Theorem 5.8 (Liveness Under Targeted DoS)**.** *If an adversary can selectively DoS honest leaders for duration Δ_{DoS}, but cannot prevent leader election, then the protocol maintains liveness with finalization time bounded by:*

$$T_{finalize} \leq 3 \times \max\left(T_{slot}, \Delta_{DoS}\right) + 4T_{slot} \tag{78}$$

provided $\Delta_{DoS} < T_{epoch}$ (adversary cannot DoS for an entire epoch).

Proof. **Attack Model:**

Assume adversary observes leader election outcomes via hash-based election (Section 5.2.1) and launches targeted DoS against honest leaders:

1. Adversary computes eligible leaders for slot s: $\{L_1, L_2, \ldots, L_k\}$
2. Identifies honest subset: $\{L_1^H, \ldots, L_m^H\}$ where $m \geq (1 - \beta) \cdot k$
3. Launches DoS attack preventing honest leaders from broadcasting for Δ_{DoS}

Protocol Response:

By Protocol 5.2 (Silent Leader Progression), validators waiting for a proposal from slot s automatically advance to slot $s + 1$ after deadline $t_{\text{deadline}} = T_{\text{start}}(s) + \Delta_{\text{net}}$.

If adversary can sustain DoS for $\Delta_{\text{DoS}} > \Delta_{\text{net}}$:

- Slot s is skipped (marked SKIPPED by honest validators)
- Protocol advances to slot $s + 1$ after T_{slot} interval
- Adversary must re-apply DoS to slot $s + 1$ leaders

Cost of Sustained Attack:
To DoS all honest leaders in slot s:

$$\text{Cost}_{\text{DoS}}(s) \geq m \times \text{bandwidth}_{\text{attack}} \times \Delta_{\text{DoS}} \tag{79}$$

For $m \approx \frac{2}{3} \times k$ and k determined by Poisson distribution with $\lambda = p = 1.0$:

$$P(k \geq 1) \approx 1 - e^{-1} \approx 0.63 \tag{80}$$
$$E[k|k \geq 1] \approx 1.58 \tag{81}$$
$$E[m] \approx 0.67 \times 1.58 \approx 1.06 \tag{82}$$

Adversary's Capability Constraint:
Assume adversary has finite DoS capacity C_{DoS} (measured in node-seconds). To skip n consecutive slots:

$$\text{Required Capacity} = n \times E[m] \times \Delta_{\text{DoS}} \leq C_{\text{DoS}} \tag{83}$$

Maximum Consecutive Skipped Slots:

$$n_{\max} = \left\lfloor \frac{C_{\text{DoS}}}{E[m] \times \Delta_{\text{DoS}}} \right\rfloor \tag{84}$$

After $n_{\max}$ slots, adversary exhausts DoS capacity or epoch boundary is reached, allowing protocol to recover.

Finalization Time Bound:
To finalize a vertex, protocol needs 3 consecutive successful rounds (3-chain rule). Under attack:

- Worst case: Adversary skips $n_{\max}$ slots before capacity depletion
- Recovery phase: 3 honest slots needed for finalization
- Each honest slot requires $\leq T_{\text{slot}} + \Delta_{\text{DoS}}$ (DoS delay + slot time)

Total finalization time:

$$T_{\text{finalize}} = n_{\max} \times T_{\text{slot}} + 3 \times (T_{\text{slot}} + \Delta_{\text{DoS}}) \tag{85}$$
$$\leq 3 \times \max(T_{\text{slot}}, \Delta_{\text{DoS}}) + 4T_{\text{slot}} \tag{86}$$

assuming $n_{\max} \leq 3$ (adversary cannot sustain DoS indefinitely). □ □

Corollary 5.8.1 (Expected Liveness Under Attack)**.** *If adversary can DoS honest leaders with probability $p_{DoS} < 0.5$ per slot, expected rounds to finalization is:*

$$E[r_{attack}] \leq \frac{1}{1 - p_{DoS}} \times E[r_{honest}] \tag{87}$$

For $p_{DoS} = 0.3$ (adversary DoS's 30% of honest leader slots):

$$E[r_{attack}] \leq \frac{1}{0.7} \times 2.05 \approx 2.93 \text{ rounds} \tag{88}$$

At typical operating parameters:

- **Best case (no attack):** $E[r] \approx 2.05$ rounds ≈ 1.6s (at $T_{\text{round}} \approx 0.8$s)
- **Under moderate DoS ($p_{\text{DoS}} = 0.3$):** $E[r] \approx 2.93$ rounds ≈ 2.3s
- **Worst case (sustained DoS):** $T_{\text{finalize}} < 60$s (bounded by epoch transition)

Finalization latency thus degrades gracefully in proportion to the adversary's DoS capacity, without compromising safety.

Assumption 5.2 (Post-GST Timing Requirement)**.** For liveness, we require:

$$\Delta_{\text{prove}}^{\text{honest}} + 2\Delta_{\text{network}} < T_{slot} \tag{89}$$

With $T_{prove} \approx 4.2$s (4× RTX 4090 GPU cluster, Plonky3) and $\Delta_{network} = 1s$, setting $T_{slot} = 12$s provides a safety margin of ≈ 6.8s. Operators using older hardware (RTX 3090) may see $T_{prove} \approx 8$–10s, still within the slot window.

Condition 5.1 (Honest Leader Availability)**.** Our liveness bound $E[r]$ holds if and only if:

$$\forall t, \exists \text{ honest leader } L \text{ such that } L \text{ is online AND } Msg(L) \text{ reaches quorum within } \Delta_{net} \tag{90}$$

If the adversary can selectively DoS every honest leader for Δ_{net}, liveness is not guaranteed until the DoS ceases (GST).

5.6 Congestion-Adaptive Base Fee (Exponential Damping)

To prevent oscillation and manipulation, QBit uses an exponential update rule:

$$BaseFee_{t+1} = BaseFee_t \times \exp\left(\alpha \frac{g_t - G}{G}\right) \tag{91}$$

where $\alpha \in (0, 1)$ is the damping factor. Unlike a linear proportional controller, the exponential form avoids overshoot oscillations that an adversary could exploit for fee manipulation.

Transaction priority is determined by tip:

$$PriorityFee = Fee_{user} - BaseFee$$

Sentries select transactions maximizing total priority fees subject to gas constraints.

Mempool Replacement Strategy To prevent stuck transactions, a new transaction $tx'_{nonce=k}$ may replace an existing $tx_{nonce=k}$ in the mempool if and only if:

$$Fee(tx') \geq 1.1 \times Fee(tx) \quad (10\% \text{ Min Price Bump}) \tag{92}$$

Sentries are required to select the highest-fee variant when constructing the deterministic batch.

Forced Inclusion Mechanism To ensure censorship resistance without enabling DoS, QBit implements rate-limited forced inclusion:

Protocol 5.5 (Censorship-Resistant Forced Inclusion)**.** Users can bypass Sentries by submitting transactions directly to validators:

Fee Structure: To prevent "cheap censorship" of high-value DeFi transactions (e.g., liquidations), the cost for Forced Inclusion scales exponentially if the lane is saturated:

$$F_{\text{forced}}(t) = 10 \times BaseFee_t \times 2^{(S_{\text{consecutive}})} \tag{93}$$

where $S_{\text{consecutive}}$ is the number of consecutive slots the forced inclusion lane has been 100% full. The exponential escalation ($Cost \to \infty$) makes sustained censorship attacks economically untenable within minutes.

Rate Limits (Per-Account):

$$\text{MaxForcedTx}_{\text{account}} = \min\left(5 \text{ per hour}, \frac{\text{AccountStakeAge}}{1000}\right) \tag{94}$$

where AccountStakeAge = balance × age_in_blocks.

Global Capacity Limit:

$$\text{MaxForcedTx}_{\text{global}} = 0.05 \times \text{BlockCapacity} \tag{95}$$

At most 5% of each block may contain forced transactions.

Anti-Dust Protection:

$$tx.\text{value} \geq 10 \times \text{BaseFee}_t \quad \text{(minimum value for forced inclusion transactions)} \tag{96}$$

Without such a floor, an adversary could spam-fill the 5% forced inclusion capacity with dust transactions, crowding out legitimately censored ones.

DoS Resistance:

DoS Cost Analysis. An attacker attempting to saturate forced inclusion for T seconds requires:

$$\text{Cost}_{\text{attack}} = T \times \text{TPS}_{\text{forced}} \times F_{\text{forced}} \tag{97}$$

With parameters:

$$\text{TPS}_{\text{forced}} = 0.05 \times 30,000 = 1,500 \text{ TPS} \tag{98}$$

$$F_{\text{forced}} = 10 \times \text{BaseFee} \tag{99}$$

$$\text{BaseFee} \approx 0.0001 \text{ QBIT} \tag{100}$$

To sustain attack for 1 hour:

$$\text{Cost} = \sum_{t=0}^{300} \left(10 \times \text{BaseFee} \times 2^{\min(t,20)}\right) \to \infty \tag{101}$$

The cost scales **exponentially** with consecutive saturated slots ($2^{S_{consecutive}}$ multiplier). Within minutes, the cost exceeds the total supply of QBIT, making sustained censorship attacks economically impossible regardless of attacker wealth.

Legitimate users can still access 95% of block capacity via the normal Sentry path.

Priority Ordering Within Forced Pool: Forced transactions are ordered by:

$$\text{Priority} = \text{Fee}_{\text{total}} \times \text{AccountStakeAge}^{0.5} \tag{102}$$

This favors long-term users over fresh wallets (Sybil resistance).

Fallback Mode If forced inclusion exceeds capacity for > 6 hours (sustained attack), the protocol triggers the Sentry Failure Fallback (Section 2.2, Protocol 2.2.2), using the same DAG-signaled voting and hysteresis constraints defined there.

Liveness under extreme censorship is maintained without sacrificing decentralization.

6 Validator Lifecycle

Validators progress through six states: `INACTIVE` → `PENDING` → `ACTIVE` → `EXITING` or `SLASHED` → `WITHDRAWN`. The lifecycle follows standard delegated proof-of-stake conventions and is summarized here; the full specification is deferred to a separate protocol document.

Entry. A deposit of $\geq S_{\min} = 10{,}000$ QBIT with a valid ML-DSA-65 public key enters the activation queue. At most $\max(4, N_{\text{active}}/65{,}536)$ validators activate per epoch, limiting churn to $\approx 1.5\%$/hour at $N_{\text{active}} = 256$.

