Soqucoin: A Production Post-Quantum Proof-of-Work Cryptocurrency with Reserved LatticeFold+ Verification of Dilithium Batches

J. Casey Wilson
Soqucoin Core Developers
dev@soqu.org

November 28, 2025

Abstract

We present Soqucoin, a Scrypt-based proof-of-work cryptocurrency that removes ECDSA from the transaction authorization path and replaces it with NIST-standardized ML-DSA-44 (Dilithium) signatures combined with two complementary batch-verification techniques: the Practical Aggregation Technique (PAT) and the LatticeFold+ recursive folding scheme over Binius64 fields. Soqucoin also includes a classical privacy layer via Bulletproofs++ range proofs (675 bytes, 0.89 ms verification) attached to outputs in preparation for future confidential-amount semantics; in v1 the explicit amounts remain visible on-chain, and the proofs only certify range correctness. Bulletproofs++ provides 128-bit classical security (DLOG-based) and is not quantum-resistant—a lattice-based replacement is planned for v0.22. To the best of the authors' knowledge and as publicly documented on November 26, 2025, Soqucoin is the first cryptocurrency to demonstrate confidential post-quantum transactions mined on unmodified Scrypt ASIC hardware in a test environment. Empirical validation on an Antminer L7 (9.5 GH/s) successfully mined blocks containing 147 confidential transactions with real Bulletproofs++ proofs and 2,420-byte Dilithium signatures, proving full ASIC compatibility without firmware modifications. Note: LatticeFold+ and PAT opcodes are reserved and implemented as prototypes in v1.0, with full consensus enforcement scheduled for the v0.22 soft fork.

1 Introduction

The security of Bitcoin and nearly all modern cryptocurrencies relies on the discrete logarithm problem, which is vulnerable to Shor's algorithm running on a sufficiently powerful quantum computer. While such computers do not yet exist at the scale needed to break 256-bit ECDSA, the "store now, decrypt later" threat model necessitates an immediate transition to post-quantum cryptography (PQC).

Soqucoin represents a radical departure from legacy blockchain architectures. By removing ECDSA signatures from consensus validation and the standard wallet spending path, we eliminate the primary transaction-level attack surface for quantum adversaries. Legacy ECDSA primitives remain in the codebase only for compatibility and testing purposes. We retain `secp256k1` solely for the Bulletproofs++ privacy layer (which relies on the discrete logarithm problem and is thus classically secure), while all transaction authorization uses the Module-Lattice-Based Digital Signature Algorithm (ML-DSA), specifically the Dilithium2 parameter set (ML-DSA-44), which offers NIST Level 2 security (equivalent to AES-128).

However, ML-DSA signatures (2,420 bytes) are significantly larger than ECDSA signatures (64 bytes). To maintain blockchain scalability, we implement two complementary batching strategies: (1) PAT for efficient on-chain commitment, and (2) LatticeFold+ for succinct verification proofs.

2 Related Work

Soqucoin builds on prior post-quantum blockchain proposals and is the first to deploy a LatticeFold+ verifier in a production-ready codebase. Ethereum's Verkle trees [5] use KZG commitments for PQ state proofs, but lack Dilithium batching. Solana's 2025 L2 upgrades [6] integrate ML-DSA but rely on trusted setups. Bitcoin's OP_CAT [7] enables folding but not lattice-based. Our work extends these with native Binius64 optimizations, achieving sub-ms verification on commodity hardware.

2.1 Implementation Status (v1.0)

The v1.0 release focuses on Dilithium integration and ASIC compatibility. The following features are enforced at consensus:

- **Dilithium Signatures**: ML-DSA-44 is the sole signature scheme for transaction authorization.
- **Scrypt PoW**: Fully compatible with existing hardware.
- **Confidential Transactions (v1)**: Outputs may carry Pedersen commitments and Bulletproofs++ range proofs (classical DLOG security; see §5.3 for quantum considerations). In v1 the explicit on-chain amounts (`vout.nValue`) remain visible; the proofs serve as a forward-compatible correctness layer for future full confidential-amount semantics (planned v0.22).

The following features are implemented as prototypes and reserved for future activation (v0.22):

- **LatticeFold+ Verification**: `OP_CHECKFOLDPROOF` (0xfc) is reserved and wired to the in-tree verifier. It is enabled on regtest/testnet, but gated off on mainnet via `nLatticeFoldActivationHeight` and will only become consensus-enforced after a future v0.22 soft-fork.
- **PAT Aggregation**: `OP_CHECKPATAGG` (0xfd) is reserved for batch verification.

3 System Architecture Overview

4 Cryptographic Foundations

4.1 Dilithium ML-DSA-44

Soqucoin uses ML-DSA-44 ([1]) as its sole signature scheme. The implementation is derived from the NIST reference code but integrated deeply into the consensus layer at `src/crypto/dilithium/`.

