Atho: A Post-Quantum Proof-of-Work Payment Network for the Quantum Age

Digital Platinum for Quantum-Secure Public Settlement

The Platinum Standard of the Quantum Age

Author: Ghost Genull

Contact: labs@atho.io

Abstract

Atho is a post-quantum-aware, proof-of-work, public UTXO payment blockchain designed for durable settlement infrastructure. The project addresses a practical systems problem: a public value-transfer network must remain independently verifiable, operationally simple, and cryptographically conservative while also planning for long-horizon changes in signature-security assumptions. Atho uses a public UTXO accounting model, local full-node validation, SHA3-384 proof-of-work hashing, Falcon-512 transaction signatures, Rust-first systems implementation, and network-specific consensus isolation.

The present codebase is organized around protocol types, consensus logic, block and UTXO persistence, Falcon cryptography, wallet state, peer networking, API surfaces, mining logic, and desktop-client interaction. The current implementation uses 100-second blocks, 8 decimal places, 1-confirmation normal transaction consensus spendability, wallet-selected confirmation policy, 100-block coinbase maturity, a 50 ATHO starting subsidy, 1,260,000-block halvings, and a 0.390625 ATHO perpetual tail reward. Recent hardening added stricter transaction decoding, stronger witness commitment behavior, explicit mainnet/testnet mining reward address requirements, bounded mempool configuration, broader storage failpoints, replay-oriented UTXO integrity checks, and a real property-test layer. This paper explains Atho's architecture, transaction model, ownership rules, block structure, mining flow, monetary behavior, wallet architecture, Base56 address encoding, storage layout, synchronization model, security posture, performance model, testing strategy, and upgrade philosophy. It distinguishes live implementation facts from earlier planning claims and treats the repository as the final source of truth.

As of this edition, Atho should be described as an extended-testnet system with roughly one month of stable public testnet operation behind it. Mainnet is close enough to plan against, but not close enough to call complete: the responsible target is approximately two to three months after this edition if fuzz runtime, differential, crash/reorg, release-gating, and infrastructure validation work finishes cleanly.

1.1. Modern Cryptography and Atho's Post-Quantum Design

Many established cryptocurrency systems rely on elliptic-curve signatures such as ECDSA or Schnorr variants. Those signature families remain practical and widely deployed today, but long-term cryptographic planning benefits from acknowledging that sufficiently capable quantum systems running algorithms such as Shor's algorithm would pose a structural threat to elliptic-curve signature assumptions. That is not a claim that Bitcoin or other elliptic-curve systems are presently broken in the real world. It is a statement about long-horizon protocol planning.

Atho addresses transaction authorization with Falcon-512, a lattice-based post-quantum signature scheme. The network also uses SHA3-384 for proof-of-work and hashing commitments. Together, those choices preserve a classical proof-of-work operating model while reducing dependence on elliptic-curve signatures for ownership authorization. The design goal is not to make exaggerated claims about permanent immunity. The design goal is to build a payment chain whose authorization model is more resilient to known quantum-era signature concerns than a conventional ECC-based ledger.

Keywords: Atho, Falcon-512, SHA3-384, proof-of-work, public UTXO, Rust, post-quantum cryptography, payment network, digital money

Table of Contents

List of Figures

List of Tables

Executive Summary

Atho is a proof-of-work payment network built for deterministic validation, post-quantum-aware transaction authorization, and operator-friendly reviewability. This edition preserves the fuller report structure of the earlier Atho white paper while updating the underlying facts to match the live repository rather than older planning language or marketing shorthand.

In the current implementation, full nodes enforce canonical transaction decoding, network-specific chain separation, strict Falcon-512 witness verification, 32-byte ownership-lock binding, block-subsidy correctness by height, and a 100-second target cadence with 1-confirmation normal transaction spendability, wallet-selected confirmation policy, and 100-block coinbase maturity. The same codebase also commits founder-hash metadata directly into canonical block headers and persists accepted chain truth through flat block archives plus LMDB-backed indexed state.

Edition basis: **2026-06-04**, repository head after the accounting and emission update.

Before preparing this edition, the repository constants, consensus modules, block header format, network parameters, wallet paths, storage layout, API surfaces, and mining behavior were reviewed directly. The current implementation enforces:

- a **100-second** target block time
- **1 confirmation** for normal transaction consensus spendability
- wallet-selected normal transaction confirmation policy, with **3 confirmations** as the default wallet filter
- **100-block** coinbase maturity across networks
- a **50 ATHO** initial block subsidy
- a **1,260,000-block** halving interval
- a **0.390625 ATHO** perpetual tail reward beginning at block **8,820,000**
- **8 decimal places**, with **1 ATHO = 100,000,000 atoms**
- no finite maximum ATHO supply cap in the current code
- fixed **founder-hash metadata** fields (`SHA3-384` and `SHA3-512`) in canonical block header serialization
- strict canonical ownership binding through a 32-byte lock digest and Falcon-512 witness public key verification
- network-specific separation through distinct consensus IDs, visible address prefixes, P2P magic values, and genesis blocks
- operator defaults that select **testnet**, while keeping mainnet launch paths present in code for controlled operator use
- required configured mining reward addresses on **mainnet** and **testnet**, with wrong-network reward addresses rejected before template construction
- bounded mempool defaults of **50,000 transactions** and **64 MiB** of virtual transaction data
- SegWit-style block sizing with a **3,000,000 vbyte** cap, **12,000,000 weight-unit** cap, and **12,000,000 raw-byte** serialized block cap
- loopback RPC defaults, cookie/HMAC credential support, and explicit public-RPC gating

- wallet runtime defaults that require encryption in operator/UI flows, while the low-level datafile format still represents an empty passphrase as plaintext mode
- built-in mainnet and testnet DNS seed/bootstrap entries, which still need live infrastructure validation before value-bearing launch
- a current launch posture of **extended stable testnet now** and **mainnet delay recommended until the remaining proof gates are green**

The paper below keeps the fuller structure of the earlier Atho report-style white paper, but updates stale information, removes deprecated policy assumptions, and aligns the discussion with the current repository.

The paper that follows is intentionally broader than a short technical brief. It covers protocol architecture, transaction and UTXO behavior, block and mempool validation, networking, storage, wallet behavior, API boundaries, performance priorities, testing expectations, and upgrade discipline so that miners, exchanges, wallet developers, node operators, and external reviewers can understand not only what Atho claims to be, but how the current code actually behaves.

1. Introduction

Digital money is most useful when it is understandable, independently verifiable, and difficult to corrupt. Over time, many blockchain systems expanded into large execution environments with broad virtual-machine surfaces, complex state interactions, bridge dependencies, wrapped-asset risk, and monetary behavior that is difficult for ordinary operators to audit directly. Those designs may fit some use cases, but they also enlarge the attack surface and the validation burden.