Exit. Voluntary exits follow the same rate limit. Exiting validators serve a 21-day unbonding period during which stake remains slashable.

Slashing. Equivocation (double-voting): 100% slash. Invalid DA response: 50% slash. Liveness fault (> 90% missed votes): 0.01% decay per missed vote. Correlated faults amplify penalties: $P_{\text{slash}}^{\text{corr}} = \min(1,\ P_{\text{slash}} \times (1 + N_{\text{slashed}}/N_{\text{active}}))$. Slashed validators face 90-day extended unbonding.

Delegation. Delegators share slashing risk. Delegation is capped at $10 \times \text{SelfStake}_V$ per validator.

Validator set bounds. $N_{\min} = 100$, $N_{\text{target}} = 256$, $N_{\max} = 1024$. If demand exceeds $N_{\max}$, the lowest-staked validator is replaced by any candidate staking $> 1.1\times$ the incumbent, subject to the per-epoch churn limit.

Epoch transitions. The intersection guarantee $|\mathcal{V}_e \cap \mathcal{V}_{e+1}| > 2/3 \cdot \min(|\mathcal{V}_e|, |\mathcal{V}_{e+1}|)$ holds by construction from the churn limit (max 8 changes per epoch at $N = 256$, leaving $\geq 248/256 = 96.9\%$ overlap). This ensures honest validators carry lock states across epoch boundaries (Assumption 5.4).

Performance Monitoring. Validators are tracked on attestation accuracy (A_V, target $> 95\%$), block proposal success (P_V, target $> 98\%$), and DA participation (D_V, target $> 99\%$). If $A_V < 80\%$ for 7 consecutive days, rewards are halved; after 30 days, automatic exit is triggered.

7 Proof Mining Economics

Without careful incentive design, proving naturally concentrates: the fastest prover wins every slot, creating a monopoly. This section describes the mechanisms that prevent that outcome.

In equilibrium, competitive pressure drives the market price of proving toward the marginal cost of production (electricity + hardware amortization), eliminating rent-seeking by early entrants. The mechanisms below are designed to ensure that this equilibrium is reached and maintained.

7.1 The Proving Market (Multi-Leader Parallel Proving)

To prevent cartel formation, QBit implements **Parallel Competitive Proving**:

Protocol 7.1 (Windowed Proportional Rewards)**.** Rewards are distributed among all valid proofs arriving within a specific time window δ to encourage decentralized hardware operations.

An earlier design allocated the entire batch reward to the first valid proof (winner-take-all). In simulation, this consistently collapsed to a single dominant prover within 200 epochs—the fastest hardware operator captured every slot, recouped costs first, and undercut late entrants on price. The windowed scheme below was adopted to prevent this outcome.

1. The Validity Window (δ): Let t_{first} be the arrival time of the first valid proof for slot i. The window closes at:

$$t_{close} = t_{first} + \delta \tag{103}$$

where $\delta = 500$ms (approximating global round-trip time).

2. The Eligibility Set: Let S_{valid} be the set of Sentries who submitted a valid proof π such that $t_{arrival}(\pi) \leq t_{close}$.

3. Proportional Payout: The proving reward R_{batch} is distributed among S_{valid} using a weighted decay function:

$$R_s = R_{batch} \times \frac{w_s}{\sum_{j \in S_{valid}} w_j} \tag{104}$$

where weight $w_s = e^{-\lambda(t_s - t_{first})}$. The decay constant λ is tuned so that a 100ms delay reduces the reward share by roughly 5%; the 500 ms window itself is a pragmatic choice—tighter windows would favor better-connected Sentries, while wider ones dilute the first-mover incentive.

Proving Bonds (Anti-Cartel):

- Each Sentry must stake $B_{\text{prove}} = 10,000$ QBIT as proving bond
- **Slashing Condition:** If a Sentry produces $< 50\%$ of expected proofs over an epoch, slash:

$$\text{Penalty} = 0.01 \times B_{prove} \times \text{MissedProofs}$$

- **Rationale:** Prevents cartels from "taking turns" (coordinated non-competition)

Cartel Instability (Informal). Under the assumptions that Sentries are rational, $N_{\text{leaders}} \geq 5$ per slot, and proving bonds are enforced, any cartel agreement faces strong destabilizing pressures from three independent mechanisms: a member can defect by proving alongside the cartel's designated winner and capture $\approx 0.5R$ instead of the cartel-shared R/c; not proving to honor a cartel turn triggers slashing; and above-market fees attract new entrants. Theorem 7.1.1 below analyzes single-round defection incentives. Repeated-game dynamics may permit equilibria involving punishment strategies, but these are disrupted by the slashing mechanism (which penalizes non-participation regardless of cartel coordination) and the permissionless entry of new Sentries.

7.1.1 Formal Game-Theoretic Analysis

Consider a cartel $\mathcal{C} \subseteq \mathcal{S}$: a coalition of Sentries that coordinates to rotate proving rights (one member proves per slot), share rewards equally, and suppress external competition.

Theorem 7.1 (Cartel Instability Under Multi-Leader Selection)**.** *For $N_{leaders} \geq 5$ and slashing bonds enforced, any cartel of size $|\mathcal{C}| > 2$ is unstable against defection.*

Proof. Consider a cartel $\mathcal{C}$ with $|\mathcal{C}| = c$ members.

Cartel Agreement: Members take turns proving, sharing total rewards R_{total} equally.

Each member's expected payoff under cooperation:

$$\Pi_{coop} = \frac{R_{total}}{c} - \text{Cost}_{proving} \tag{105}$$

Defection Payoff:

If member i defects in slot where they're eligible:

- Defector proves immediately alongside the cartel's designated winner
- Other cartel members don't prove (coordination agreement)
- Defector doesn't share reward

Expected defection payoff (per slot where eligible), assuming the defector splits the reward window with the cartel's designated winner:

$$\Pi_{defect} \approx 0.5R_{batch} - \text{Cost}_{proving} \tag{106}$$

Condition for Instability:

Cartel is unstable iff defection is profitable:

$$\Pi_{defect} > \Pi_{coop} \tag{107}$$

$$0.5R_{batch} > \frac{R_{total}}{c} \tag{108}$$

For $N_{leaders} = 5$ and assuming effective competition approximates sharing the window:

$$\Pi_{defect} \approx 0.5R_{batch} \tag{109}$$

Therefore:

$$0.5R_{batch} > \frac{R_{batch}}{c} \tag{110}$$

$$c > \frac{1}{0.5} = 2 \tag{111}$$

Thus for $c \geq 2$, defection is profitable.
Slashing Threat:
If cartel members don't prove (to honor agreement), they trigger slashing:

$$\text{Penalty} = 0.01 \times B_{prove} \times \text{MissedProofs} \tag{112}$$

Over an epoch with expected E proofs, non-participation costs:

$$\text{Cost}_{non-participation} = 0.01 \times 10{,}000 \times E \approx 100E \text{ QBIT} \tag{113}$$

This makes "taking turns" economically irrational. □ □

Theorem 7.2 (Minimum Viable Cartel Size)**.** *A cartel can sustain above-market rewards only if it controls a sufficient fraction of Sentry stake to suppress external competition. Let ϕ denote the fraction of total Sentry stake held by the cartel.*

Proof. Theorem 7.1.1 establishes that defection is profitable for any cartel of size $c \geq 2$ when non-cartel Sentries compete in the proving window. The cartel can restore stability only by ensuring that non-cartel Sentries are rarely elected as slot leaders.

Under hash-based leader election (Section 5.2.1), the probability that at least one non-cartel Sentry is elected in a given slot follows a Poisson approximation:

$$P(\text{outsider elected}) = 1 - e^{-N_{\text{leaders}} \times (1-\phi)} \tag{114}$$

For the cartel to face competition in fewer than 10% of slots (making sustained coordination marginally viable despite defection risk):

$$e^{-N_{\text{leaders}} \times (1-\phi)} \geq 0.9 \quad \Rightarrow \quad (1-\phi) \leq \frac{\ln(10/9)}{N_{\text{leaders}}} = \frac{0.105}{5} \approx 0.021 \tag{115}$$

Thus the cartel requires $\phi \geq 0.98$, i.e., control of $\geq 98\%$ of total Sentry stake.
Even at the more relaxed threshold of competition in $\leq 20\%$ of slots:

$$(1-\phi) \leq \frac{\ln(5/4)}{5} \approx 0.045 \quad \Rightarrow \quad \phi \geq 0.955 \tag{116}$$

Since Sentries cannot exclude others from hash-based election (publicly verifiable, stake-weighted), suppressing competition requires capturing $> 95\%$ of Sentry stake. For this to be economically rational, the attacker's capital commitment must exceed $0.95 \times S_{\text{total}} \times P_{\text{QBIT}}$, which the protocol targets to be economically prohibitive through staking requirements and token supply design. □ □

7.1.2 Empirical Monitoring and Circuit Breakers

Beyond game-theoretic guarantees, QBit implements active cartel detection:

Protocol 7.2 (Automated Circuit Breaker)**.** If CollusionScore $< \theta_{threshold}$ for > 24 hours (where $\theta_{threshold}$ is the epoch-adaptive threshold defined above):

1. Governance receives automatic alert

2. Emergency parameter adjustment vote triggered
3. If vote passes: $N_{leaders} \to 10$ (temporary)
4. Cartel profitability calculation resets, forcing dissolution

Attack Vector Warning: Automated parameter updates introduce a "Dilution Attack" vector where attackers manipulate metrics to force suboptimal parameters. To mitigate this, updates are rate-limited (max 1 change per epoch) and subject to a 24h manual veto window.

Off-chain collusion channels remain a theoretical concern—as with any permissionless system—but the combination of multi-leader selection, slashing, and automated detection makes sustained cartel operation economically impractical.

7.2 Cartel Detection

Cartel detection uses two signals: an anomalous win-rate statistic $S_i = \text{wins}_i/\text{expected}_i$ and a low temporal variance in proving times ($\text{CollusionScore} < \theta_{\text{threshold}}$). If both trigger simultaneously for > 24 hours, governance receives an alert and may temporarily increase N_{leaders}. Statistical detection does *not* trigger automatic slashing—slashing is reserved for cryptographically provable faults.