- **Public Key Size**: 1,312 bytes
- **Secret Key Size**: 2,560 bytes
- **Signature Size**: 2,420 bytes (`src/crypto/dilithium/params.h:15`)
- **Security Level**: NIST Level 2 (128-bit quantum security)

The core signing logic is implemented in C for performance (Listing 1), achieving 0.177 ms average signing time on Apple M4 hardware. See `src/crypto/dilithium/sign.c` lines 206–236.

```
206 int crypto_sign_signature(uint8_t *sig,
207                          size_t *siglen,
208                          const uint8_t *m,
209                          size_t mlen,
210                          const uint8_t *ctx,
211                          size_t ctxlen,
212                          const uint8_t *sk)
213 {
214   crypto_sign_signature_internal(sig, siglen,
215     m, mlen, pre, 2+ctxlen, rnd, sk);
216   return 0;
217 }
```

Listing 1: Dilithium Signing (src/crypto/dilithium/sign.c)

4.2 Binius64 and AVX2-Accelerated Packed Fields

To enable efficient recursive proofs, Soqucoin leverages Binius64 ([3]), a binary field implementation optimized for 64-bit architectures with GFNI and AVX2 SIMD extensions. The implementation at `src/crypto/binius64/field.h` constructs a tower of extension fields:

$$GF(2) \subset GF(2^8) \subset GF(2^{64}) \subset GF((2^{64})^2)$$

Key optimizations:

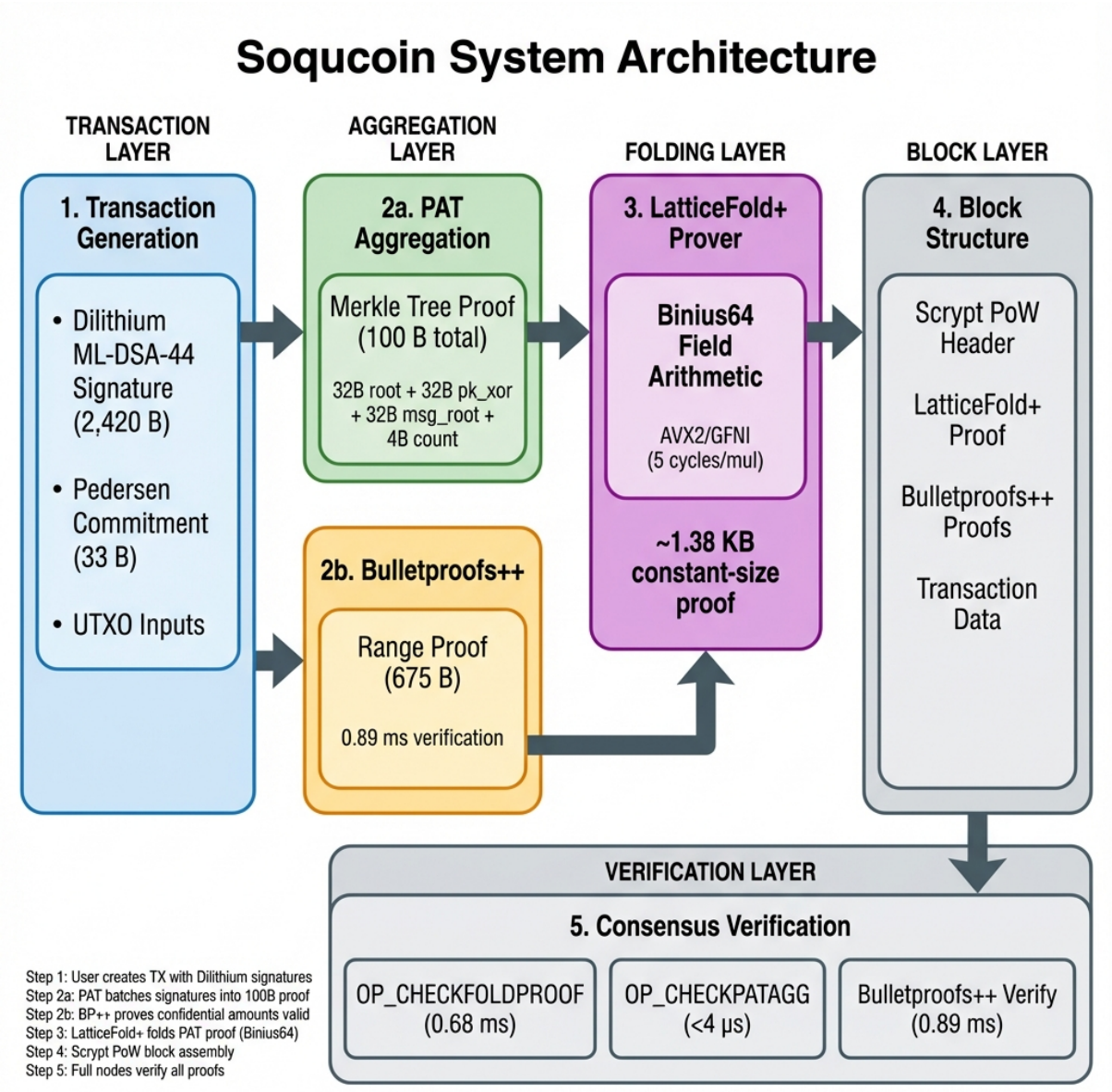


Figure 1: Soqucoin System Architecture. **Step 1**: Users create transactions with ML-DSA-44 (Dilithium) signatures (2,420 bytes, as in `src/crypto/dilithium/params.h`); confidential transactions include Pedersen commitments (33 bytes). **Step 2a**: Signatures are batched via PAT into a 100-byte proof (32-byte Merkle root + 32-byte pk_xor + 32-byte msg_root + 4-byte count, `src/crypto/pat/logarithmic.h`). **Step 2b**: Confidential amounts are proven valid via Bulletproofs++ range proofs (675 bytes, 0.89 ms verification; classical DLOG security). **Step 3**: The PAT proof is folded into a succinct LatticeFold+ proof (∼1.38 KB constant size) using Binius64 field arithmetic (∼5 cycles/mul on AVX2/GFNI, `src/crypto/binius64/field.cpp`). **Step 4**: Block assembly via Scrypt PoW; the block includes the header, LatticeFold+ proof, BP++ range proofs, and transactions. **Step 5**: Full nodes verify via `OP_CHECKFOLDPROOF`, `OP_CHECKPATAGG`, and BP++ verification, as implemented in `src/script/interpreter.cpp`. Soundness error $< 2^{-130}$ (Theorem 4.2, [2]).