Atho takes a narrower and more conservative path. It is built as a payment-oriented Layer 1 with deterministic validation, simple UTXO ownership semantics, proof-of-work ordering, explicit network separation, and post-quantum-aware signatures. The goal is not to maximize programmability at the base layer. The goal is to provide a settlement network that users, miners, wallet developers, exchanges, and auditors can reason about without ambiguity.

That philosophy appears throughout the implementation. Ownership is bound directly to canonical lock digests. Legacy lock forms are rejected. Full nodes validate emitted value against height-based subsidy rules. Block headers commit to founder-hash metadata, network identity, previous chain state, and canonical commitment roots. Storage is split between flat block archives and LMDB-backed indexed state. Wallets construct transactions locally, but they do not decide consensus truth. Consensus truth remains the job of full nodes.

2. Problem Statement

Atho exists because public settlement systems must solve several hard problems at once:

- they must transfer value without relying on a central operator to approve each spend
- they must expose rules that any independent node can validate deterministically
- they must remain useful for small payments and fee markets over long time horizons
- they must protect ownership through strong signatures and canonical transaction encoding
- they must isolate networks so that mainnet, testnet, regnet, and other modes cannot bleed into each other accidentally
- they must maintain a storage and synchronization model that is practical for operators rather than only for highly specialized infrastructure

The current Atho repository responds to those problems with explicit constants, strongly typed Rust modules, Falcon-512 authorization, SHA3-384 proof-of-work, Base56 visible addresses, canonical lock-digest ownership, deterministic transaction and block validation, and network-specific genesis anchors. The remaining challenge is not to invent more moving parts, but to preserve these rules in a form that is auditable and production-oriented.

3. Atho Design Principles

3.1. Security First

Atho prioritizes strict validation order, explicit ownership binding, fail-closed witness verification, deterministic block acceptance, and typed error handling. Invalid data should fail loudly and early.

3.2. Post-Quantum Awareness

Atho uses Falcon-512 for transaction authorization to reduce dependence on elliptic-curve signatures in its ownership model.

3.3. Proof-of-Work Fairness

Blocks are ordered through SHA3-384 proof-of-work. Miners compete by producing verifiable computational work rather than by receiving privileged issuance rights.

3.4. Scarcity Through Explicit Monetary Rules

Even though the current code does not enforce a finite maximum supply cap, it does enforce an explicit reward curve. The issuance model is deterministic, height-based, and locally verifiable by every full node.

3.5. Payment Precision

Eight decimal places give Atho Bitcoin-style E-8 accounting: 1 ATHO equals 100,000,000 atoms, and all consensus amounts remain integer-only.

3.6. Simplicity Over Excessive Base-Layer Complexity

The base layer focuses on payments, ownership, block validation, and settlement. General-purpose virtual-machine complexity is intentionally not the center of the design.

3.7. Deterministic Operations

Nodes accept or reject the same canonical transaction and block bytes under the same consensus conditions. APIs, wallets, miners, and P2P paths are expected to feed into those same rules rather than bypass them.

4. System Overview

Atho is a proof-of-work payment network implemented in Rust and split across focused crates. `atho-core` defines canonical protocol objects, consensus constants, hashing, addresses, signing messages, and network identities. `atho-storage` validates chain objects in context and persists accepted state. `atho-crypto` owns Falcon-512 and hashing primitives. `atho-wallet` handles local wallet material and transaction construction. `atho-node` composes runtime behavior including mining, API, relay, and synchronization. `atho-qt` acts as a client interface rather than a second independent consensus engine.

The overall operating model is simple to state:

- wallets derive keys and build candidate transactions
- transactions are signed locally with Falcon-512

- peers and APIs feed raw bytes into strict decode and canonical validation
- the mempool stores only policy-valid, consensus-compatible transactions
- miners assemble candidate blocks from validated mempool entries
- full nodes independently validate blocks before committing state
- accepted blocks update the UTXO set, transaction indexes, and block metadata

4.1. Technical Overview of Current Code Parameters

Table 1 condenses the current code-grounded protocol parameters that frame the rest of the paper.

Table 1

Technical Overview of Current Code Parameters

Parameter	Current Code Value	Why It Matters
Consensus	Proof of Work	Orders blocks through locally verifiable work rather than delegated authority.
Proof-of-Work hash	SHA3-384	Provides a 384-bit mining digest with a modern sponge-based hash family.
Signature scheme	Falcon-512	Authorizes UTXO spends with a post-quantum-aware signature primitive.
Target block time	100 seconds	Sets the network's expected confirmation cadence and annual block count.
Block sizing model	SegWit-style vbytes and weight	Counts non-witness bytes at full weight while discounting witness bytes through virtual-size accounting.
Max block virtual size	3,000,000 vbytes	Primary throughput and fee-market limit used by miners and validators.
Max block weight	12,000,000 weight units	Equivalent to 3,000,000 vbytes under the 4:1 weight-to-vbyte scale.
Max serialized block size	12,000,000 raw bytes	Hard raw-byte ceiling for witness-heavy blocks and storage safety.
Normal transaction consensus spendability	1 confirmation	A normal transaction is confirmed and spendable once included in a valid block.
Wallet confirmation policy	User/app selected; default 3 confirmations	Wallets, merchants, and exchanges choose risk thresholds above the consensus floor.
Coinbase maturity	100 blocks	Prevents immediate reuse of freshly mined subsidy outputs.

Parameter	Current Code Value	Why It Matters
Decimals	8	Supports atom-level accounting with Bitcoin-style E-8 precision.
Base unit	atom	Keeps all accounting in fixed integer units.
Atoms per ATHO	100,000,000	Prevents floating-point ambiguity in wallet and node accounting.
Ownership model	Canonical 32-byte lock digest	Binds spend authorization to a strict public-key-derived ownership commitment.
Address format	Base56	Gives users a visible network-aware address form over the canonical payment digest.
Validation model	Deterministic full-node validation	Keeps consensus truth in the node, not in miners, explorers, or wallets.

4.2. Design Goal of the System Overview

The purpose of the system overview is not merely architectural neatness. It is to make it clear where responsibility boundaries live. Wallet code is not allowed to define consensus truth. Explorer output is not allowed to redefine spendability. The network layer is not allowed to bypass local validation. Mining is not allowed to create a private shortcut around the rules.