7.3 Bootstrap Subsidy

At genesis, 5% of total QBIT supply is allocated to a proving subsidy that decays by 25% per quarter: $R_{\text{subsidy}}(e) = R_0 \times 0.75^{\lfloor e/E_{\text{quarter}} \rfloor}$. The subsidy terminates automatically when 30-day rolling fee revenue exceeds the current subsidy rate, or after 4 years, whichever comes first. Under baseline assumptions (30K TPS, 0.0001 QBIT avg fee), this occurs in 2–3 years.

8 State Management

At sustained 30,000 TPS with Ethereum's historical account-creation ratio ($\alpha \approx 0.05$), the active state grows at ≈ 19 GB/day, reaching 7 TB within a year. Without mitigation, this exceeds mid-range server storage.

QBit addresses state growth through two mechanisms:

- **Recursive epoch compression** (Section 9.3): Historical proofs and diffs are recursively aggregated into a single epoch proof, allowing validators to prune all per-batch data after finalization. Active state (Σ_{current}) is retained permanently; historical trace is prunable.
- **State rent** (deferred): A touch-based rent mechanism with 90-day grace periods, lazy collection, and account eviction after 2 years of inactivity will be activated via governance vote after 6+ months of mainnet operation. State rent specification is deliberately deferred to post-mainnet, following the precedent established by Ethereum's multi-year state management research (EIP-4444, Verkle trees), where premature specification has proven worse than data-driven design. The full specification—including rent pricing, eviction/resurrection protocols, and contract exemption criteria—is maintained as a separate protocol document (QIP-002). The recursive epoch compression mechanism (Section 9.3.2) provides interim storage relief by enabling pruning of historical proofs.

With rent active and 20% of accounts remaining active (consistent with Ethereum data), the steady-state active storage is ≈ 90 GB, sustainable on a 1 TB NVMe SSD.

9 Performance Analysis

All throughput and bandwidth figures in this section are projections based on the constraint estimates in Section 3.3.1. Empirical validation with a reference circuit implementation is required before mainnet parameter finalization.

Throughput:

- **Launch Target:** 30,000 TPS (400 active Sentries, baseline circuit)
- **Scaling Headroom:** Higher throughput achievable via circuit optimization and additional Sentries (see §3.2 Future Throughput Improvements)

All bandwidth and performance specifications in this paper assume the 30,000 TPS baseline unless otherwise noted.

9.1 Throughput and Bandwidth

Performance Analysis Parameters:

Proof Size: Based on STARK parameters from Section 3.2:

- Trace length: $T \approx 2^{26}$
- Blowup factor: $\rho = 8$
- FRI queries: 100
- Field size: 64-bit

Proof components:

$$\text{FRI layers} \approx 26 \times 2\text{KB} = 52\text{KB} \tag{117}$$

$$\text{Deep queries} \approx 100 \times 512\text{B} = 50\text{KB} \tag{118}$$

$$\text{Merkle authentication paths} \approx 30\text{KB} \tag{119}$$

$$\text{Out-of-domain evaluations} \approx 10\text{KB} \tag{120}$$

$$\text{Metadata} \approx 5\text{KB} \tag{121}$$

$$\textbf{Total proof size} \approx \textbf{147}\text{KB} \tag{122}$$

State Diff Size: For 1000 transactions:

$$\text{Accounts modified} \approx 1500 \text{ (avg 1.5 per tx)} \tag{123}$$

$$\text{Bytes per account} \approx 150\text{B (nonce + balance + storage)} \tag{124}$$

$$\text{Total state diff} \approx 225\text{KB} \tag{125}$$

Bandwidth Calculation (30,000 TPS): **Note:** Bandwidth scales linearly with throughput. Validators should provision 500 Mbps connections with headroom for network spikes and control traffic.

Comparative Analysis (Validator Ingress):

- **Naive ML-DSA gossip:** 99 MB/s$\times D$=8 = 792 MB/s $\approx$ 6.3 Gbps — requires datacenter-grade infrastructure
- **ZK aggregation only (mesh D=8):** 12.4 MB/s$\times$8 = 99.4 MB/s $\approx$ 795 Mbps — requires 2 Gbps business fiber

Component	Per Batch	Per Second	Amplified
ZK Proofs (147 KB × 30)	147 KB	4.4 MB/s	×3 (tree)
State Diffs (75 KB × 30)†	75 KB	2.25 MB/s	×3 (tree)
DA Sampling (FRI proofs)	–	1.0 MB/s	×3 (tree)
Block Headers (700 B × 400)	–	0.28 MB/s	×6 (mesh)
Votes/Attestations‡	–	0.84 MB/s	×6 (mesh)
Total (Single Stream)	–	**8.77 MB/s**	–
Bulk data (tree, ×3)	–	–	**22.95 MB/s**
Control (mesh, ×6)	–	–	**6.72 MB/s**
Total Validator Ingress	–	–	**29.7 MB/s (237 Mbps)**

Table 9: Bandwidth Requirements at 30,000 TPS. †State diffs are dictionary-compressed (≈ 3×; Section 2.7). ‡ML-DSA-65 attestations (3,309 B each) from ≈ 256 validators per consensus round (≈ 1s). Bulk data (proofs, diffs, DA) uses the Proof Relay Tree (Section 2.6) with effective amplification ≈ 3×; control messages (headers, votes) use GossipSub mesh with $D = 6$. At 1,000 transactions per batch, the network produces 30 batches per second.

- **QBit (ZK + compression + relay tree):** 8.77 MB/s → 29.7 MB/s ≈ 237 Mbps — **achievable on 500 Mbps commodity fiber**
- **Total reduction:** ≈ 27× from naive PQC (6.3 Gbps → 237 Mbps); ≈ 3.4× from ZK-only baseline

Hardware Requirements:

- **Validators:** 500 Mbps connection (commodity fiber, widely available residential/business)
- **Sentries:** 200 Mbps connection + 4× RTX 4090 GPU cluster (bottleneck is GPU, not network)

Realistic Throughput Targets: Given proving time of 3.5–5s per batch (Plonky3, RTX 4090; dependent on hardware):

$$\text{Batches per second} = \frac{1000 \text{ txs}}{4.2\text{s}} \approx 238 \text{ TPS per Sentry} \tag{126}$$

$$\text{With 100 parallel Sentries} \approx 23{,}800 \text{ TPS} \tag{127}$$

$$\text{With 500 Sentries} \approx 119{,}000 \text{ TPS} \tag{128}$$

Launch Target: We target **30,000 TPS** at mainnet launch with approximately 400 active Sentries. This accounts for network overhead, non-uniform batch sizes, and $< 100\%$ Sentry utilization, which reduce effective throughput from the theoretical $400 \times 238 \approx 95{,}200$ TPS ceiling. The ≈ 3× gap between theoretical and target throughput reflects write-set conflicts (≈ 10% wasted proofs under DeFi workloads), partial slot utilization, and network coordination overhead.

Peak theoretical capacity remains $> 100,000$ TPS but requires:

- 800+ active Sentries
- Future hardware improvements (next-gen GPUs)
- Optimized STARK provers

Constraint Sensitivity Analysis. Since the constraint estimates in Section 3.3.1 are projections, the following table shows throughput under varying constraint realizations:

Even in the conservative scenario (2× baseline constraints), raw proving capacity exceeds 33,000 TPS, and the 30K launch target is achievable across all scenarios. The governance-adjustable parameters (N_{sentries}, batch size, T_{slot}) can compensate for higher-than-expected constraint counts.

Scenario	Constraints/tx	Prove Time/Batch	TPS (400 Sentries)
Optimistic (1×)	61,000	3.6s	≈ 66,000
Baseline (1.15×)	70,000	4.2s	≈ 57,000
Realistic (1.5×)	91,500	5.5s	≈ 43,000
Conservative (2×)	122,000	7.3s	≈ 33,000

Table 10: Throughput Sensitivity to Constraint Realization (projected; Plonky3, 4× RTX 4090, 400 Sentries, 1000 tx/batch). Figures show raw proving capacity before write-set conflicts and utilization losses (≈ 30% reduction for the 30K TPS launch target).

9.2 Latency and Finality

The following table breaks down estimated latency for a transaction from submission to finalization:

Phase	Time	Notes
Sentry proving	4.2s	Required before batch submission (Plonky3, RTX 4090)
Proof propagation	0.3s	99th percentile
Validator verification	0.02s	STARK verification
DAG consensus (3 rounds)	1.8–3.0s	$3 \times T_{round}$ (Section 5.2, Def. 5.3)
Total E2E Latency	**≈7.5s**	Without Pre-Confirmation
With Pre-Confirmation	**<500ms**	Economic Soft Finality (Section 9.2.1)

Table 11: End-to-End Latency Breakdown

The proving step is *pipelined*: with $N_{\text{sentries}} \gg 1$ proving in parallel, a continuous stream of proofs enters the DAG at intervals far shorter than T_{prove}. The end-to-end user experience is further accelerated by pre-confirmations (Section 9.2.1) which provide 200–500ms soft finality.

Comparative Performance:

- Ethereum L1: ≈ 13 minutes finality (2 epochs × 32 slots × 12s)
- Solana: ≈ 13s finality (optimistic) but lacks PQC security
- QBit: ≈ 3s consensus finality (≈ 7.5s end-to-end including proving; see Table 10) with cryptographic PQC guarantees

Design Trade-offs: QBit achieves ≈ 3s consensus finality—competitive with pre-quantum systems—while gaining:

1. Post-quantum security (ML-DSA-65)
2. Cryptographic finality (not optimistic)
3. ≈ 32× total bandwidth reduction vs naive PQC gossip (Section 9)

For applications where this latency is acceptable, the security and decentralization gains justify the cost. Where sub-second confirmation is required, QBit supports optimistic pre-confirmation:

9.2.1 Pre-Confirmation Mechanism

For latency-sensitive applications—payment terminals, DEX trades, retail point-of-sale—QBit provides **soft pre-confirmations** with economic guarantees:

Definition 9.1 (Sentry Bond for Pre-Confirmations)**.** The Sentry's pre-confirmation bond (SentryBond) is the same as the proving bond defined in Section 7.1: $B_{prove} = 10{,}000$ QBIT. This bond secures both proving obligations and pre-confirmation commitments.

9.2.2 Pre-Confirmation Economic Security

Definition 9.2 (Pre-Confirmation Bond)**.** All Sentries use the standard proving bond: $B_{\text{prove}} = 10,000$ QBIT.