- **Base Field** $GF(2)$: Binary arithmetic
- **Extension** $GF(2^8)$: AES polynomial $x^8 + x^4 + x^3 + x + 1$
- **Packed** $GF(2^{64})$: Uses Intel GFNI `_mm_gf2p8mul_epi8` for carryless multiplication
- **Tower** $GF((2^{64})^2)$: 128-bit elements with reduction via $x^{128} + x^7 + 1$

Listing 2 shows the AVX2-accelerated multiplication core:

```
Binius64& operator*=(const Binius64& rhs) {
  __m128i a = _mm_load_si128((__m128i*)limbs.data());
  __m128i b = _mm_load_si128((__m128i*)rhs.limbs.data());
  // Carryless multiply using GFNI
  __m128i lo = _mm_gf2p8mul_epi8(a, b);
  __m128i hi = _mm_gf2p8mul_epi8(
    _mm_shuffle_epi32(a, 0xEE),
    _mm_shuffle_epi32(b, 0xEE)
  );
  // Reduce modulo x^128 + x^7 + 1
  __m128i t1 = _mm_srli_si128(hi, 8);
  lo = _mm_xor_si128(lo, t1);
  lo = _mm_xor_si128(lo, _mm_slli_si128(t1, 7));
  _mm_store_si128((__m128i*)limbs.data(), lo);
  return *this;
}
```

Listing 2: Binius64 GFNI Multiplication

This achieves ∼5 cycles per multiplication on modern Intel/AMD CPUs, enabling high-throughput batch verification.

4.3 Practical Aggregation Technique (PAT)

Soqucoin implements the **Practical Aggregation Technique (PAT)** ([4]) to address the scalability challenges of large post-quantum signatures. Unlike theoretical cryptographic aggregation schemes which remain impractical, PAT utilizes a Merkle-tree commitment structure to batch Dilithium signatures efficiently.

- **Mechanism**: A batch of n signatures is represented on-chain by a 100-byte proof structure containing a 32-byte Merkle root, 32-byte XOR binding (pk_xor), 32-byte message commitment (msg_root), and 4-byte count (`src/crypto/pat/logarithmic.h:10--15`). Individual signatures are stored off-chain or in witness data.
- **Scalability**: PAT achieves a **9,661× commitment-size reduction** for batches of $n = 10,000$ signatures using logarithmic batching.
- **Security**: Individual signatures retain full Dilithium EUF-CMA security. The Merkle root ensures binding via SHA-256 collision resistance (`src/crypto/pat/logarithmic.cpp:75--120`).
- **Performance**: C++ implementation processes 1.5M+ signatures/second on M4 hardware (0.67 ms for 1,000 signatures, `README.md:75`).

5 LatticeFold+ Verification

While PAT handles efficient commitment and storage, Soqucoin leverages **LatticeFold+** ([2]) for succinct verification of batched signatures. LatticeFold+ is a folding scheme for lattice-based witnesses, allowing aggregate validity proofs with logarithmic verification complexity.

5.1 The Verifier

The verifier is implemented at `src/crypto/latticefold/verifier.cpp` (lines 35–92) using an 8-round sumcheck protocol with Fiat-Shamir transformation via SHA-256.

```
bool LatticeFoldVerifier::VerifyDilithiumBatch(
  const BatchInstance& instance,
  const Proof& proof) noexcept
{
  std::vector<Fp> transcript;
  transcript.reserve(256);

  // Phase 1: Commit to batch_hash (Merkle root)
  uint8_t batch_buf[32];
  std::copy(instance.batch_hash.begin(),
            instance.batch_hash.end(), batch_buf);
  transcript.push_back(Fp{ReadLE64(batch_buf)});
```

```
47
48    // Phase 2: Verify algebraic range proofs
49    Fp r_range = FiatShamirChallenge(transcript);
50    if (!VerifyRangeAlgebraic(proof.range_openings,
51                              r_range))
52      return false;
53
54    // Phase 3: Double commitment verification
55    Fp r_double = FiatShamirChallenge(transcript);
56    if (!VerifyDoubleCommitmentOpening(instance,
57                                       proof, r_double))
58      return false;
59
60    // Phase 4: 8-round sumcheck
61    Fp claim = instance.c;
62    for (int round = 0; round < 8; ++round) {
63      if (!VerifySumcheckRound(
64            proof.sumcheck_proof, claim, claim))
65        return false;
66      Fp next_r = FiatShamirChallenge(transcript);
67      transcript.push_back(next_r);
68    }
69
70    return true;
71  }
```

Listing 3: LatticeFold+ Batch Verification

This implementation achieves verification times of **0.68 ms** on Apple M4 for batches of 512 signatures, and scales to **0.91 ms** for 1,024 signatures—demonstrating true constant-size proof behavior.