5.1. Atho System Overview

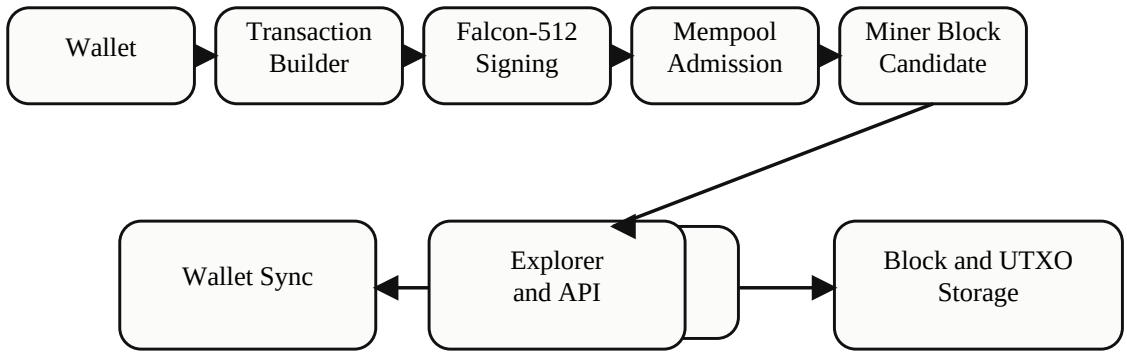

Figure 1. Atho System Overview

Atho persists accepted chain truth through a hybrid storage model. Canonical raw block bytes are archived in Bitcoin-style flat block data files, while LMDB stores block metadata, transaction archives, chainstate snapshots, UTXOs, peers, addresses, and peer health. APIs and explorer-like consumers can read from those state products, but they do not become consensus authorities. Figure 1 summarizes the movement from wallet transaction

creation through storage, UTXO update, and downstream API or wallet synchronization.

5. Protocol Architecture

Atho is a Layer 1 proof-of-work blockchain with a public UTXO ledger. Its architectural boundary is deliberately modular. `atho-core` defines protocol objects, consensus constants, network identities, addresses, and signing messages. `atho-storage` performs contextual validation and durable state application. `atho-node` composes storage, mempool, mining, RPC/API, P2P, and runtime status. `atho-wallet` owns key derivation and wallet datafiles. `atho-qt` is a client of validated node state.

This separation matters because it reduces accidental consensus drift. If user-interface code changes, consensus behavior should not change with it. If explorer presentation changes, chain acceptance should remain unaffected. If mining or API code evolves, both paths should still route through the same validated state and object model.

The repository also keeps several protocol-versioning facts explicit rather than implied. The active protocol version is `1`. The active ruleset version is `1`. The active block and transaction version are both `1`. A V2 ruleset placeholder exists in code with no activation height, which means future upgrade intent is visible without claiming that a second ruleset has already been deployed. The storage schema version is also fixed in code rather than inferred from opaque local state.

5.2. Atho Node Architecture

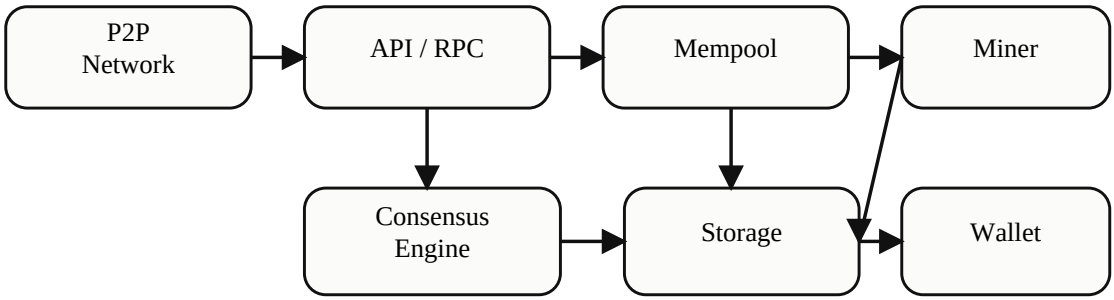

Figure 2. Atho Node Architecture

The live implementation keeps the network, API, mempool, consensus, storage, mining, and wallet interfaces separated enough that each can be inspected as its own responsibility boundary. P2P proposes data. API surfaces expose or submit data. The mempool tracks policy-valid pending transactions. Consensus validates blocks and transactions. Storage persists accepted truth. Wallet code manages keys and construction, but does not overrule the node.

6. Rust Implementation Strategy

Atho is implemented in Rust because the system benefits from explicit ownership semantics, strong typing, low-level performance control, and the ability to express consensus-critical behavior without garbage-collector pauses or unchecked memory mutation. That does not mean Rust automatically makes the protocol safe. It does mean the language gives the implementation a strong baseline for determinism, memory safety, and structured error handling when used carefully.

Consensus paths in Atho should continue to prefer domain types such as `Amount`, `TxId`, `BlockHash`, `Network`, and digest-width arrays rather than anonymous integers or unbounded strings when confusion is plausible. Recoverable failures should return typed errors. Network message handling should reject malformed input rather than panic. Serialization should stay canonical and testable. Unsafe shortcuts should remain excluded from consensus and storage paths.

6.1. Rust Safety Priorities in Practice

The current codebase already reflects several of those priorities:

- Falcon secret material uses zeroizing memory wrappers
- decoding functions reject malformed lengths and trailing bytes
- network identity is carried through typed enums and explicit visible prefixes
- consensus constants are centralized and unit-tested
- block headers and transactions use fixed byte encoders rather than ambiguous object serialization

Table 2 summarizes why Rust remains a strong fit for Atho's core infrastructure.

Table 2

Why Rust Fits Atho Core Infrastructure

Requirement	Rust Advantage	Atho Use Case
Memory safety	Ownership and borrowing reduce broad classes of aliasing and use-after-free errors.	Consensus, storage, wallet state, and P2P parsing.
Explicit errors	`Result<T, E>` models rejection and recovery without exceptions.	Decode failures, validation rejection, API error surfaces, and wallet loading.
Performance	Native compilation and direct memory layout enable predictable hot-path behavior.	Proof-of-work loops, UTXO validation, mempool admission, and storage commits.
Concurrency discipline	Rust's type system makes data-race patterns harder to express incorrectly.	Parallel Falcon verification, background sync, and runtime status collection.
Modularity	Cargo workspaces and crates encourage narrow protocol boundaries.	`atho-core`, `atho-storage`, `atho-node`, `atho-wallet`, `atho-crypto`, and `atho-qt`.

Table 3 captures the practical Rust safety rules that matter most for consensus and storage code in the current Atho repository.

