Eliminating Weak Subjectivity via Post-Quantum Recursive SNARKs

Technical Review: Q-NarwhalKnight Cryptographic Light Client Protocol

Version 1.0.0

Q-NarwhalKnight Protocol Team
`protocol@quillon.xyz`

December 2024

Abstract

This document presents a novel approach to eliminating weak subjectivity in BFT consensus systems using **post-quantum recursive SNARKs**. We leverage Q-NarwhalKnight's existing LatticeGuard (RLWE-based zk-SNARK) and ZK-STARK infrastructure to create an **Incrementally Verifiable Computation (IVC)** chain that allows new nodes to cryptographically verify the entire blockchain history in constant time (∼10ms) without trusting any checkpoint provider.

This is the first design for **post-quantum recursive proofs for hybrid DAG-BFT consensus**.

Contents

1 Problem Statement

1.1 What is Weak Subjectivity?

In BFT/PoS consensus systems, new nodes joining the network face a fundamental problem. An attacker can create a fake chain with the same genesis but different history, and a new node cannot distinguish the real chain from the fake one without external trust.

The Weak Subjectivity Problem

Real Chain: $[G] \to [1] \to [2] \to \ldots \to [1000000]$
Fake Chain: $[G] \to [1'] \to [2'] \to \ldots \to [1000000']$

Both chains start from the same genesis G, but contain different history. A new node cannot distinguish them without external trust!

Why BFT systems have this problem:

- Validator sets change over time
- Old validators may have unbonded and sold keys
- Attacker can buy old keys and sign alternate history
- No "proof of work" anchoring history to physics

1.2 Why This Matters

System	Bootstrap Trust	Verification Time	Post-Quantum
Bitcoin	None (verify from genesis)	Hours-Days	No
Ethereum 2.0	Checkpoint trust	Minutes	No
Current Q-NarwhalKnight	Checkpoint trust	Minutes	Yes
Proposed Q-NarwhalKnight	**None (cryptographic)**	**∼10ms**	**Yes**

Table 1: Comparison of blockchain bootstrap mechanisms

1.3 Goal

Create a system where:

1. New nodes verify entire chain history in **constant time** (∼10ms)
2. **No trusted checkpoints** – purely cryptographic verification
3. **Post-quantum secure** – resistant to quantum attacks
4. **Decentralized proof generation** – no single prover

2 Background: Current Q-NarwhalKnight Architecture

2.1 Consensus Layers

Q-NarwhalKnight implements a hybrid consensus with four layers:

1. **Layer 1: Lightweight Mining** – CPU-friendly proof-of-computation with memory-hard operations for Sybil resistance
2. **Layer 2: VDF Leader Election** – Genus-2 Jacobian VDF with sequential computation (no parallel speedup)
3. **Layer 3: DAG Structure** – Parallel block production with multiple parents per block
4. **Layer 4: BFT Finality** – 2f+1 validator signatures using SQIsign/Dilithium5 post-quantum signatures

2.2 Existing ZK Infrastructure

Q-NarwhalKnight has three ZK systems:

2.2.1 LatticeGuard (Post-Quantum SNARK)

Based on Ring-LWE assumption with security levels PQ128, PQ192, PQ256:

Listing 1: LatticeGuard proof structure

```
pub struct LatticeGuardProof {
    commitments: Vec<LatticeCommitment>,
    evaluations: (Scalar, Scalar, Scalar),
    product_proofs: Vec<ApproximateProductProof>,
    transcript_state: [u8; 32],
}
```

Key parameters: Dimension 1024-4096, Modulus 32-64 bits, Proof size 10-50 KB.

2.2.2 ZK-STARK (Hash-Based, Inherently PQ)

Transparent setup with GPU acceleration, targeting 50K+ TPS.

2.2.3 Traditional SNARKs

Groth16, PLONK, Marlin, Sonic for non-PQ applications where performance is critical.

2.3 Network Layer (libp2p)

Gossipsub topics for P2P communication:

- `/qnk/testnet/blocks` – Block propagation
- `/qnk/testnet/peer-heights` – Height announcements
- `/qnk/testnet/bft-votes` – BFT signature collection

3 Solution Overview: Recursive Proof Chain

3.1 Core Idea: Incrementally Verifiable Computation (IVC)

Instead of verifying each block individually, we create a **single proof** that attests to the validity of all blocks from genesis to current height.

Definition 3.1 (Recursive Epoch Proof)**.** *Let π_n be the proof for epoch n. Then:*

$$\pi_n = \mathit{Prove}\left(\mathit{Verify}(\pi_{n-1}) = 1, \mathit{blocks}_n, \mathit{sigs}_n, \mathit{state}_n\right)$$

The key property is that verification time is $O(1)$ regardless of chain length.