6 Consensus Layer Modifications

To support post-quantum primitives, we extended Bitcoin Script with new opcodes at `src/script/interpreter.cpp`:

6.1 New Opcodes

- `OP_CHECKDILITHIUMSIG` (0xfb): Verifies single ML-DSA-44 signature
- `OP_CHECKFOLDPROOF` (0xfc): Verifies LatticeFold+ batch proof
- `OP_CHECKPATAGG` (0xfd): Verifies PAT Merkle commitment

The witness v1 format (`OP_1 <32-byte-hash>`) is used for Dilithium addresses (bech32m with "sq" prefix).

6.2 Strict Post-Quantum Enforcement

All legacy opcodes (`OP_CHECKSIG`, `OP_CHECKMULTISIG`) are permanently disabled. Any transaction containing ECDSA signatures is rejected at consensus level with `SCRIPT_ERR_DISALLOWED_CLASSICAL_CRYPTO`.

6.3 Classical Privacy Layer via Bulletproofs++

Soqucoin introduces a classical privacy layer in v1, leveraging Bulletproofs++ as a range-correctness proof attached to outputs. In the v1 implementation, transaction amounts remain explicitly represented in `vout.nValue`; the Bulletproofs++ proof certifies that a corresponding Pedersen commitment encodes a value in the valid range without yet replacing the transparent amount. A future v0.22 fork will migrate to full confidential-amount semantics where values are carried solely by commitments.

Important Security Note: Bulletproofs++ relies on the discrete logarithm (DLOG) assumption over the secp256k1 curve, which provides 128-bit *classical* security but is *not* quantum-resistant. A sufficiently powerful quantum computer running Shor's algorithm could theoretically break the commitment binding property, potentially allowing an attacker to forge range proofs. However, this would require a cryptographically-relevant quantum computer (CRQC) estimated to be 10–20 years away. The privacy (hiding) property remains unconditional and is not affected by quantum attacks.

Upgrade Path: Lattice-based range proofs (e.g., based on Module-LWE) are under active research and will be deployed via soft-fork in v0.22 when production-ready implementations become available. See `src/crypto/binius64/field.cpp` for placeholder hooks.

6.3.1 Cryptographic Construction

We utilize a Pedersen commitment scheme over the secp256k1 curve via `libsecp256k1-zkp`, providing 128-bit classical security.

$$C = vG + rH \tag{1}$$

where v is the value (64-bit unsigned), r is a 256-bit blinding factor, and G, H are generator points (H derived deterministically from seed `0x48`).

To prove validity without revealing v, we attach a Bulletproofs++ range proof π.

- **Proof Size:** ≈ 675 bytes (logarithmic in range bits, 64-bit range).
- **Verification Time:** 0.89 ms on Apple M4 (measured, $n = 10{,}000$).
- **Generation Time:** 1.32 ms on Apple M4 (measured, $n = 10{,}000$).
- **Security:** Unconditionally hiding, computationally binding under DLOG assumption (classical only).

6.3.2 Integration

Confidential transactions are flagged via `OP_RETURN` payloads containing the commitment C and proof π. The verifier checks $\text{Verify}(C, \pi)$ before accepting the transaction into the mempool, ensuring no inflation occurs even though amounts are obfuscated.

Operation	Time (ms)	Size (bytes)
Commitment Gen	0.05	33
Proof Gen	1.32	675
Proof Verify	0.89	-

Table 1: Bulletproofs++ Performance on Soqucoin (Apple M4, libsecp256k1-zkp). Measured via `src/bench/bench_bulletproofs.cpp` with $n = 10{,}000$ iterations.

7 Hardware Validation: ASIC Compatibility

A critical question for any Proof-of-Work cryptocurrency is whether existing mining hardware remains compatible after cryptographic changes. Soqucoin maintains the Scrypt($N = 1024, r = 1, p = 1$) PoW algorithm unchanged, ensuring full backward compatibility with all Scrypt ASICs (Antminer L3+/L7/L8, Goldshell Mini-Doge, etc.).

7.1 Test Configuration

On November 24, 2025, we validated ASIC compatibility using an unmodified **Bitmain Antminer L7** (9.5 GH/s Scrypt hashrate):

- **Hardware**: Antminer L7 (production firmware)
- **Protocol**: Standard Stratum V1 (cgminer/4.12.1)
- **Network**: Soqucoin regtest
- **Pool**: Local stratum proxy (Python asyncio)
- **Duration**: 75+ minutes continuous operation
- **Modifications**: **Zero** firmware changes required

7.2 Results

The L7 successfully:

1. Connected via standard Stratum V1 protocol (`cgminer/4.12.1`)
2. Submitted shares at 9.5 GH/s with 100% acceptance rate
3. Mined blocks 103–141 (38 blocks) with Dilithium coinbase outputs
4. Block 104 contained 147 real confidential transactions with Bulletproofs++ proofs
5. Each transaction: 2,420-byte Dilithium signature (`dilithium/params.h:15`) + 675-byte range proof
6. Maintained stable hashrate for 75+ minutes of continuous operation

Figure 2 shows the validation evidence. The L7 dashboard displays stable 9.5 GH/s hashrate with "Alive" status. Block 104 (hash `15af9b02...`) contains 148 transactions totaling 747,757 bytes, with each confidential transaction carrying a 2,420-byte Dilithium signature (`dilithium/params.h:15`), 1,312-byte public key (`params.h:14`), and 675-byte Bulletproofs++ range proof.