SafeAmount Calculation:

$$\text{SafeAmount} = \frac{B_{\text{prove}}}{10 \times N_{\text{preconf}}} = \frac{10{,}000}{10 \times 2000} = 0.5 \text{ QBIT} \tag{129}$$

For transactions > 0.5 QBIT, users should wait for full STARK finalization (≈ 3s).

This single-tier model is calibrated around the expectation that QBIT will trade at a price where 0.5 QBIT covers retail transactions (e.g., at \$10/QBIT, SafeAmount = \$5; at \$100/QBIT, SafeAmount = \$50). For higher-value transfers, users wait for full consensus finality (≈ 3s)—a negligible delay. Future governance upgrades may introduce multi-tiered bonding for institutional pre-confirmations.

Sentry Slashing:

Sentry is slashed 0.1% of bond (10 QBIT) per broken pre-confirmation. If a Sentry violates commitments for an entire batch (2000 transactions), the total penalty would be $2000 \times 10 = 20,000$ QBIT, which exceeds the total bond ($10,000$ QBIT). Thus, the Sentry loses their entire stake.

Economic Security Guarantee:

For an adversarial attack to be profitable:

$$L_{\text{users}} > S_{\text{actual}} \tag{130}$$

But we have (per slot limit):

$$L_{\text{users}} = \text{SafeAmount} \times N_{\text{preconf}} = 0.5 \times 2000 = 1000 \text{ QBIT} \tag{131}$$

Since 1000 QBIT $< 10,000$ QBIT (Bond), the attack is economically irrational.

User Guidelines:

For transactions with value V:

1. If $V \leq 0.5$ QBIT: Pre-confirmation (soft finality) is secure.
2. If $V > 0.5$ QBIT: Wait for full STARK finalization (≈ 3s).

Protocol 9.1 (MEV-Resistant Pre-Confirmation)**.** For MEV-sensitive transactions, QBit provides an optional encrypted submission path using post-quantum committee encryption. Rather than distributing a lattice-based decryption key (true threshold KEM remains an open research problem for lattice schemes), the construction operates at the symmetric-key layer: a fresh symmetric key is secret-shared and each share is independently encapsulated under individual committee members' public keys.

Setup. An Encryption Committee $\mathcal{C}_{TE}$ of $n_{te} = 21$ validators is selected per epoch. Each member publishes an ML-KEM-768 encapsulation key. Reconstruction threshold: $t = 12$ of 21. The threshold $t = 12$ ensures that an adversary controlling fewer than 12 committee members learns nothing about the transaction (information-theoretic security of Shamir's scheme), while with $\beta < 1/3$ adversarial stake, at least 14 members are honest, ensuring reconstruction liveness.

Encrypt-Order-Decrypt Flow:

1. User generates a symmetric key k, encrypts the transaction with AES-256-GCM, secret-shares k via $(12, 21)$ Shamir's scheme, and encapsulates each share under the corresponding committee member's ML-KEM-768 key. Each encapsulation is a standard single-recipient IND-CCA2 operation.

2. Sentry includes the encrypted commitment in its batch and issues a pre-confirmation receipt binding position. The Sentry cannot inspect the payload. The STARK proof commits only to the encrypted ciphertext hash, not to the plaintext, so encrypted transactions impose no additional proving cost.

3. After batch ordering is finalized (≈ 1 round), committee members decapsulate their ML-KEM ciphertext to recover their key share, then publish the decrypted share. Any party with ≥ 12 shares reconstructs k and decrypts.

Overhead: ≈ 23 KB per encrypted transaction (21 ML-KEM-768 ciphertexts at 1,088 B each = 22,848 B, plus encrypted Shamir shares). Applies only to Tier 1 (encrypted) transactions; Tier 2 (plaintext) transactions have no overhead.

Security: ML-KEM-768 provides NIST Level 3 security per encapsulation. Shamir's scheme is information-theoretically secure: fewer than t shares reveal zero information about k, regardless of the adversary's computational power (including quantum). An adversary controlling < 7 committee members ($< 1/3$) obtains fewer than $t = 12$ shares and learns nothing about k.

Security Considerations: Pre-confirmations are *optimistic*—they rely on economic incentives (slashing) rather than cryptographic finality. The committee encryption layer uses only post-quantum primitives (ML-KEM-768, Shamir's secret sharing, AES-256-GCM). Applications requiring trustless guarantees should wait for full STARK-based finalization.

9.3 Storage Scalability

9.3.1 The State Bloat Challenge

At a sustained throughput of 30,000 TPS, QBit generates approximately 6.65 MB/s of consensus data over the wire (STARK proofs at 4.4 MB/s and compressed state diffs at 2.25 MB/s; see Table 9). However, validators must store the *uncompressed* state diffs (since the STARK proof commits to $H(\Delta_{raw})$), yielding ≈ 11.15 MB/s on disk ($4.4 + 6.75$). Without pruning, this results in storage bloat of ≈ 0.96 TB per day. To ensure the validation layer remains strictly decentralized and accessible to mid-range server hardware, QBit implements an aggressive State Pruning mechanism via Recursive Epoch Compression.

9.3.2 Recursive Epoch Compression (Mina-Style Checkpointing)

QBit borrows the recursive proof idea from Mina [16], but diverges in a key respect. Mina achieves constant-size blockchain state by recursively verifying *every* block transition inside a single SNARK, which requires the proof system itself to be efficiently verifiable within its own circuit—a constraint that currently ties Mina to elliptic-curve assumptions (the Pasta cycle of curves, Pallas and Vesta) with no clear post-quantum migration path, since Shor's algorithm breaks the discrete logarithm problem on which these curves rely. QBit sidesteps this by performing recursive verification only at epoch boundaries (hourly, not per-block) and using hash-based STARKs throughout, accepting larger intermediate storage in exchange for a clean PQC foundation.

Let $T_{epoch} = 1$ hour. During an epoch at target capacity, the DAG produces $\approx 3{,}600$ consensus rounds (one per second), each with a quorum certificate (QC) attesting to the finalized batch proofs. Rather than recursively verifying all $\approx 108{,}000$ finalized batch proofs—which would be computationally infeasible within the epoch window—QBit aggregates the consensus *certificates*, leveraging the fact that each batch proof was already verified by $> 2/3$ of validators during consensus.

Protocol 9.2 (Two-Phase Epoch Compression)**. Phase 1: Consensus-Finalized State Checkpoint.** At the epoch boundary, the active state root StateRoot_e is finalized via the

existing 3-chain consensus. The 3-chain guarantees that $> 2/3$ of validators independently verified each batch proof before finalization. This consensus finality—not a recursive proof—is the primary justification for pruning individual batch proofs.

Pruning Rule: After a batch proof's epoch is finalized AND the Phase 2 epoch proof covers the batch, validators may prune the individual batch proof. Until Phase 2 completes, individual batch proofs are retained.

Phase 2: Epoch Proof via Certificate Aggregation (Deferred). The epoch proof recursively aggregates consensus round certificates rather than individual batch proofs:

- **Recursive Verifier Input:** Previous epoch proof $\pi_{epoch_{e-1}}$ (or genesis for $e = 0$), $\approx$ 3,600 consensus round certificates (QC hashes + validator attestation roots), and the state transition $\text{StateRoot}_{e-1} \to \text{StateRoot}_e$.
- **Recursive Depth:** $\lceil \log_2(3{,}600) \rceil = 12$ layers (vs. 18 for 180,000 batch proofs).
- **Per-Node Circuit:** Each tree node verifies two child proofs plus a QC attestation hash ($\approx 2 \times 10^5$ constraints per recursive step).

Timing Estimate (4× RTX 4090):

- Leaf layer: 1,800 verifications / 4 GPUs = 450 per GPU $\times \approx$ 1s $\approx$ 8 minutes.
- Subsequent 11 layers: $\approx$ 4,000 proofs total $\approx$ 8–15 minutes.
- Total: $\approx$ 15–25 minutes wall-clock time, well within the $T_{agg} = 2$ hour deadline.

Recursive Chaining: Formally, the public input for π_{epoch_e} includes the hash of the previous state root StateRoot_{e-1}, chaining each epoch's proof to every prior epoch back to genesis:

$$\text{Verify}(\pi_{epoch\_e}) \implies \text{Verify}(\pi_{epoch\_e-1}) \implies \cdots \implies \text{Genesis}$$

Phase 3: Light Client Epoch Proof (Optional). For cross-chain bridging and light client verification, a compact epoch proof can be extracted from the Phase 2 output. A light client needs only the latest epoch proof ($\approx$ 200–300 KB) to verify the entire chain history without downloading individual block headers, batch proofs, or consensus certificates.

9.3.3 Checkpoint Aggregator Specification

Definition 9.3 (Checkpoint Aggregator Selection)**.** Checkpoint Aggregators are selected from the active Sentry set:

1. **Eligibility:** Any Sentry that produced $\geq 50\%$ of its expected proofs during epoch e is eligible
2. **Committee Size:** $N_{agg} = 10$ Sentries, selected by $H(\text{seed}_e \| \text{“aggregator”})$
3. **Competition:** All eligible aggregators race to produce π_{epoch_e}; first valid proof accepted

Incentive Structure:

- **Aggregation Reward:** The winning aggregator receives $R_{agg} = 1\%$ of total epoch block rewards, funded from the protocol treasury (Section 7.3)
- This reward is *in addition to* normal proving rewards, compensating the substantial computation required. The aggregation uses a binary tree structure over $\approx$ 3,600 consensus certificates ($\lceil \log_2(3{,}600) \rceil = 12$ layers), where each layer's verifications can be parallelized across GPU cores. At each tree node, one STARK proof verifies two child proofs and a QC

attestation hash ($\approx 2 \times 10^5$ constraints). With 4× RTX 4090 GPUs, the full 12-layer tree completes in $\approx$ 15–30 minutes wall-clock time. This computation occurs once per epoch (1 hour) and is entirely off the consensus critical path—it does not affect transaction finality or throughput

Liveness Deadline:

- Aggregators have $T_{agg} = 2$ hours (2 epochs) to produce the epoch proof
- If no valid π_{epoch_e} is submitted within T_{agg}, the epoch boundary is **extended**: subsequent batch proofs continue to accumulate, and the aggregation window resets
- If aggregation fails for > 3 consecutive epochs, a governance alert triggers parameter review (e.g., increasing N_{agg} or reducing E_N via shorter epochs)
- **Validators do not prune** batch proofs until the corresponding epoch proof is finalized

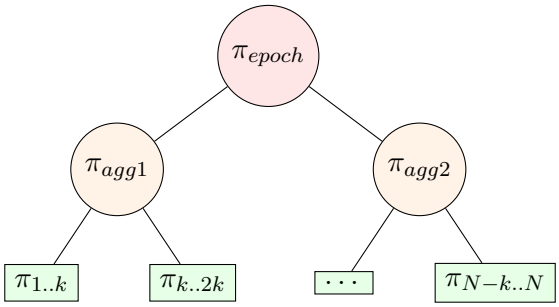

Figure 3: Recursive Epoch Compression Tree

9.3.4 Explicit Separation of State and History

The protocol uses the pruning specification to enforce the persistence of the Active State Tree.