Table 3

Rust Design Requirements for Atho Core Infrastructure

Requirement	Reason	Enforcement Target
Strong domain types for amounts, heights, hashes, and addresses	Prevents unit confusion and wrong-context comparisons.	`atho-core` protocol types and validation APIs.
`Result`-based rejection for recoverable failures	Invalid network input is normal control flow, not exceptional control flow.	Decoders, validators, wallet import/export, and API handlers.
No `unsafe` in consensus-critical crates	Reduces the chance of hidden memory-unsafe behavior in the core rule path.	`atho-core`, `atho-storage`, `atho-node`, and `atho-crypto` boundaries.
Canonical byte encoding only	Prevents ambiguous hashes, txids, and signature messages.	Block, transaction, witness, and address encoding.
Malformed input must fail closed	Prevents peers, APIs, or wallets from smuggling partially valid state.	P2P codec, raw transaction decode, address decode, and witness parsing.
Upgrade points must stay explicit	Makes protocol changes reviewable before activation.	Ruleset scheduling, block/transaction versions, and storage schema versioning.

7. Cryptographic Design

Atho’s cryptographic surface is intentionally narrow. SHA3-384 is used for proof-of-work hashing, block and transaction identities, signing-message derivation, and checksum-related utilities where appropriate. Falcon-512 is used for spend authorization. The design goal is not to mix multiple signature families or complex script semantics into the same ownership surface. The goal is to keep ownership proofs and hashing commitments explicit.

Cryptographic design choices in Atho should be evaluated by the following questions:

- does the node reconstruct exactly the same signing message as the wallet
- does ownership bind to a canonical digest rather than loosely interpreted script bytes
- do malformed public keys and signatures fail before acceptance
- do network-specific contexts prevent cross-network misuse
- do hash commitments include the exact fields they are supposed to include and no hidden ambiguity

7.1. SHA3-384 in Atho

SHA3-384 produces a 384-bit digest and gives Atho a larger hashing margin than shorter digest sizes while preserving deterministic, machine-verifiable output for mining and block identification. In Atho, SHA3-384 is not a branding choice. It is the actual hash family used in consensus-facing code.

7.2. Post-Quantum Security Comparison

Table 4 positions Atho relative to common classical and post-quantum-aware ledger designs.

Table 4

Post-Quantum Security Comparison

System Type	Common Examples	Signature Type	Quantum Risk Profile	Atho Difference
Traditional Bitcoin-style systems	Bitcoin-like payment chains	ECDSA or Schnorr over elliptic curves	Long-term exposure in principle to Shor-style attacks against ECC if sufficiently capable quantum systems emerge.	Atho moves spend authorization to Falcon-512 rather than ECC.
Ethereum-style systems	Account-based smart-contract platforms	ECDSA over elliptic curves	Similar long-term ECC exposure, often with a larger smart-contract surface around key usage.	Atho narrows base-layer scope and uses a UTXO payment model with Falcon-512 witnesses.
Classical payment ledgers	Conventional digital asset and payment systems	Often centralized key management or classical asymmetric schemes	Security depends on institution and implementation model rather than public-node consensus validation.	Atho combines public-node validation, SHA3-384 proof-of-work, and post-quantum-aware signatures.
Post-quantum-aware systems	Research or newer signature-transition systems	Lattice-based or hash-based signatures	Designed to reduce dependence on elliptic-curve assumptions, but implementation maturity varies.	Atho places Falcon-512 directly in the live spend-authorization path.
Atho	Atho	Falcon-512	Designed for stronger long-term cryptographic resilience in transaction authorization without claiming permanent immunity to all future attacks.	Couples Falcon-512 authorization with SHA3-384 proof-of-work and deterministic UTXO validation.

7.3. Cryptographic Scope Discipline

The project should continue to resist unnecessary cryptographic surface growth. A payment chain benefits when signature rules, hash commitments, ownership digests, address derivation, and network separation are few enough to audit well.

8. Falcon-512 Implementation in Atho

Falcon-512 is the active transaction-signature profile in Atho. Public keys, secret keys, and signatures are represented through fixed-length wrappers in `atho-crypto`. The current implementation performs strict length checks on imported key bytes, rejects malformed witness material, and keeps secret-key material in zeroizing memory structures where possible.

The role of Falcon-512 in Atho is narrow and important: it proves spend authorization for a specific transaction input under a specific signing context. It is not used for proof-of-work. It is not used as a general-purpose remote-authentication layer. It is used to authorize value transfer.

At the implementation level, the active wire and validation sizes are explicit: Falcon-512 public keys are fixed at `897` bytes, secret keys at `1,281` bytes, and signatures at `666` bytes. Those lengths are enforced at import and verification boundaries before the node attempts expensive witness verification. The node also rebuilds the canonical signing digest from transaction bytes and network context locally, rather than trusting wallet-supplied metadata about what was signed.

8.1. Signature Verification and Rejection Behavior

The consensus consequences of witness verification are more important than the abstract signature primitive. The node must reject:

- malformed public keys
- malformed signatures
- wrong-message signatures
- signatures attached to the wrong UTXO
- signatures whose public key does not map to the expected ownership digest
- missing witness components
- replay attempts across incompatible network or ownership contexts

Table 5 summarizes the expected behavior for common Falcon-512 failure cases.

Table 5

Falcon-512 Validation Failure Cases

Failure Case	Detection Method	Expected Result
Malformed public key	Length check and decode failure in the verification path	Reject as invalid witness or key material.
Malformed signature	Signature length check or verifier failure	Reject before mempool or block acceptance.
Wrong signing message	Node rebuilds the canonical signing digest and verifies against witness	Reject as invalid witness.
Changed transaction output	Base transaction bytes change the `txid` and signing context	Existing signature no longer verifies.
Wrong UTXO ownership	Lock digest derived from public key does not match stored canonical lock	Reject as ownership mismatch.
Missing signature or key	Witness payload lacks required fields	Reject as malformed witness.

9. Transaction Model

Atho transactions consume existing UTXOs and create new UTXOs. Ownership is not inferred from a broad script interpreter. Instead, spendability flows through canonical transaction serialization, explicit input references, output value accounting, and Falcon-512 witness authorization tied to a 32-byte canonical lock digest.

The transaction body is serialized deterministically. Transaction IDs derive from canonical bytes. Witnesses are stored separately from the base transaction identity so the node can preserve a stable base transaction digest while still validating the witness payload needed to authorize spends.

The current implementation also applies transaction-level resource and anti-spam policy with explicit constants. Standard transactions are bounded at `250,000` raw bytes and `250,000` vbytes. The minimum relay and construction fee floor is `vbytes * 1 atom`, with a one-atom guard for impossible zero-size cases. Dust-like outputs below `100` atoms are rejected, and standard transactions are capped at `64` outputs and `1,024` inputs. These limits are policy-facing, but they shape the operational transaction model that wallets, APIs, and miners actually use.