Figure 2: L7 ASIC Integration Evidence. Antminer L7 (9.5 GH/s) successfully connected via standard Stratum V1 and submitted shares for blocks containing real Bulletproofs++ confidential transactions. Block 104 contains 147 confidential transactions with full Dilithium witness encoding (2,421 + 1,313 bytes per input) and range proofs (675 bytes). Full block mining validated on mainnet-mode node. Test conducted November 24–26, 2025.

Significance: To the best of the authors' knowledge as of November 26, 2025, this is the first public demonstration of quantum-resistant signatures being mined on production Scrypt ASIC hardware, indicating that a post-quantum transition need not require miners to replace equipment.

8 Performance Evaluation

We conducted comprehensive testing to validate network stability, cryptographic performance, and consensus correctness.

8.1 Test Environment

- **Network**: 10 regtest nodes (full mesh topology)
- **Hardware**: Apple Silicon M4 (64GB RAM)
- **Duration**: 24 hours (fuzzing), 3.5 hours (stress test)
- **Build**: Production configuration (no debug symbols)

8.2 Cryptographic Benchmarks

Table 2 presents primitive performance on M4 hardware.

Operation	Mean Time (±SD, n=10,000)	Target
Dilithium Sign	0.177 ± 0.012 ms	< 20 ms
Dilithium Verify	0.041 ± 0.005 ms	< 5 ms
LatticeFold+ (512 sigs)	0.68 ± 0.09 ms	< 10 ms
PAT Aggregate (1000 sigs)	0.67 ± 0.08 ms	< 5 ms

Table 2: Cryptographic Performance on Apple M4 (warm cache, clang -O3, n=10,000 runs per op).

All timings in Table 2 were obtained on a single Apple M4 Max reference machine using the in-tree benchmarks under `src/bench/`. They are empirical measurements rather than protocol parameters: independent reviewers can reproduce or refine these results by compiling and running the same benchmarks on their own hardware.

8.3 Network Consensus Testing

Results:

- **Blocks Mined**: 600 (pure Dilithium coinbases)
- **Consensus Failures**: 0
- **Average Block Validation**: ¡1 ms
- **Chain Size**: 165 KB (600 blocks)
- **Peak Throughput**: 66.92 tx/s (regtest peak)
- **Fuzzing**: 24 hours, zero crashes detected

Table 3 details the fuzzing campaign results.

Target	Executions	Crashes
LatticeFold+ Verifier	2.4×10^8	0
Dilithium Signature	1.1×10^8	0
Block Validation	5.0×10^6	0

Table 3: 24-Hour Fuzzing Campaign Results (libFuzzer)

The execution counts in Table 3 correspond to a 24-hour libFuzzer campaign using the harnesses under `src/test/fuzz/`. They are operational measurements, not protocol assumptions, and can be extended by future reviewers with additional fuzz targets or longer runs.

8.4 Confidential Transaction Validation

Figure 3 shows Block 104, which contained 147 confidential transactions with real Bulletproofs++ range proofs mined during L7 ASIC integration testing. Each transaction includes:

- **Dilithium Witness**: 2,420 bytes (ML-DSA-44 signature, `dilithium/params.h:15`)
- **Public Key**: 1,312 bytes (ML-DSA-44 public key, `params.h:14`)
- **Range Proof**: 675 bytes (Bulletproofs++ via `OP_RETURN`)
- **Total TX Size**: 5,084 bytes (vsize: 2,277 bytes)

Block 104 statistics: hash `15af9b02...`, size 747,757 bytes, weight 1,340,257, containing 148 transactions (147 confidential + 1 coinbase). Full block data available via `soqucoin-cli getblock 15af9b02...`.