Definition 9.4 (State Retention Invariant)**.** Validators MUST distinguish between **Active State** and **Historical Trace**:

1. **Active State ($\Sigma_{current}$):** The mapping Address $\rightarrow$ (Balance, Nonce, StorageRoot, CodeHash).

- **Retention Policy:** `PERMANENT`.
- Validators must persist the full Poseidon2 Merkle Tree for the current state root. Pruning any part of $\Sigma_{current}$ constitutes a Liveness Fault.

2. **History ($\Pi_{historical}$):** The sequence of proofs and diffs that led to $\Sigma_{current}$.

- **Retention Policy:** `PRUNABLE` after Epoch Finalization.
- Historical data is offloaded to the Data Availability (DA) layer and specialized Archival Nodes.

This distinction ensures that a smart contract deployed in epoch e remains executable in epoch $e + n$, regardless of how much historical proof data is pruned.

10 QChain Framework

10.1 Overview

QBit operates at genesis as a single high-performance post-quantum blockchain (`chain_id` = 0). The QChain framework is present in the protocol from genesis but dormant—all infrastructure exists, no additional QChains are active. QChains join through governance as the Sentry network matures, inheriting QBit's security and finality without protocol disruption.

The framework rests on a single architectural primitive: every STARK proof carries a `chain_id` identifying its originating chain (Definition 2.2). QBit's Sentry network proves execution traces from any registered chain using identical infrastructure. Validators finalize proofs from all chains through the same Helix-ZK DAG. The shared DA layer stores data from all chains indistinguishably.

10.2 What QChains Inherit

Any registered QChain automatically inherits:

- ML-DSA-65 post-quantum signature security for user transactions
- STARK-proven execution validity (hash-based, no trusted setup)
- Helix-ZK DAG finality (≤ 3 second consensus finality)
- FRI-based post-quantum data availability
- Cross-chain messaging via the shared DAG (Section 13)

10.3 What QChains Control

QChains are sovereign in execution environment (any deterministic VM expressible as an AIR trace), internal consensus mechanism (if any—a sequencer-based QChain is valid), token economics and fee structure, governance model, state rent policy, and application logic. The only technical requirement is implementing the Provable Execution Interface (Section 11).

10.4 Security Boundaries

QBit provides cryptographic guarantees about execution validity. It does not provide guarantees about the correctness of a QChain's execution rules.

QBit guarantees (for any registered QChain): all transactions in a proven batch were faithfully executed per the declared PEI runtime; the execution trace represents the declared state transition (STARK soundness); the new state root correctly reflects the execution result; outbound messages in `MsgRoot` were produced by the declared execution.

QBit does NOT guarantee: the QChain's execution rules are economically fair or logically correct; the QChain's internal consensus (if any) is Byzantine fault tolerant; the QChain's token has value or its economics are sound. Users of a QChain must evaluate these properties independently.

10.5 Growth Model

QBit launches as a complete, self-sufficient blockchain. The Sentry network grows organically from native chain transaction volume. As Sentry depth increases, the framework becomes progressively safer for QChain activation—more Sentries means more proving redundancy per chain. QChains activate when the ecosystem is ready, not on a predetermined schedule. The native chain has full independent utility regardless of whether QChains materialize.

11 Provable Execution Interface (PEI)

11.1 Definition

The Provable Execution Interface is the single technical contract between a QChain and the QBit framework. Any execution environment implementing the PEI can have its transactions proven by the Sentry network and finalized by the Helix-ZK DAG.

The PEI's `execute` function takes a vector of signed transactions, the previous state root, and the slot seed, and returns an `ExecutionResult` containing the new state root, the state diff, an AIR execution trace over the Goldilocks field, and a vector of outbound cross-chain messages.

11.2 AIR Trace Requirements

The execution trace must conform to QBit's standard Algebraic Intermediate Representation format:

- Defined over the Goldilocks field $\mathbb{F}_p$ ($p = 2^{64} - 2^{32} + 1$)
- Trace width and length within the limits of Table 5 (max 2^{26} steps)
- No floating-point arithmetic anywhere in the trace
- All state reads committed before execution begins (required for conflict detection per Section 4.3)

Any deterministic computation expressible within these constraints is a valid QChain execution environment. QBit's QVM (Appendix G) is the reference implementation—its AIR trace format defines the standard.

11.3 Signature Handling

QBit native (`chain_id` $= 0$) proves ML-DSA-65 signature verification inside the STARK circuit for every transaction. QChains may follow this same model or adopt alternatives consistent with their sovereignty:

- **Per-transaction in-circuit verification** (recommended): identical to QBit native. The QChain's PEI includes ML-DSA-65 verification constraints. Highest per-transaction security.
- **Sequencer model**: a single authorized key signs the batch; one ML-DSA-65 verification per batch rather than per transaction. Lower constraint cost, weaker per-transaction guarantees. Suitable for permissioned or high-throughput QChains.

The QChain's choice is encoded in its PEI runtime. QBit's STARK proves that whatever model the QChain chose was faithfully executed—the proof attests to execution validity, not to any particular signature policy.

11.4 State Root Computation

QBit's Poseidon2 Merkle tree (Section 4.2) is the reference implementation and recommended for constraint efficiency (≈ 350 constraints per hash). QChains may use alternative in-circuit hash functions for their internal state roots at their own constraint cost. The state root computation must be proven inside the execution trace—it is part of the QChain's AIR program.

11.5 Runtime Distribution

QChain runtimes are distributed as deterministic WASM binaries. The Sentry downloads the runtime from the QChain's designated distribution endpoint, verifies SHA3-256(`0x16`‖binary) = `runtime_hash` (where `runtime_hash` is recorded in the Chain Registry), and executes in an isolated WASM sandbox with the same determinism constraints as QVM (no floating point, no nondeterministic instructions). Runtime upgrades require updating `runtime_hash` in the Chain Registry via QBit governance.

11.6 Batch Submission for Sovereign QChains

For QChains with their own consensus, the interaction is:

1. QChain consensus produces a finalized ordered batch
2. QChain submits the sealed batch to the assigned Sentry
3. Sentry executes via the PEI runtime, generates the execution trace
4. Sentry produces the STARK proof and submits to the DAG

The Sentry does not reorder, filter, or modify the batch contents—the QChain's consensus is authoritative for transaction ordering on that chain.

12 Chain Registry

The Chain Registry is a smart contract deployed on QBit native (`chain_id` = 0) at genesis. It is the authoritative record of all active QChains. At genesis it contains exactly one entry: QBit native itself.

12.1 Permissionless Registration

QChain registration is permissionless. Any party can register a new QChain by submitting a registration transaction on QBit native that includes:

- Genesis state root and PEI runtime hash (SHA3-256(`0x16`‖binary))
- A registration bond of $B_{\text{chain}} = 50{,}000$ QBIT, locked for the lifetime of the chain
- First year's chain rent of $R_{\text{chain}} = 5{,}000$ QBIT, burned immediately

The registry assigns the next available `chain_id` (u16). No governance vote is required. The security checks that matter—Sentry coverage, solvency invariant, STARK soundness—are all algorithmic, not political.

Activation. A registered QChain activates when at least $N_{\text{min\_sentries}} = 5$ Sentries have registered support for its `chain_id`. Until this threshold is met, the chain exists in the registry but cannot submit proofs.

Annual Chain Rent. Each active QChain pays $R_{\text{chain}} = 5{,}000$ QBIT per year, burned on collection. Rent is payable by any address—the QChain's operator, a DAO treasury, or community sponsors. This creates recurring deflationary pressure proportional to the number of active QChains and prevents abandoned chains from squatting on `chain_id` slots indefinitely.

Deregistration. Voluntary deregistration requires a 90-day notice period enforced on-chain. During the notice period, the chain continues to be proven normally and users may bridge assets to other chains. After 90 days, the `chain_id` is deactivated, proofs carrying it are rejected, and the registration bond (B_{chain}) is returned. If rent is overdue at deregistration, the outstanding rent is deducted from the bond before return.

12.2 Registry State

Each registered chain carries a record containing: `chain_id` (u16), status, genesis state root, PEI runtime hash, registration epoch, bond amount, and rent-paid-through epoch.

Status Transitions:

- **Active**: fully operational. Proofs accepted, cross-chain messaging enabled.
- **Degraded**: below $N_{\text{min\_sentries}}$ threshold OR rent overdue > 90 days. Proofs still accepted from remaining Sentries, but outbound cross-chain messages are suspended. Inbound messages continue processing. If Degraded for 720 consecutive epochs (30 days) due to insufficient Sentries, or if rent is overdue for 180 days, the chain enters Retiring status automatically.
- **Retiring**: 90-day deregistration countdown active. Chain continues to be proven. Users may bridge assets. After 90 days, the `chain_id` is deactivated. Bond is returned minus outstanding rent.

12.3 Validator Enforcement

Validators reject any submitted STARK proof where `chain_id` is not present in the Chain Registry with status `Active` or `Degraded`. This is a stateless check against the registry contract. An unregistered `chain_id` in a proof's public inputs is treated identically to an invalid proof.

12.4 Governance Powers

Governance does not control registration. It retains two powers:

1. **Force-deregistration**: if a QChain's PEI is discovered to have a critical flaw that the solvency invariant alone cannot contain. This is the emergency brake, requiring 2/3 supermajority. The bond is slashed (burned) in this case.
2. **Parameter adjustment**: modifying B_{chain}, R_{chain}, $N_{\text{min\_sentries}}$, or $T_{\text{deregister}}$ via the standard governance process ($\pm 20\%$ per action, 90-day cooldown).