Normal non-coinbase transactions also carry wallet transaction proof-of-work in the current code. The wallet signs the transaction first, then derives a `SHA3-256` anti-spam preimage under the `ATHO_TX_POW_V1` domain, network consensus ID, and genesis hash, and solves for a nonce that satisfies the required difficulty bits. That per-transaction PoW is dynamic: testnet uses a reduced fixed value, while other networks scale the required bits based on transaction size, output count, and fee rate. The design goal is to add sender-side cost for fragmented or low-fee traffic without weakening block-level consensus proof-of-work.

9.1. Transaction Signing and Verification Flow

Figure 3. Atho Transaction Signing and Verification Flow

Wallet code selects spendable UTXOs, constructs a canonical transaction body, derives the signing digest, signs it with Falcon-512, and broadcasts the result. The node then rebuilds that digest locally and verifies it against the witness material. That reconstruction step is essential. Consensus cannot trust a wallet's claim about what it signed.

9.2. Transaction Lifecycle

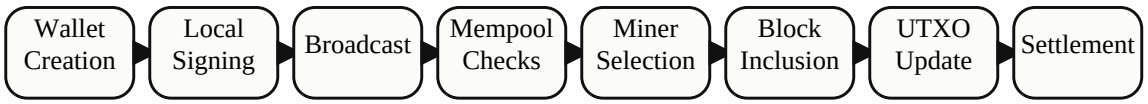

Figure 4. Transaction Lifecycle

The normal lifecycle is wallet construction, local signing, raw-byte relay, mempool validation, miner selection, block inclusion, full block validation, UTXO update, and confirmations. Each step has a different responsibility, but none is allowed to override consensus validity.

10. UTXO and Accounting Rules

The UTXO set is the live accounting surface of Atho. Every accepted spend removes previously unspent outputs and creates new ones. Correctness therefore depends on four properties:

- referenced outpoints must exist and be unspent
- the spending witness must authorize the exact UTXO being consumed
- the sum of created value must not exceed allowed input value plus valid subsidy where applicable
- duplicate spends must be rejected both within a transaction and within a block

The current repository uses canonical lock digests rather than permissive legacy script forms. Noncanonical locking data is rejected. Oversized or malformed witness data is rejected. Coinbase outputs remain locked behind a maturity threshold before they are spendable.

Table 6 summarizes the current UTXO validation rules that matter most for operators and integrators.

Table 6

UTXO Validation Rules

Rule	Current Behavior	Why It Matters
Normal transaction spendability	1 confirmation at consensus	A normal UTXO can be spent once mined into the active best chain.
Wallet/app confirmation policy	User/app selected	Wallets, merchants, and exchanges can require 0, 1, 3, 6, 20, or more confirmations based on risk.
Coinbase spendability	100 confirmations	Prevents immediate reuse of freshly mined subsidy outputs.
Lock format	Canonical 32-byte ownership digest	Prevents ambiguous or legacy anyone-can-spend behavior.
Duplicate inputs	Rejected	Stops same-transaction double spends.
Missing UTXO	Rejected	Prevents phantom value creation.
Ownership mismatch	Rejected	Ensures the witness key matches the exact lock being spent.

11. Block Structure

Each Atho block contains a canonical header plus a transaction list headed by a coinbase transaction. The header commits to the network ID, version, previous block hash, commitment roots, timestamp, difficulty bits, nonce, and founder-hash metadata fields. Those founder hashes are part of canonical header bytes in the current implementation and therefore part of block identity.

Block validation is not only about proof-of-work. The block also has to satisfy structural rules, canonical transaction decoding, coinbase placement rules, duplicate-spend rejection, UTXO validity, and monetary correctness. Valid proof-of-work alone does not rescue an invalid block.

The header structure matters because it is the smallest object whose bytes affect chain ordering. Atho's canonical header contains: block version, network ID, height, previous block hash, merkle root, witness root, founder-hash `SHA3-384`, founder-hash `SHA3-512`, timestamp, target bytes, and nonce. Nodes hash those bytes deterministically and compare the resulting digest against the target. Any disagreement about field order, digest width, or included metadata would create immediate consensus breakage.

The live implementation also makes block resource limits explicit: `3,000,000` vbytes, `12,000,000` raw serialized bytes, and `12,000,000` weight units. Blocks additionally carry fee-accounting fields for total fees, miner fees, burned fees, pooled fees, and cumulative burned amount, even though the active miner path currently routes selected fees to the miner and leaves the burn-oriented fields at zero. This keeps fee accounting visible in the canonical block object rather than hiding it in side effects.

11.1. Block Size, SegWit-Style Weight, and TPS

Atho uses SegWit-style block accounting rather than a single raw-byte block limit. The validator calculates block weight as:

```
block_weight = (base_bytes * 3) + raw_serialized_bytes
block_vbytes = ceil(block_weight / 4)
```

Because `raw_serialized_bytes = base_bytes + witness_bytes`, this is equivalent to counting base transaction data at 4 weight units per byte and witness data at 1 weight unit per byte. In practical terms, non-witness transaction data consumes the scarce vbyte budget more heavily, while Falcon witness material is still committed, serialized, bounded, and validated under the raw-byte and witness-root rules.

The active block limits are:

- maximum virtual block size: **3,000,000 vbytes**
- maximum block weight: **12,000,000 weight units**
- maximum raw serialized block size: **12,000,000 bytes**
- target block time: **100 seconds**

Estimated transaction throughput is therefore a function of average transaction vsize:

```
approx_tx_per_block = floor(3,000,000 / average_tx_vbytes)
approx_tps = approx_tx_per_block / 100
```

Example full-block throughput estimates:

- **250 vbyte** average transaction: about **12,000 tx/block**, or **120 TPS**
- **500 vbyte** average transaction: about **6,000 tx/block**, or **60 TPS**
- **590 vbyte** average transaction: about **5,084 tx/block**, or **50.8 TPS**
- **1,000 vbyte** average transaction: about **3,000 tx/block**, or **30 TPS**

These are capacity estimates from consensus sizing constants, not a promise that every deployed node will sustain those numbers under all conditions. Real throughput also depends on average input count, output count, witness size, mempool load, peer bandwidth, block propagation, disk performance, and Falcon verification throughput. The correct headline is that Atho's current code targets a **3,000,000-vbyte block budget every 100 seconds**, with raw witness-heavy blocks bounded at **12 MB**.

12. Proof-of-Work and Mining

Atho uses SHA3-384 proof-of-work. Miners search the nonce space for a header hash below the active target. The node's mining path is not supposed to create a separate private notion of validity. Candidate blocks must still pass the same validation rules that apply to blocks received from peers.