3.2 What Each Epoch Proof Contains

Listing 2: Epoch proof public inputs

```
pub struct EpochPublicInputs {
    pub previous_state_root: [u8; 32],
    pub current_state_root: [u8; 32],
    pub epoch: u64,
    pub height_range: (u64, u64),
    pub validator_set_hash: [u8; 32],
    pub signature_count: u32,
}
```

3.3 The Recursive Circuit

The epoch transition circuit verifies:

1. C_1: Previous proof verification $\text{Verify}(\pi_{n-1}, \text{prev\_root}) = 1$
2. C_2: Validator set hash correctness
3. C_3: BFT threshold $\geq 2f + 1$ valid signatures
4. C_4: All epoch blocks are valid
5. C_5: State transition correctness
6. C_6: Merkle root computation

4 Circuit Designs

4.1 LatticeGuard Verifier Circuit (Recursive Component)

The most critical component: a circuit that verifies a LatticeGuard proof inside itself.

Theorem 4.1 (Recursive Verification Complexity)**.** *The LatticeGuardVerifierCircuit requires approximately 100,000 R1CS constraints for verification of a proof with dimension 1024.*

The circuit performs:

1. **Commitment verification** – Verify RLWE ciphertexts are well-formed
2. **Fiat-Shamir transcript reconstruction** – Recompute challenges using Poseidon hash
3. **Polynomial evaluation verification** – Check evaluations at challenge point
4. **Approximate product verification** – Verify R1CS satisfaction with bounded error

4.2 BFT Signature Verification Circuit

Proves that $\geq 2f + 1$ validators signed the epoch blocks using Dilithium5 signatures.

Listing 3: BFT signature circuit

```
pub struct BFTSignatureCircuit {
    n_validators: usize,
    f: usize,  // Byzantine threshold
    validator_keys: Vec<DilithiumPublicKey>,
    signatures: Vec<Option<DilithiumSignature>>,
    message: [u8; 32],
}
```

Dilithium verification requires approximately 100,000 constraints per signature.

4.3 State Transition Circuit

Verifies that epoch state transitions are valid:

- Block hash computation correctness
- DAG parent existence verification
- VDF output correctness (lightweight check)
- Transaction validity
- State update computation

4.4 Complete Epoch Circuit

Component	**Constraints**	**Notes**
LatticeGuard Verifier	∼100,000	Recursive verification
BFT Signatures (5 sigs)	∼500,000	Minimum viable
State Transition	∼200,000	Depends on epoch size
Overhead	∼50,000	Glue logic
Total per Epoch	∼850,000	Conservative estimate

Table 2: Epoch transition circuit constraint breakdown

5 Decentralized Proof Generation via libp2p

5.1 The Decentralization Challenge

Generating recursive proofs is computationally expensive:

- ∼850K constraints per epoch
- ∼30-60 seconds proving time (CPU)
- ∼5-10 seconds with GPU acceleration

We cannot rely on a single prover – this would centralize trust.

5.2 Decentralized Architecture

Multiple prover nodes compete to generate epoch proofs:

1. **Epoch Finalized** – BFT consensus completes
2. **Task Broadcast** – Gossipsub: `/qnk/epoch-proof-task`
3. **Parallel Proving** – Multiple provers race
4. **First Valid Wins** – Gossipsub: `/qnk/epoch-proofs`
5. **All Verify** – 10ms verification by all nodes
6. **DHT Storage** – Key: `/qnk/proofs/epoch/{N}`

5.3 libp2p Protocol Specification

New gossipsub topics:

```
pub const TOPIC_EPOCH_PROOF_TASK: &str = "/qnk/epoch-proof-task";
pub const TOPIC_EPOCH_PROOFS: &str = "/qnk/epoch-proofs";
pub const TOPIC_PROOF_VERIFICATION: &str = "/qnk/proof-verification";
```

5.4 Incentive Mechanism

Listing 4: Proof reward calculation

```
pub fn calculate_reward(submission: &EpochProofSubmission, task: &
    EpochProofTask) -> u64 {
    let mut reward = BASE_REWARD;

    // Speed bonus for fast proofs
    if submission.proving_time_ms < TARGET_PROVING_TIME_MS {
        let speedup = TARGET_PROVING_TIME_MS - submission.
            proving_time_ms;
        reward += SPEED_BONUS * speedup / TARGET_PROVING_TIME_MS;
    }

    // Late penalty (no negative rewards)
    if now > task.deadline {
        let penalty = LATE_PENALTY_PER_SECOND * (now - task.deadline);
        reward = reward.saturating_sub(penalty);
    }

    reward
}
```

5.5 Light Client Sync Protocol

The key function – trustless bootstrap in ∼10ms:

Listing 5: Light client bootstrap

```
pub async fn bootstrap(&mut self) -> Result<()> {
    // Request proof from multiple peers
    let responses = self.request_from_multiple_peers(request).await?;
    let best_response = self.find_consensus_response(&responses)?;

    // CRITICAL: Verify the proof - trustless!
    let is_valid = self.verifier.verify(
        &best_response.proof,
        &public_inputs,
    )?;

    // Verification time: ~10ms
    // Trust required: NONE
}
```

6 Implementation Roadmap

1. **Phase 1: Circuit Foundations (4-6 weeks)**

- Poseidon hash gadget for LatticeGuard
- Dilithium signature verification circuit
- Merkle tree verification circuit

2. **Phase 2: Recursive Prover (6-8 weeks)**
   - LatticeGuardVerifierCircuit
   - BFTSignatureCircuit
   - StateTransitionCircuit
   - EpochTransitionCircuit
3. **Phase 3: P2P Integration (4-6 weeks)**
   - New gossipsub topics
   - ProverNode implementation
   - Proof verification and storage
4. **Phase 4: Light Client (3-4 weeks)**
   - LightClient bootstrap
   - Proof request/response protocol
   - Wallet integration
5. **Phase 5: Optimization (Ongoing)**
   - GPU acceleration
   - Signature aggregation
   - Proof compression

7 Security Analysis

7.1 Threat Model

Threat	**Mitigation**
Malicious prover generates invalid proof	All nodes verify proofs before accepting
Prover collusion	Multiple independent provers race; any valid proof accepted
Proof withholding	Timeout triggers re-proving by other nodes
Long-range attack (fake history)	Recursive proof verifies entire history cryptographically
Quantum attack on RLWE	Parameters chosen for 128+ bit post-quantum security

Table 3: Threat model and mitigations

7.2 Security Assumptions

1. **RLWE hardness**: Ring-LWE problem is hard for quantum computers
2. **Hash collision resistance**: BLAKE3/Poseidon are collision-resistant
3. **Honest majority for BFT**: $< 33\%$ Byzantine validators
4. **At least one honest prover**: Some prover generates valid proofs

8 Performance Projections

8.1 Proving Times

Hardware	Estimated Time	Speedup
CPU (16 cores)	45-60 seconds	1x (baseline)
GPU (RTX 4090)	8-12 seconds	5-6x
GPU Cluster	3-5 seconds	10-15x
FPGA (future)	<1 second	50x+

Table 4: Proving time estimates by hardware

8.2 Proof Sizes and Verification

- **Total Light Client Proof**: ~50 KB
- **Verification Time**: ~10-20 ms
- **Verification Complexity**: $O(1)$ regardless of chain length

9 Comparison with Existing Work

9.1 Mina Protocol

Aspect	Mina	Q-NarwhalKnight
Proof System	Pickles (Pasta curves)	LatticeGuard (RLWE)
Post-Quantum	No	Yes
Consensus	Ouroboros Samasika	DAG-BFT + VDF Mining
Proof Size	~22 KB	~50 KB
Verification	~1 second	~10 ms

Table 5: Comparison with Mina Protocol

9.2 Key Innovation

Q-NarwhalKnight is the first to combine:

1. Post-quantum recursive proofs (LatticeGuard)
2. Hybrid consensus (VDF mining + BFT)
3. Decentralized proving via libp2p
4. Elimination of weak subjectivity for BFT

10 Open Research Questions

10.1 Efficiency Improvements

- Can we aggregate Dilithium signatures to reduce BFT circuit size?
- Can we prove sub-epochs and combine (incremental proving)?
- Can recursive proof size be reduced below 50KB?

10.2 Security Questions

- What RLWE parameters balance security vs. performance?
- What if all provers go offline?
- How long should proofs be valid?

10.3 Economic Questions

- What reward structure ensures sufficient provers?
- Should there be a proof marketplace?
- Should validators be required to prove?

11 Conclusion

This design eliminates weak subjectivity from Q-NarwhalKnight's BFT consensus layer using post-quantum recursive SNARKs. The key innovations are:

1. **LatticeGuard Recursion**: First recursive proof system based on RLWE
2. **Decentralized Proving**: P2P network of competing provers via libp2p
3. **Constant-Time Verification**: New nodes verify entire history in ∼10ms
4. **Full Post-Quantum Security**: All cryptographic components are quantum-resistant

This represents a significant step forward in blockchain technology: **trustless light clients for BFT consensus** without any social-layer assumptions.

A Glossary

Term	**Definition**
IVC	Incrementally Verifiable Computation – proofs that verify other proofs
RLWE	Ring Learning With Errors – post-quantum hardness assumption
R1CS	Rank-1 Constraint System – arithmetic circuit representation
Weak Subjectivity	Need for trusted checkpoints in BFT systems
BFT	Byzantine Fault Tolerance – consensus despite malicious actors
VDF	Verifiable Delay Function – sequential computation proof