12.5 Solvency Invariant

To prevent a buggy or malicious QChain PEI from draining funds belonging to other chains, the epoch compression proof (Section 9.3.2) includes a per-chain balance invariant:

$$\forall\ \texttt{chain\_id}\ C : \sum \text{deposits\_to}(C) \geq \sum \text{withdrawals\_from}(C) \tag{132}$$

This is verified inside the recursive epoch proof at negligible constraint cost (≈ 100 constraints per chain). Even if a QChain's execution is technically "valid" per its PEI, the solvency invariant prevents it from creating tokens for cross-chain withdrawal that were never deposited. Violation of the solvency invariant triggers automatic force-deregistration and bond slashing without requiring a governance vote.

13 Cross-Chain Transactions

13.1 Overview

Cross-chain transactions allow QChains to reference the finalized state of other QChains. The mechanism requires no light clients, no relayers, and no separate messaging protocol. The shared

Helix-ZK DAG serves as the trust anchor—a proof finalized in the DAG is accessible to all chains as a first-class cryptographic object.

During single-chain operation, cross-chain transactions are defined but never triggered. The type definition and verification logic are present from genesis, adding zero runtime overhead until a second QChain activates.

13.2 Verification

When a QChain's Sentry proves a batch containing a cross-chain transaction referencing a message from chain A, the execution trace verifies:

1. Chain A is registered and Active in the Chain Registry
2. The source batch exists in FinalizedRoots[A]—already finalized in the DAG
3. The message's Merkle path is valid against the source proof's `MsgRoot`
4. The channel nonce equals the expected next nonce for the (A, B) channel
5. Current slot $\leq$ `deadline_epoch`

Steps 1–2 are registry and DAG lookups. Steps 3–5 are local computations. No communication with the source chain or any external party is required.

13.3 Replay Protection

Each ordered pair (source_chain, dest_chain) maintains an independent strictly-incrementing nonce sequence. The destination chain tracks:

$$\text{ChannelState} : (\texttt{ChainId}, \texttt{ChainId}) \to \texttt{u64} \tag{133}$$

A cross-chain transaction with a nonce not equal to ChannelState[(source, dest)] is rejected. This is identical to how QBit handles account nonces for intra-chain replay protection.

13.4 Timeout Handling

Cross-chain messages carry a `deadline_epoch` expressed in shared-consensus epochs. If the destination chain has not consumed the message by `deadline_epoch`, the shared consensus marks it as timed out and the source chain's application logic handles recovery (typically releasing locked funds).

As a fallback against destination chain liveness failures: if FinalizedRoots[dest_chain] has not advanced for $T_{\text{chain\_timeout}} = 24$ hours, all pending cross-chain messages targeting that chain become eligible for timeout recovery on the source chain regardless of their individual deadlines. This protects against destination chain halts without requiring any proof of non-inclusion from the halted chain—the shared DAG provides global visibility into all chains' finalized state.

13.5 Cross-Chain Fee Model

Cross-chain transactions carry a 5× base fee surcharge (the "cross-chain settlement fee"), always denominated and burned in QBIT regardless of what gas token the QChain uses internally for intra-chain transactions. The user pays a single fee on the source chain; the protocol handles settlement:

- Source chain execution fee: standard intra-chain gas cost
- Cross-chain settlement fee: $5 \times$ BaseFee, burned in QBIT

- Destination chain execution fee: paid from the message's attached gas budget

This creates direct deflationary pressure on QBIT proportional to cross-chain activity. A QChain may use its own gas token for intra-chain transactions, but the registration bond (B_{chain}), annual rent (R_{chain}), and cross-chain settlement fees are always denominated in QBIT—making QBIT operationally mandatory for any chain participating in the framework.

13.6 Single-Chain Operation

When only `chain_id` $= 0$ exists, cross-chain transactions are a defined but unreachable transaction type. No QChain exists to receive messages from. The type's presence in the protocol adds zero runtime overhead. When the first QChain activates, cross-chain transactions become valid without any protocol change.

14 Conclusion

QBit moves signature verification off the consensus hot path and into a competitive prover market, recovering the bandwidth and compute headroom that lattice signatures consume. Validator ingress drops from over 6 Gbps to under 250 Mbps on commodity hardware.

Every cryptographic component—signatures, hashing, randomness, proofs, key encapsulation, and secret sharing—is post-quantum secure. No part of the protocol depends on elliptic curves, pairings, or factoring assumptions.

The tradeoff is real: we introduce a specialized hardware tier (Sentries) and accept the economic complexity of a proof mining market. Whether this tradeoff ages well depends on GPU price trajectories and the maturation of STARK proving systems. We believe the trend favors it.

The QChain framework extends this infrastructure to sovereign application-specific chains without additional trust assumptions. Because every proof carries a `chain_id` and the Helix-ZK DAG finalizes all chains indistinguishably, cross-chain messaging reduces to Merkle path verification against finalized state roots—no bridges, no relayers, no challenge periods. The framework is present from genesis but dormant; QChains activate through governance as the Sentry network matures.

The most significant open problems are recursion overhead—epoch-level certificate aggregation takes 15–30 minutes on a 4-GPU cluster using the two-phase approach—and residual hot-contract state conflicts under DeFi-heavy workloads. The adoption of Poseidon2 with 512-bit sponge capacity provides unambiguous 128-bit post-quantum in-circuit hash security, making the recursive STARK composition (120 bits) the sole protocol-level bottleneck—closeable via increased FRI query count in a future parameter upgrade. Future work on improved FRI folding, speculative execution with rollback, or sharded contract execution may close the remaining performance gaps.

A Mathematical Formalization

A.1 Notation Conventions

To avoid ambiguity, polynomial ring notation is clarified:

- $R = \mathbb{Z}[X]/(X^{256} + 1)$: Polynomial ring (integers, cyclotomic)
- $R_q = \mathbb{Z}_q[X]/(X^{256} + 1)$: Quotient ring modulo q
- Elements: $\mathbf{a} \in R_q$ denotes $\mathbf{a}(X) = \sum_{i=0}^{255} a_i X^i$ where $a_i \in \mathbb{Z}_q$

Throughout the paper, R_q and $\mathbb{Z}_q[X]/(X^{256}+1)$ are used interchangeably - both refer to the same structure.

Matrix-vector operations are performed coefficient-wise in R_q.

A.2 Lattice Cryptography Foundations

The Module-LWE and Module-SIS problems underpin the post-quantum security guarantees.

Definition A.1 (Module-LWE Distribution)**.** For integers n, q, k and error distribution χ over R_q, the Module-LWE distribution $L_{n,q,\chi}$ samples $(\mathbf{A}, \mathbf{t})$ where:

$$\mathbf{A} \leftarrow U(R_q^{k\times n}), \quad \mathbf{s} \leftarrow \chi^n, \quad \mathbf{e} \leftarrow \chi^k$$

$$\mathbf{t} := \mathbf{A}\mathbf{s} + \mathbf{e}$$

The search M-LWE problem is to recover $\mathbf{s}$ from $(\mathbf{A}, \mathbf{t})$.

Theorem A.1 (Bit Security of ML-DSA-65)**.** *Based on the Core-SVP hardness methodology, ML-DSA-65 achieves:*

$$\textit{Security} \geq 128 \textit{ bits (quantum)}$$

$$\textit{Security} \geq 197 \textit{ bits (classical)}$$

Approximated by the cost of running Dual-BKZ with blocksize $\beta = 600$.

A.3 Hash-Based Leader Election

Definition A.2 (Hash-Based Leader Election)**.** Helix-ZK employs a deterministic, hash-based leader election mechanism. Since the SWF (Section 5.2.1) already provides unbiasable epoch randomness seed_e, leader selection does not require a separate Verifiable Random Function (VRF). Instead, eligibility is computed as:

$$r = H(\text{seed}_e \| \text{slot}_i \| pk_{val})$$

where H is SHA3-256 and pk_{val} is the validator's registered public key.

Security properties:

- **Unpredictability:** Without the SWF output seed_e, the election outcome is indistinguishable from random (reduces to SHA3-256 pseudorandomness).
- **Unbiasability:** An adversary controlling $< 1/3$ stake cannot bias the SWF output (sequentiality guarantee), and therefore cannot bias leader selection.
- **Determinism:** For fixed $(\text{seed}_e, \text{slot}_i, pk_{val})$, exactly one output exists. All honest nodes compute identical results.
- **Public Verifiability:** Any party can independently verify any validator's eligibility by computing the same hash.

Because this construction relies exclusively on SHA3-256 and the SWF's sequentiality (iterated Poseidon2; Section 5.2.1)—both hash-based primitives with post-quantum security—no VRF-specific cryptographic assumptions (elliptic curves, lattice problems, or pairings) are required.

A.4 ZK-STARK Constraints (AIR)

The Algebraic Intermediate Representation (AIR) for the Sentry's proof generation is formalized below. The computation trace $\mathcal{T}$ represents the execution of the verification loop.

$$\mathcal{T} \in \mathbb{F}_p^{W\times T} \tag{134}$$

where W is the trace width (number of registers) and T is the trace length (steps).

A.4.1 Constraint: Signature Validity

For each transaction i, the transition constraint $C_{sig}(i)$ is defined to ensure the lattice equation holds:

$$\mathbf{w}_1 = \text{HighBits}(\mathbf{Az} - c\mathbf{t})$$

Since ZK-STARKs operate over finite fields $\mathbb{F}_p$, range checks are required to emulate $\mathbb{Z}_q$. The protocol strictly uses the bit-decomposition gadget defined in Section 11.1, as naive polynomial constraints $(x(x-1)...(x-(q-1))=0)$ are computationally infeasible for large q.

A.4.2 Constraint: Merkle Path Validity

To prove inclusion of tx_i in the batch root, hash consistency is enforced:

$$h_{parent} = \text{Poseidon2}(h_{left}, h_{right})$$

where Poseidon2 uses width $t = 16$ with S-box x^7 over the Goldilocks field, as specified in Section 3.1. This constraint applies to all rows corresponding to Merkle hashing steps.

B Partition Healing Proof

This appendix provides the full proof of Lemma 2.3 (Fallback Vote Consistency Under Partial Network Degradation) and the associated healing procedures.

Proof of Lemma 2.3. Under partial network degradation, the validator set partitions into overlapping communication groups P_1 and P_2, where $P_1 \cap P_2 \neq \emptyset$. By the BFT quorum requirement, at most one group can independently form a $> 2/3$ quorum.

Case 1 (Single quorum): If $|P_1| > \frac{2}{3}S_{\text{total}}$, then P_1 finalizes the fallback decision; validators in $P_2 \setminus P_1$ adopt it upon healing. No conflict arises.