The active code uses a 100-second target block time and a 1,260,000-block halving interval. Mining therefore combines two tasks:

- ordering pending valid transactions into a candidate block
- solving for valid proof-of-work under the current target

Difficulty retargeting is also explicit rather than inferred. The current profile retargets every block using a `17`-block averaging window, an `11`-block median-time window, a damping factor of `4`, a maximum upward

adjustment of `16%`, and a maximum downward adjustment of `32%`. This makes difficulty movement responsive enough for shorter block times while still bounding abrupt oscillation.

12.1. Mining Responsibilities

Mining must:

- build a coinbase transaction that does not overpay subsidy plus fees
- include only transactions that remain valid under current UTXO state and policy
- respect canonical block serialization
- preserve founder-hash header metadata
- submit solved blocks through the same full-validation path used for externally sourced blocks

Table 7 summarizes the practical mining responsibilities exposed by the current codebase.

Table 7

Mining Components and Responsibilities

Component	Current Role	Validation Constraint
Block template builder	Selects candidate mempool transactions and constructs a coinbase.	Must use transactions that still pass node validation at template time.
Coinbase builder	Pays subsidy plus allowed fees to the configured output.	Must never overpay the height-based reward plus fees.
Merkle and witness commitment logic	Commits ordered transaction and witness data into the header.	Must match the validator’s recomputation exactly.
Proof-of-work loop	Searches nonce space for a header hash below the current target.	Valid proof-of-work cannot rescue an invalid block body.
Local block submission path	Returns solved blocks to the node.	Must re-enter the same full block validation path as peer-sourced blocks.
GPU/native acceleration	Optimizes hash-search throughput where supported.	Must not alter canonical header bytes, target rules, or submission behavior.

The latest mining path no longer uses public deterministic reward keys for value-bearing networks. Mainnet and testnet candidate construction require an explicit configured Atho reward address through `ATHO_MINING_REWARD_ADDRESS` or `atho.conf`, and the address must decode for the node’s active network. Deterministic development reward defaults are limited to regnet and prunetest.

13. Monetary Policy and Emissions

The current Atho codebase enforces a deterministic emission schedule, but it does **not** currently enforce a finite maximum ATHO supply cap. That distinction must remain explicit in every public technical document. Earlier planning language described a 78,000,000 ATHO cap, but the live consensus code does not implement it.

The active monetary behavior is:

- initial block reward: **50 ATHO**
- halving interval: **1,260,000 blocks**
- reward floor: **0.390625 ATHO**
- tail emission start height: **8,820,000**
- supply at tail start: **125,015,625 ATHO**
- block cadence assumption used by code: **100 seconds**
- blocks per year: **315,360**

Nodes reject blocks that overpay subsidy at a given height. They do not, today, reject blocks for exceeding a finite cumulative cap because the repository does not define such a cap in consensus code.

13.1. Why Current Emissions Still Matter

Even without a hard cap, the current emission path matters operationally. Wallets and exchanges need predictable accounting. Miners need a knowable reward schedule. Explorers need consistent display logic. Auditors need a height-based rule they can recompute independently.

13.2. Atho Emission Model

50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 ATHO tail

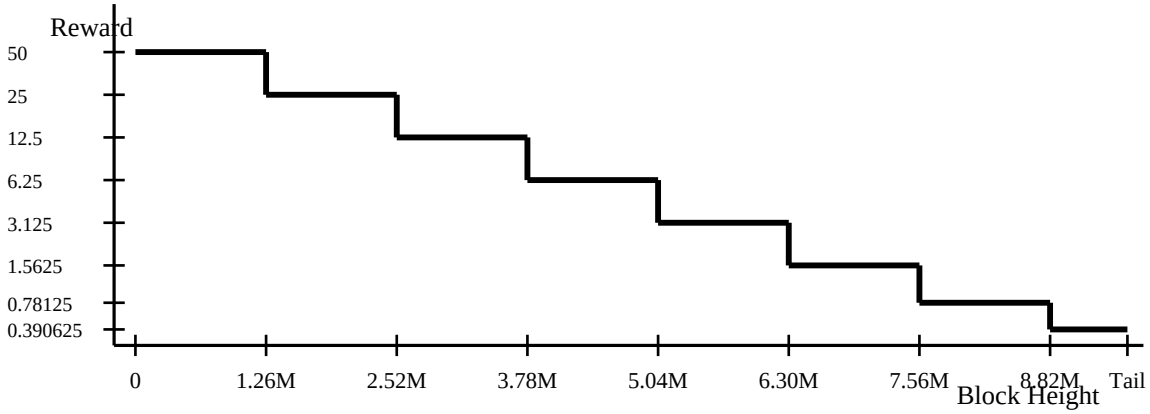

Figure 6. Atho Emission Model

The current reward path is halving-based for seven pre-tail eras and then becomes a perpetual tail reward. That means long-term issuance is eventually linear rather than capped. The active rule can be stated in plain language as **50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 ATHO**, after which the reward remains at 0.390625 ATHO indefinitely unless a future consensus change says otherwise.

Table 8 summarizes the policy constants used by the current implementation.

Table 8

Monetary Policy Constants and Current Consensus Values

Constant	Current Value	Operational Meaning
Target block time	100 seconds	36 blocks per hour and 315,360 blocks per year.
Initial block reward	50 ATHO	Opening subsidy before halvings.
Halving interval	1,260,000 blocks	Height interval between reward cuts.
Tail reward	0.390625 ATHO	Permanent floor beginning at block 8,820,000.
Tail start supply	125,015,625 ATHO	Cumulative subsidy before permanent tail emission begins.
Coinbase maturity	100 blocks	Earliest block age before coinbase outputs become spendable.
Normal transaction spendability	1 confirmation	Consensus floor for normal UTXO spendability.
Wallet confirmation policy	User/app selected; default 3 confirmations	Applications choose risk thresholds above the consensus floor.

14. Mempool Design

The mempool is the staging area for policy-valid transactions that have not yet been mined into a block. In Atho, the mempool should never become an alternate consensus system. Its job is narrower:

- reject malformed or consensus-incompatible transactions before they waste miner or relay resources
- track pending conflicts
- expose a validated pending set for block template construction
- provide useful status information to local tooling and API surfaces

The mempool should be stricter than “accept anything that might later work.” Loose pending-state logic creates relay noise, miner waste, and confusing wallet behavior.

The current node configuration gives the mempool explicit production-shaped bounds: the default cap is `50,000` transactions and `64 MiB` of transaction vbytes, with operator overrides clamped into a bounded range. Mining views revalidate entries against current UTXO state instead of blindly trusting cached admission results.