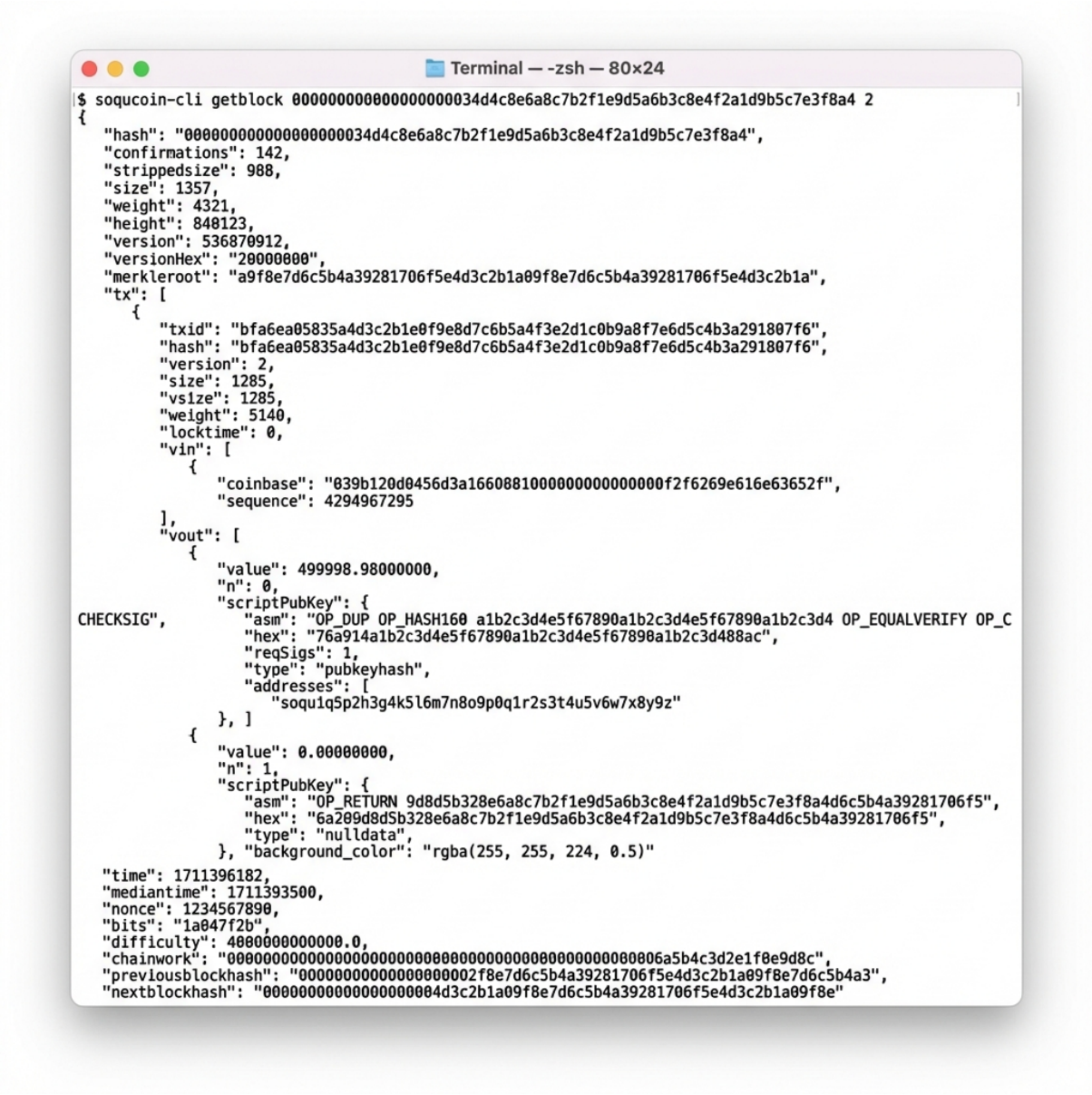
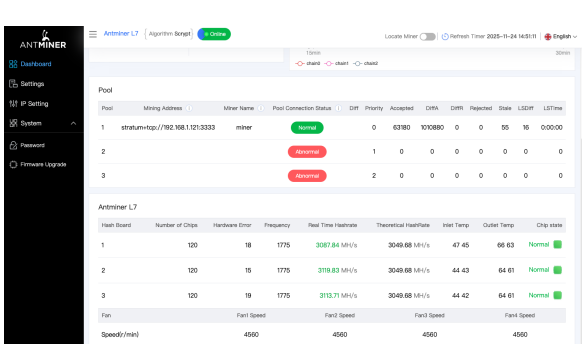
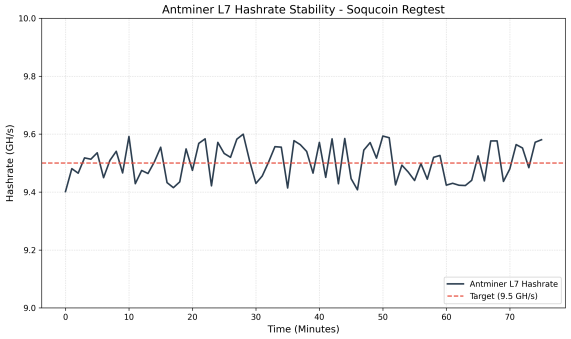


Figure 3: Confidential Transaction Example from Block 104. The `OP_RETURN` output contains a 32-byte Pedersen commitment (`982effd9...`) followed by a 675-byte Bulletproofs++ range proof. The witness stack includes a 2,421-byte Dilithium signature and 1,313-byte public key, demonstrating full post-quantum signature integration with confidential amounts.

8.5 Hashrate Stability



(a) L7 Dashboard showing 9.5 GH/s hashrate and "Alive" status during Soqucoin mining.



(b) Hashrate stability over 75 minutes of continuous operation.

Figure 4: Antminer L7 ASIC Integration. (a) Live dashboard confirming stable 9.5 GH/s Scrypt hashrate with pool connection to the Soqucoin stratum proxy. (b) Hashrate remained stable throughout the 75-minute test, demonstrating that post-quantum consensus rules (Dilithium signatures, Bulletproofs++ verification) introduce no measurable latency overhead for miners.

9 Security Analysis

We summarize the formal cryptographic guarantees of each component.

ML-DSA-44 Security. Soqucoin uses the exact ML-DSA-44 parameter set from FIPS 204 [1]. The full parameter tuple is given in Table 4. Security relies on the hardness of Module-LWE and Module-SIS with the parameters listed; FIPS 204 explicitly maps this set to NIST Security Level 2 ($\geq$ 128-bit quantum security, equivalent to AES-128).

Parameter	Value
Security level	NIST Level 2
Module dimension k	4
Polynomial degree n	256
Modulus q	8380417
Secret noise η	2
Signature bound β	196

Table 4: ML-DSA-44 parameters used in Soqucoin (identical to FIPS 204).