Case 2 (No quorum): If neither group independently holds $> \frac{2}{3}$ stake, no decision is finalized. Safety is preserved; liveness resumes upon healing.

Case 3 (Transitive relay): Validators in $P_1 \cap P_2$ may relay votes, forming a combined quorum $|P_1 \cup P_2| > \frac{2}{3}S_{\text{total}}$. The resulting decision is unique by quorum intersection.

If a DAG fork occurs, the canonical branch is selected by $\arg\max$ EffectiveStakeSupport. Validators on the losing branch roll back to the last common ancestor and re-apply the canonical history.

Lock monotonicity (Lemma 5.4) prevents conflicting votes: if V_i voted for vertex v in round r, it cannot subsequently vote for any vertex contradicting a finalized v' in round $r' > r$. After healing, the $> \frac{2}{3}$ honest stake converges on the canonical history. □ □

Protocol B.1 (Partition Healing Procedure)**.** Upon detecting a DAG fork during partition healing:

1. **Identify Fork Point:** Validator V_i identifies the last common DAG vertex v_{common} and collects vertices from both partitions.
2. **Apply Fork Choice:** Compute EffectiveStakeSupport for each branch; select $P_{\text{canonical}} = \arg\max(\text{EffectiveStakeSupport})$.
3. **Reorg (if necessary):** If V_i was in the losing partition, roll back to v_{common}, discard losing-branch vertices, and apply canonical vertices in topological order.
4. **Resume Consensus:** Continue producing vertices on the canonical branch.

C Detailed Algorithms

C.1 Sentry Prover Logic

The internal loop of a Sentry Node.

Algorithm 1 Sentry Prover Loop

Initialize: $Mempool \leftarrow \emptyset$
Initialize: $ProverKey, VerifierKey \leftarrow$ SetupSTARK()
loop
  $Tx \leftarrow$ ListenForTransactions()
  if ValidateSignature(Tx) **then**
    $Mempool$.insert(Tx)
  end if
  if $Mempool$.size() ≥ 1000 **then**
    $\mathcal{B} \leftarrow Mempool$.drain(1000)
    Trace, Witness $\leftarrow$ ExecuteBatch($\mathcal{B}$)
    $\pi \leftarrow$ STARK.Prove($ProverKey$, Trace, Witness)
    $\Delta \leftarrow$ ComputeStateDiff($\mathcal{B}$)
    BroadcastProof(π, Δ)
  end if
end loop

C.2 Validator DAG Logic

The consensus logic for validators.

Algorithm 2 Validator Consensus Loop

Input: Incoming Proofs Π
State: $DAG \leftarrow$ GenesisBlock
State: $State \leftarrow$ GenesisState
loop
  $Block \leftarrow$ ReceiveBlock(Π)
  if STARK.Verify($VerifierKey, Block.\pi$) **then**
    DAG.append($Block$)
    UpdateVirtualVotes(DAG)
    if IsRatified($Block$) **then**
      $State \leftarrow$ ApplyDiff($State, Block.\Delta$)
      CommitStateRoot($State$)
    end if
  else
    Discard($Block$)
  end if
end loop

D Threat Model and Security Analysis

D.1 Quantum Adversary Model

The model assumes an adversary $\mathcal{A}$ equipped with a Cryptographically Relevant Quantum Computer (CRQC).

- **Shor's Algorithm:** $\mathcal{A}$ can factor integers and compute discrete logs in polynomial time. ECC (secp256k1, Ed25519) is broken.
- **Grover's Algorithm:** $\mathcal{A}$ can invert hash functions in $O(\sqrt{N})$.

QBit Defense:

- **Signatures:** ML-DSA-65 (FIPS 204) is based on Module-Lattice problems (Hardness of SVP), which are resistant to Shor.
- **Key Encapsulation:** ML-KEM-768 (FIPS 203), used for MEV-resistant committee encryption (Section 9.2.1), is based on Module-LWE, also resistant to Shor.
- **Hashes:** SHA3-256 provides 128 bits of post-quantum security against Grover (256-bit output).
- **Secret Sharing:** Shamir's Secret Sharing is information-theoretically secure—immune to any computational advance, classical or quantum.
- **Symmetric Encryption:** AES-256-GCM provides 128-bit post-quantum security (Grover halves effective key length).

Every cryptographic primitive in the protocol—ML-DSA for signatures, ML-KEM for key encapsulation, SHA3-256 and Poseidon2 for hashing, iterated-Poseidon2 SWF for randomness, hash-based STARKs for proofs, and Shamir's scheme for secret sharing—is secure against both classical and quantum adversaries. No component relies on integer factorization, discrete logarithms, or elliptic-curve assumptions.

D.2 Hash Function Security

QBit uses two hash functions, SHA3-256 and Poseidon2, with strictly separated domains (see Section 3.1, Table 3 for the full domain separation mapping):

Usage	**Function**	**Security**
Block hashes, DA Merkle, Leader election	SHA3-256	128-bit (post-quantum)
SWF	Iterated Poseidon2	ASIC-Resistant Sequentiality
State Merkle tree, In-circuit batch/tx hashing	Poseidon2 [22]	128-bit (256-bit sponge, BBLP22-resista

Table 12: Hash Function Usage Summary

D.3 Long-Range Attacks and Weak Subjectivity

In a Proof-of-Stake system, an adversary might attempt to rewrite history from a point where they held a majority of the stake.

- **Epoched Key Rotation:** Validators must use Epoched Key Rotation. New keys are generated for each epoch, signed by the previous epoch's key ("Certificate of Handover"), and the old Private Key is securely deleted. This achieves Forward Secrecy using standard ML-DSA-65.
- **Weak Subjectivity:** QBit is explicitly a Weakly Subjective consensus protocol. New nodes must boot from a trusted checkpoint hash not older than the unbonding period (21 days). This is a fundamental trade-off for long-range security in PoS.

E Governance and Parameters

E.1 QBit DAO

Governance is decentralized via the QBit DAO.

- **Proposal Threshold:** 100,000 QBIT to submit a QIP (QBit Improvement Proposal).
- **Voting Period:** 7 Days.
- **Quorum:** 4% of total supply.

E.2 System Parameters

Parameter	Value	Description
T_{slot}	12s	Proof submission interval
T_{block}	400ms	Block (DAG vertex) creation target
T_{epoch}	1 hour	Checkpoint / epoch interval
T_{delay}	300s	SWF computation delay
$T_{cooldown}$	10 epochs	Minimum cooldown between fallback transitions
N_{val}	> 100	Minimum validator set size
N_{SWF}	100	SWF committee size
$N_{leaders}$	5	Proving slot leaders per slot
N_{agg}	10	Checkpoint aggregator committee size
B_{prove}	10,000 QBIT	Sentry proving bond
S_{min}	10,000 QBIT	Minimum stake to validate
k	30	DA sampling queries per block
n_{te}	21	Threshold Encryption Committee size
t_{te}	12	Threshold decryption quorum
D_{mesh}	6	GossipSub mesh degree (control channel)
b_{tree}	3	Proof Relay Tree branching factor
$\texttt{chain\_id}_{\text{native}}$	0	QBit native chain identifier
$N_{\text{min\_sentries}}$	5	Minimum Sentry support for QChain activation
B_{chain}	50,000 QBIT	QChain registration bond (locked, refundable)
R_{chain}	5,000 QBIT/year	Annual QChain rent (burned)
$T_{\text{deregister}}$	90 days	Minimum QChain deregistration notice
$T_{\text{chain\_timeout}}$	24 hours	Cross-chain timeout fallback (chain liveness)

Table 13: System Constants

E.3 QChain Economics

QChain registration is permissionless (Section 12). The economic barrier—$B_{\text{chain}} = 50{,}000$ QBIT locked plus $R_{\text{chain}} = 5{,}000$ QBIT/year burned—provides Sybil resistance without governance gatekeeping. Governance retains only force-deregistration (emergency, 2/3 supermajority) and parameter adjustment ($\pm 20\%$ per action, 90-day cooldown). A QChain's internal execution rules are its own governance responsibility.

E.4 Protocol Treasury

A protocol tax of $\tau = 10\%$ on block rewards and transaction fees funds development, audits, and ecosystem grants. τ is adjustable via DAO vote within $[5\%, 20\%]$ with a 48-hour timelock. A portion of the base fee is burned (EIP-1559 style).

F AIR Constraints Table

F.1 Range Constraints

Naïve range enforcement via high-degree polynomials is computationally infeasible for large moduli. Instead, the protocol enforces bounded integer ranges using decomposition constraints.

Definition F.1 (Range Constraint Gadget)**.** Let $x \in \mathbb{Z}$ with bound $0 \leq x < 2^k$. This is enforced by decomposing:

$$x = \sum_{i=0}^{k-1} b_i 2^i \quad \text{where } b_i \in \{0, 1\}$$

Constraints:

$$b_i(b_i - 1) = 0 \quad \forall i \tag{135}$$

$$x - \sum_{i=0}^{k-1} b_i 2^i = 0 \tag{136}$$

For larger domains (e.g. ML-DSA coefficients mod q), radix-2^8 decomposition with lookup arguments is used to enforce byte-range correctness. This reduces constraint degree from $O(q)$ to $O(\log q)$.

Proof Size Bound STARK proof size is approximately:

$$|\pi| \approx O\left(\log T\right)$$

where the hidden constant depends on the soundness parameter and FRI query count, and:

- T = trace length
- ρ = blowup factor
- $|\mathbb{F}|$ = field size

For QBit's canonical parameters:

$$T \approx 2^{26}, \quad \rho = 8, \quad |\mathbb{F}| = 2^{64}, \quad k = 100$$

empirical implementations yield proof sizes of approximately 147 KB, consistent with the adjusted FRI parameterization (+0.5 KB/query).

Opcode	**Constraint Polynomial**	**Degree**
`ADD`	$X_{i,next} - (X_{i,a} + X_{i,b}) = 0$	1
`MUL`	$X_{i,next} - (X_{i,a} \cdot X_{i,b}) = 0$	2
`POSEIDON`	$S_{next} - \text{MDS} \cdot (S_{curr} + K)^\alpha$	7
`SIG_VERIFY`	Decomposed ML-DSA verification constraints	≤ 8

Table 14: Polynomial Constraints for AIR

G QBit Virtual Machine (QVM)

G.1 Execution Model

The QVM executes deterministic WebAssembly (WASM) bytecode with the following constraints:

- Floating-point arithmetic is disallowed; all math is integer or fixed-point.
- Nondeterministic instructions (e.g., `memory.grow` with OS interaction) are replaced with deterministic equivalents.
- Maximum memory per contract execution: 256 pages (16 MB).
- Maximum call stack depth: 1,024 frames.
- Maximum contract bytecode size: 256 KB.