14.1. Mempool Admission Flow

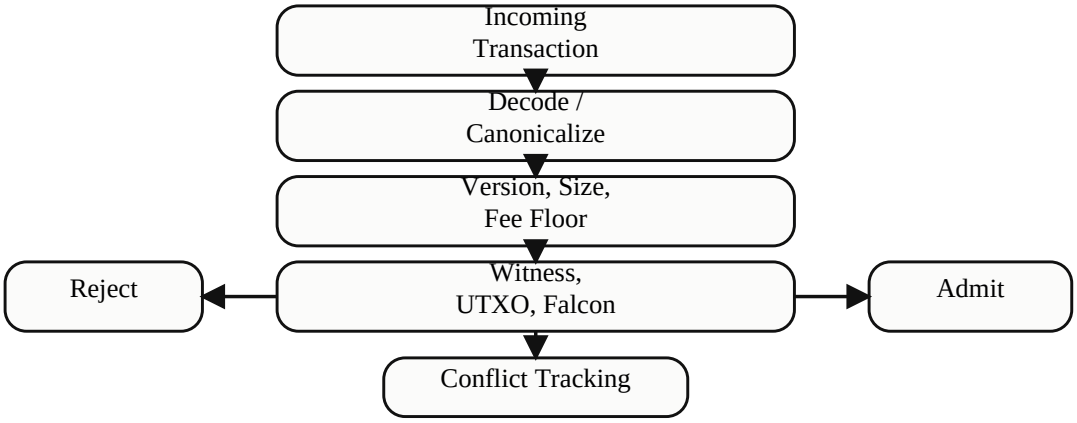

Figure 7. Mempool Admission Flow

An incoming transaction must pass decode, canonical structure, version rules, fee-floor and size policy, witness shape checks, UTXO availability checks, Falcon verification, and local conflict tracking before it is admitted.

15. Consensus Validation

Consensus validation is where Atho becomes a blockchain rather than just a messaging system. The network does not accept data because a miner produced it, a wallet broadcast it, or an API submitted it. The network accepts data because the local node can prove it is valid under canonical rules.

Transaction validation and block validation are related but distinct. Transaction validation checks canonical structure, ownership, spendability, fees, and witness correctness. Block validation adds header rules, proof-of-work, coinbase correctness, duplicate-spend detection, commitment verification, and atomic state transition requirements.

Table 9 lists representative invalid cases that must continue to fail identically across node, mempool, mining, and API entry points.

Table 9

Consensus-Critical Invalid Cases

Invalid Case	Rejection Surface	Why It Is Consensus-Critical
Wrong-network block or transaction	Header/network checks and address decode paths	Prevents cross-network acceptance and replay.
Legacy or noncanonical lock format	Output lock parsing and UTXO ownership validation	Prevents ambiguous or anyone-can-spend behavior.
Malformed witness, public key, or signature	Witness parser and Falcon verification	Prevents unauthorized spends and parser ambiguity.
Duplicate inputs or duplicate spends	Transaction and block contextual validation	Prevents explicit double spends.
Overpaid coinbase	Block monetary validation	Preserves deterministic issuance.
Wrong transaction or block version	Version checks against active ruleset	Prevents unactivated rule paths from slipping into production.
Bad merkle or witness commitment root	Block commitment verification	Prevents tampering with the transaction set or witness set.
Oversized block or transaction	Size and weight limits	Preserves resource bounds and deterministic relay/validation rules.

15.1. Block Validation Pipeline

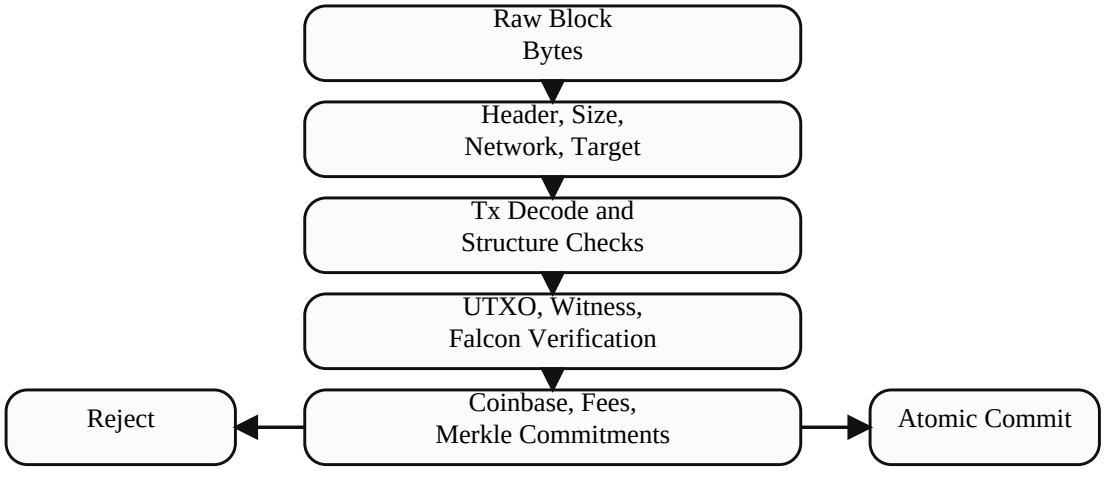

Figure 5. Block Validation Pipeline

The validation order should reject cheap failures first: malformed bytes, wrong network, bad header structure, oversized objects, coinbase violations, missing UTXOs, malformed witnesses, incorrect signatures, overpaid

subsidy, and bad commitments. Expensive work should occur only after cheap structural rejection paths are exhausted.

15.2. Validation Pipeline and Parallel Work Distribution

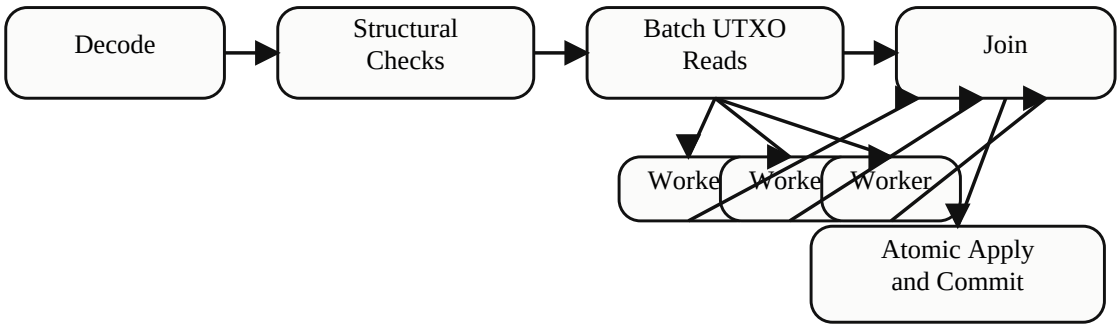

Figure 10. Validation Pipeline and Parallel Work Distribution

The current codebase already contains room for safe parallelism in independent signature verification and block-level work distribution. Parallel verification is valuable only if it remains deterministic and does not alter final state-application order. Structural checks and UTXO batch reads can prepare work in parallel, but final chainstate mutation must remain authoritative and ordered.