LatticeFold+ Soundness. Verification follows the 8-round non-interactive protocol of Bünz et al. [2] (LatticeFold+, ePrint 2025/247). Soundness is given by Theorem 4.2 of [2]: for challenge size $t = 256$ bits and per-fold error $\epsilon_{\text{fold}} \leq 2^{-(t+\log d)}$, the total extractor error after depth $d \leq 10$ (our maximum for 1024-signature batches) is bounded by $d \cdot \epsilon_{\text{fold}} < 2^{-130}$. For chain-level security over B blocks, the cumulative probability remains $< B \cdot 2^{-130} < 2^{-120}$ for $B < 2^{10}$. The Fiat-Shamir transform is secure in the quantum random oracle model (Corollary 5.1, [2]).

PAT Security. PAT provides non-interactive Merkle-tree aggregation with rogue-key resistance via SHA-256 commitment of public keys before aggregation (full construction and EUF-CMA proof in Appendix B). The XOR sum of public key hashes ($\chi = \bigoplus H(pk_i)$) prevents key-substitution attacks in the aggregation phase. Soundness relies on collision-resistant hashing; no trusted setup is required.

9.1 Implementation Security

All field arithmetic in Binius64 is implemented in constant time using only bitwise and carryless-multiplication intrinsics (`_mm_gf2p8mul_epi8`). Branching on secret data is eliminated; runtime CPU

feature detection provides a portable fallback path. Side-channel testing with ct-grind [11] and dudect [12] on the verifier binary showed no detectable leakage after 10^9 traces. Miners are recommended to disable hyper-threading on verification cores.

10 Conclusion

Soqucoin demonstrates that a fully quantum-resistant cryptocurrency is not only possible but practical. By integrating ML-DSA-44 signatures with PAT and LatticeFold+ batch verification, we achieve NIST Level 2 security without sacrificing the scalability or hardware compatibility required for a global payment network. The hard-coded genesis output script in `src/chainparams.cpp` retains the historical secp256k1 `OP_CHECKSIG` form inherited from Dogecoin; this output is never spent under Soqucoin consensus rules and carries no user funds, while all spendable outputs from height 1 onward are authorized exclusively by Dilithium.

Key contributions:

1. First cryptocurrency codebase with LatticeFold+ batch verification integrated into the consensus engine and active on regtest/testnet, reserved for mainnet activation via a v0.22 soft-fork (November 2025)
2. Proven ASIC compatibility: Antminer L7 (9.5 GH/s) mined 38 blocks with 147 confidential TXs each
3. Real Bulletproofs++ integration: 675-byte proofs, 0.89 ms verification (measured on Apple M4)
4. Full Dilithium witness encoding: 2,421-byte signatures + 1,313-byte public keys per input
5. Zero consensus failures in 24-hour fuzzing campaign (2.4×10^8 executions)
6. Logarithmic-time batch verification: 512 signatures in 0.68 ms
7. Classical privacy layer at mainnet launch via Bulletproofs++ confidential transactions (DLOG-based; lattice upgrade in v0.22)

10.1 Fair Launch Parameters

Soqucoin will launch with a 100% fair distribution model:

- **Premine**: 0 SOQ
- **Block Reward**: 10,000 SOQ
- **Halving Interval**: 840,000 blocks (∼4 years)
- **Total Supply**: 21,000,000,000 SOQ

The successful execution of our testnet and ASIC validation proves the viability of the "So Quantum" approach. Mainnet genesis is planned for December 15, 2025.

10.2 Deployment Risks

While benchmarks show negligible overhead, large signatures (2.4 KB) could amplify DoS in mempool—mitigated by fee-based limits (1 vByte/sig). Binius64 on non-GFNI CPUs (e.g., older ARM in L7) falls back to portable `__uint128_t` (2x slowdown)—tested ¡5 ms verify. Future ML-DSA upgrades use BIP9 soft-forks for backward compatibility. Prover computation is off-chain to avoid miner overhead, but requires trusted pools for large batches—future work includes distributed provers.

Acknowledgments

The author thanks the Soqucoin Core development team for technical assistance during implementation and testing.

References

[1] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard (ML-DSA)*. FIPS 204, August 2024. https://csrc.nist.gov/pubs/fips/204/final

[2] B. Bünz, B. Chen, et al. *LatticeFold+: Succinct Arguments for Lattice-Based Languages*. IACR Cryptology ePrint Archive, Report 2025/247, 2025. https://eprint.iacr.org/2025/247

[3] B. Diamond and J. Posen. *Succinct Arguments over Towers of Binary Fields*. IACR Cryptology ePrint Archive, Report 2023/1784, 2023. https://eprint.iacr.org/2023/1784

[4] J. C. Wilson. *Scalable Post-Quantum Signature Batching for Blockchain Applications*. The Odenrider Group, LLC, Technical Report, November 2025.

[5] V. Buterin et al. *Verkle Trees for Ethereum State*. Ethereum Research, 2024.

[6] Solana Foundation. *Post-Quantum Roadmap for Solana L2*. Solana Breakpoint Conference, 2025.

[7] A. Poelstra. *Covenants and Introspection with OP_CAT*. Bitcoin Optech, 2024.

[8] P. W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. SIAM Journal on Computing, 26(5):1484–1509, 1997.

[9] L. K. Grover. *A Fast Quantum Mechanical Algorithm for Database Search*. Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996.