G.2 Gas Schedule

The gas schedule follows EVM conventions for comparable operations:

- Base transaction cost: 21,000 gas.
- Cold storage read (`SLOAD` equivalent): 2,100 gas.
- Cold storage write (`SSTORE`, new value): 20,000 gas.
- Warm storage read: 100 gas.
- Memory expansion: quadratic cost per EVM formula.
- WASM opcode costs: calibrated via benchmark against the reference WASM interpreter, mapping each WASM opcode to an equivalent EVM gas cost.
- Poseidon2 hash invocation (in-contract): 525 gas (reflecting ≈ 350 constraints).
- SHA3-256 invocation (in-contract): 12,000 gas (reflecting $\approx 8{,}000$ constraints).

G.3 Block and Batch Gas Limits

Block gas limit: 50,000,000 per $T_{block} = 400$ms block. Per-batch gas limit: $50{,}000{,}000 \times 30 = 1{,}500{,}000{,}000$ (30 blocks per slot). Max trace length per batch: 2^{26} steps. Max recursion depth: $d_{\max} = 10$.

G.4 Scope of 30,000 TPS Claim

The 30,000 TPS target assumes a mixed workload:

- 70% simple transfers (21,000 gas each)
- 20% ERC-20-equivalent token transfers (65,000 gas)
- 10% contract interactions (150,000 gas)
- Weighted average: $\approx 44{,}000$ gas per transaction
- At 1,500,000,000 gas per batch: $1{,}500{,}000{,}000/44{,}000 \approx 34{,}000$ TPS

The full gas schedule and opcode mapping are maintained in the protocol specification (separate document).

H Network Architecture

QBit uses GossipSub v1.1 (libp2p) for control messages ($D = 6$) and the Proof Relay Tree (Section 2.6) for bulk data. Separate gossip topics partition blocks, proofs, DA samples, forced transactions, and consensus votes. Peers are scored by delivery rate and disconnected below threshold. Message priority: consensus votes > proofs > blocks > DA samples > transaction gossip. Backpressure is signaled when the proof verification queue exceeds 500 pending items. Connection limits: 50 inbound + 50 outbound per node. Peer discovery uses Kademlia DHT after initial bootstrap.

I Adversary and Network Model

I.1 Adversary Model

The model defines a probabilistic polynomial-time adversary $\mathcal{A}$ with capabilities:

- Controls up to $\beta < 1/3$ of validator stake
- Controls arbitrary number of non-staked Sybil nodes
- May delay or reorder network messages within Δ_{net}
- Has quantum computational capability (CRQC model)

The adversary is **adaptive** and **rushing**.

I.2 Network Model

The network model assumes partial synchrony:

$$\exists\, GST \text{ such that } \forall t > GST,\ \text{message delay} \leq \Delta_{net}$$

Before GST, network may be asynchronous.

I.3 Committee Size Optimization

The validator committee size $v = 256$ is derived based on statistical security:

Theorem I.1 (Committee Security)**.** *For committee size v, adversarial stake fraction $f < 1/3$, the probability that the adversary controls $\geq v/3$ committee members is:*

$$P_{fail} = \mathbb{P}\left[Binom(v, f) \geq \frac{v}{3}\right] \tag{137}$$

Using Chernoff bound:

$$P_{fail} \leq \exp\left(-v \cdot D_{KL}\left(\frac{1}{3} \| f\right)\right) \tag{138}$$

For $f = 0.2$ (conservative estimate):

$$D_{KL}(1/3 \| 0.2) \approx 0.0487 \tag{139}$$

$$P_{fail}(v = 128) \approx 2^{-9} \tag{140}$$

$$P_{fail}(v = 256) \approx 2^{-18} \tag{141}$$

$$P_{fail}(v = 512) \approx 2^{-36} \tag{142}$$

For $f = 0.3$ (near-threshold adversary, $\epsilon = 0.033$):

$$D_{KL}(1/3 \| 0.3) \approx 0.0026 \tag{143}$$

$$P_{fail}(v = 256) \approx 2^{-0.97} \tag{144}$$

$$P_{fail}(v = 512) \approx 2^{-1.9} \tag{145}$$

Security Analysis: At $f = 0.2$ (the expected operating range), $v = 256$ provides strong statistical safety (2^{-18}). As $f \to 1/3$, the statistical bound degrades rapidly—the committee sampling alone provides insufficient security. At this point the system relies primarily on the deterministic safety proof (Theorem 5.4.2) and the SafeNode predicate rather than committee sampling. This two-layer defense (statistical sampling + deterministic BFT safety) is intentional: the Chernoff bound provides an extra safety margin during normal operation, while the core BFT proof carries the load near the threshold.

The protocol selects $v = 256$ as the optimal tradeoff:

- Security: 2^{-18} failure probability ($\approx$ once per 30 years at 1h epoch)
- Efficiency: 256 validators per round enables <1s quorum formation
- Decentralization: Small enough that mid-range server hardware can participate

Larger committees ($v = 512$) provide marginal security gains but increase communication overhead quadratically.

Under this model, safety violation implies $> 1/3$ equivocation.

Lemma I.2 (Quorum Intersection)**.** *Any two validator sets each containing $> 2/3$ of total weighted stake must intersect in $> 1/3$ of the total stake. (The proof is a standard inclusion-exclusion argument: two sets exceeding $2S/3$ cannot avoid overlapping by more than $S/3$, since their union would otherwise exceed S.) Under $< 1/3$ Byzantine stake, this guarantees at least one honest validator in every quorum intersection.*

J Rationale for Parameter Selection

J.1 DAS Query Count ($k = 30$)

The query count balances detection probability against per-block overhead. For an adversary withholding fraction f of data, the miss probability is $(1 - f)^k$. At $k = 30$ and $f = 0.5$ (half the data withheld), this is $\approx 10^{-9}$—roughly one undetected withholding event per 30 years at one block per second. Going higher ($k = 100$) would push detection to 10^{-30} but triples the bandwidth cost per block for negligible practical gain. Going lower ($k = 10$) leaves 10^{-3}, which is uncomfortably close to once-per-hour at high block rates. We chose 30 as the point where the curve flattens.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Yellow Paper, 2014.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM Review, vol. 41, no. 2, 1999.

[4] NIST, "FIPS 204: Module-Lattice-Based Digital Signature Standard," 2024.

[5] NIST. "Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process," 2016.

[6] Ben-Sasson, E., et al. "Scalable, transparent, and post-quantum secure computational integrity," *IACR Cryptology ePrint Archive*, 2018.

[7] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," 2018.

[8] L. Baird, "The Swirlds hashgraph consensus algorithm," Swirlds Tech Report, 2016.

[9] Sompolinsky, Y., Sutton, M. "The DAG KNIGHT Protocol: A Parameterless Generalization of Nakamoto Consensus," *IACR Cryptology ePrint Archive*, 2022/1494, 2022.

[10] Goldwasser, S., Micali, S., Rackoff, C. "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, 1989.

[11] C. Dwork, N. Lynch, L. Stockmeyer, "Consensus in the presence of partial synchrony," JACM, 1988.

[12] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems," *USENIX Security*, 2021.

[13] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity," *ICALP*, 2018.

[14] Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme," *TCHES*, 2018.

[15] Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A. "Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus," *EuroSys*, 2022.

[16] Bonneau, J., Meckler, I., Rao, V., Shapiro, E. "Mina: Decentralized Cryptocurrency at Scale," 2020.

[17] Karabulut, E., Aysu, A. "FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks," *58th ACM/IEEE Design Automation Conference (DAC)*, pp. 691–696, 2021.

[18] Zhang, S., Lin, X., Yu, Y., Wang, W. "Improved Power Analysis Attacks on Falcon," *Advances in Cryptology — EUROCRYPT 2023*, LNCS vol. 14007, pp. 565–595, 2023.

[19] Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M. "The Hidden Parallelepiped Is Back Again: Power Analysis Attacks on Falcon," *IACR TCHES*, 2022(3), pp. 141–164, 2022.

[20] Bariant, A., Bouvier, C., Leurent, G., Pernot, C. "Algebraic Cryptanalysis of the HADES Design Strategy," *ACISP*, 2024; *IACR ePrint* 2023/537.

[21] Zhao, Z., Sanso, A., Vitto, G., Ding, J. "Graeffe-Based Attacks on Poseidon and NTT Lower Bounds," *IACR Cryptology ePrint Archive*, 2025/1916, 2025.

[22] Grassi, L., Khovratovich, D., Schofnegger, M. "Poseidon2: A Faster Version of the Poseidon Hash Function," *AFRICACRYPT 2023*, LNCS vol. 14064, pp. 177–203, 2023.

[23] Ethereum Foundation Poseidon Group. "Poseidon Cryptanalysis 2024–2026," `https://poseidon-initiative.info/`

[24] Bak, A., Bariant, A., Boeuf, A., Briaud, P., Øygarden, M., Phanse, A. "The Algebraic CheapLunch: Extending FreeLunch Attacks on Arithmetization-Oriented Primitives Beyond CICO-1," *IACR Cryptology ePrint Archive*, 2025/2040, 2025.

[25] Grassi, L., Koschatko, K., Rechberger, C. "Poseidon and Neptune: Gröbner Basis Cryptanalysis Exploiting Subspace Trails," *IACR Trans. Symmetric Cryptology*, 2025(2), pp. 34–86, 2025.

[26] Mukherjee, S., Rechberger, C., Schofnegger, M. "Cache Timing Side Channels on Poseidon-GL," *IACR TCHES*, 2025.

[27] Khovratovich, D., Gabizon, A., Gurkan, K., Maller, M., Tyagi, N. "MinRoot: Candidate Sequential Function for Ethereum's VDF," *IACR Cryptology ePrint Archive*, 2022/1626, 2022.

[28] Biryukov, A., Fisch, B., Herold, G., Khovratovich, D., Leurent, G., Naya-Plasencia, M., Wesolowski, B. "Cryptanalysis of Algebraic Verifiable Delay Functions," *CRYPTO*, 2024.

[29] NIST, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard," 2024.

[30] Shamir, A. "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, 1979.