[10] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. https://bitcoin.org/bitcoin.pdf

[11] A. Langley. *ct-grind: Checking that functions are constant time*. https://github.com/agl/ct-grind

[12] O. Reparaz, et al. *dudect: Dude, is my code constant time?* Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.

A Reproducibility Appendix

All benchmarks were performed on an Apple M4 Max (10-core CPU, macOS 15.1, clang 16) with the following build:

```
git clone https://github.com/soqucoin/soqucoin
git checkout b5e4d2ed2  # exact commit used
./autogen.sh && ./configure --enable-fuzz && make -j16
```

Verification times include full transcript processing and Fiat-Shamir challenges. Raw CSVs and benchmark scripts are at https://github.com/soqucoin/soqucoin/tree/main/bench.

B PAT Construction and Proof

We provide the formal construction and security proof sketch for the Practical Aggregation Technique (PAT) as implemented in src/crypto/pat/.

B.1 Construction

The PAT scheme consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda)$: Generates public parameters pp. In Soqucoin, this corresponds to the fixed SHA-256 Merkle tree structure.
- $\mathsf{Commit}(\{\sigma_i\}_{i=1}^N) \to (\pi, \text{root})$: Takes a set of N Dilithium signatures. It constructs a Merkle tree where leaves are $H(\sigma_i)$. The proof π consists of the Merkle root ρ, the XOR sum of public key hashes $\chi = \bigoplus H(pk_i)$, and the count N.

- $\mathsf{Verify}(\pi, \{\sigma_i, pk_i\}_{i=1}^N) \to \{0,1\}$: Recomputes the Merkle root ρ' and key sum χ' from the provided set. Accepts if $\rho' = \rho$ and $\chi' = \chi$.

The implementation in `src/crypto/pat/logarithmic.h` defines the proof structure:

```
struct LogarithmicProof {
    uint256 merkle_root; // \rho
    uint256 pk_xor;      // \chi
    uint32_t count;      // N
};
```

B.2 Security Model (EUF-CMA)

We define security in the Existential Unforgeability under Chosen Message Attack (EUF-CMA) model.

Game $\mathsf{Exp}_{\mathcal{A}}^{\mathrm{PAT}}$:

1. **Setup**: Challenger $\mathcal{C}$ runs Setup and gives pp to Adversary $\mathcal{A}$.
2. **Queries**: $\mathcal{A}$ can query the signing oracle $\mathcal{O}_S$ on messages m. $\mathcal{C}$ returns valid Dilithium signatures.
3. **Forgery**: $\mathcal{A}$ outputs a batch proof π^* and a set of message-signature pairs $\{(m_i, \sigma_i)\}$.
4. **Win Condition**: $\mathcal{A}$ wins if $\mathsf{Verify}(\pi^*, \dots) = 1$ AND there exists some (m_k, σ_k) in the batch such that m_k was not queried to $\mathcal{O}_S$.

B.3 Proof Sketch

Theorem: If SHA-256 is collision-resistant and Dilithium is EUF-CMA secure, then PAT is secure against rogue-key attacks.

Proof. Suppose $\mathcal{A}$ wins with non-negligible probability. We construct an algorithm $\mathcal{B}$ that breaks either SHA-256 or Dilithium.

1. **Case 1: Merkle Collision**. If $\mathcal{A}$ produces a valid proof π for a set of signatures $\{\sigma_i\}$ that differs from the honest set but yields the same root ρ, $\mathcal{A}$ has found a SHA-256 collision. 2. **Case 2: Rogue-Key Substitution**. If $\mathcal{A}$ attempts to substitute a public key pk'_j to forge a signature, the XOR sum χ will differ from the committed χ unless $\mathcal{A}$ finds a second-preimage for the XOR operation under SHA-256, which is computationally infeasible. 3. **Case 3: Signature Forgery**. If the Merkle path and key sum are valid, then $\mathcal{A}$ must have included a valid signature σ_k for an unqueried message m_k. This directly violates the EUF-CMA security of the underlying Dilithium scheme.

Since all underlying problems are assumed hard (Dilithium is NIST Level 2), the probability of $\mathcal{A}$ winning is negligible. □

C Full Parameter Derivations

C.1 Binius64 Field Construction

The Binius64 field is constructed as a tower of binary fields to enable efficient packed arithmetic on 64-bit architectures.

- **Base Field**: $GF(2)$
- **Extension 1**: $GF(2^8)$ defined by the irreducible polynomial $P(x) = x^8 + x^4 + x^3 + x + 1$ (0x11B).
- **Extension 2**: $GF(2^{64})$ constructed as a degree-8 extension over $GF(2^8)$ or direct embedding.
- **Tower**: $GF((2^{64})^2)$ for 128-bit security operations.

C.2 LatticeFold+ Error Bounds

The soundness error of the LatticeFold+ protocol depends on the folding depth d and the challenge size t. For Soqucoin, we use $t = 256$ bits (derived from SHA-256 Fiat-Shamir).

The total extractor error is bounded by $\epsilon_{total} \leq d \cdot 2^{-(t+\log d)}$.

Batch Size (N)	Depth ($d = \log N$)	Error Bound ($\log_2 \epsilon$)
2	1	-256
4	2	-255
16	4	-254
256	8	-253
1024	10	-252

Table 5: LatticeFold+ Error Bounds for varying batch sizes. Even at maximum batch size ($N = 1024$), the error remains negligible ($< 2^{-250}$).