16. Network Layer

Atho exposes separate network identities for `mainnet`, `testnet`, `regnet`, and `prunetest`. Each has its own consensus ID, genesis anchor, visible address prefix, port set, and P2P magic values. This matters because network isolation is not a convenience feature. It prevents one network's artifacts from being accepted as another network's truth.

The P2P layer is responsible for peer connections, version exchange, inventory relay, headers-first synchronization, block retrieval, and invalid-data rejection. It is not responsible for deciding consensus truth independently of the validation engine.

16.1. Node Sync and Block Propagation Flow

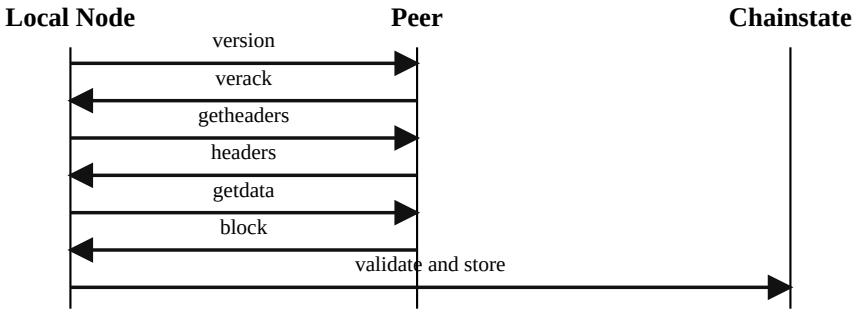

Figure 8. Node Sync and Block Propagation Flow

Headers-first synchronization allows a node to learn candidate chain shape before fetching full block payloads. After header exchange, the node requests blocks, validates them locally, persists accepted state, and only then advertises the new tip. Propagation should never outrun validation.

17. Storage Layer

The current storage model is hybrid. Canonical raw block bytes are archived in flat block data files, while LMDB stores indexed metadata, UTXO state, chain metadata, transaction archives, and related lookup structures. This mirrors a practical separation of concerns:

- flat files are efficient for canonical historical payload storage
- LMDB is efficient for current indexed state and metadata lookups

Storage remains consensus-adjacent because partial or unsafe commits would create false local chain truth. Accepted blocks must therefore update state durably and coherently.

17.1. Hybrid Storage Commit Model

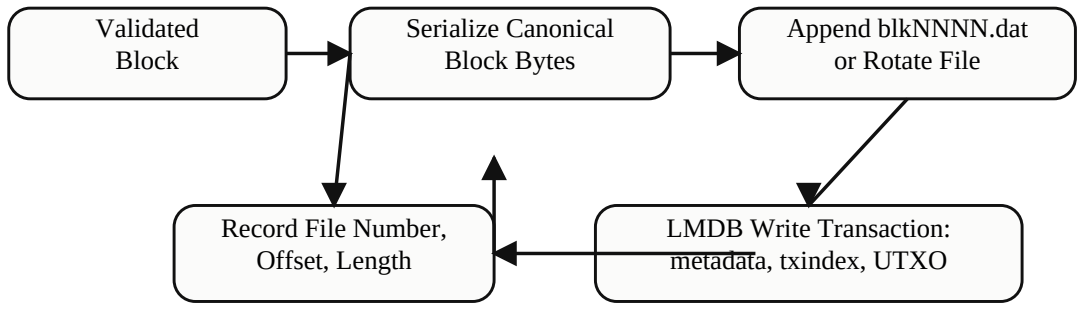

Figure 11. Hybrid Storage Commit Model

After a block is validated, the node serializes canonical block bytes, appends them to the active flat block archive or rotates to the next file, records file offsets, and then performs an LMDB-backed write transaction for metadata, indexes, and UTXO changes. The important design property is not the file layout by itself; it is that accepted local truth remains reconstructable and queryable without redefining consensus.

18. Wallet Architecture

Atho wallets derive keys, build addresses, select UTXOs, estimate fees, create change, sign transaction witnesses, and track balances relative to node-reported state. Wallet code should be conservative: it should never create transactions that consensus rejects under normal conditions when all necessary local data is available.

The live repository also separates wallet responsibilities from node responsibilities. The wallet owns keys and local intent. The node owns validation, relay, mempool policy, and block acceptance.

18.1. Wallet-to-Node Interaction

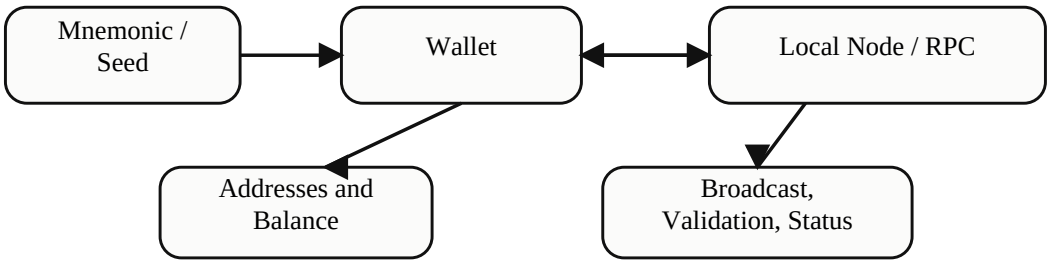

Figure 9. Wallet-to-Node Interaction

The normal wallet flow begins with mnemonic or seed material, derives local keys and receive/change addresses, builds a candidate transaction, signs it with Falcon-512, submits it to the local node or API surface, and then listens for status updates, balance changes, and confirmation depth through validated node state.

19. Address and Encoding Design

Atho uses user-facing Base56 address encoding derived from a 32-byte payment digest. The visible address carries a network-specific prefix and checksum. The internal consensus lock is not the visible string itself. The visible string is an interface encoding for the lock digest.

That separation matters:

- wallets and users interact with Base56 addresses
- consensus validates the canonical underlying digest
- network identity is visible in the address prefix
- decoding failures reject malformed user input before transaction construction

Table 10 summarizes the design tradeoffs of the current address format.

Table 10

Address Design Tradeoffs

Address Feature	Benefit	Constraint or Tradeoff
Base56 visible encoding	Avoids visually ambiguous characters and remains user-facing.	Slightly less conventional than Base58/Bech32 for outside tooling.
Visible network prefix	Makes network selection human-readable at the string level.	Wallet and exchange tooling must preserve prefix checks strictly.
32-byte payment digest	Matches canonical ownership binding in validation.	Visible address is an encoding of ownership, not ownership itself.
Fixed checksum suffix	Rejects malformed or mistyped input before transaction build.	Checksum is for input safety, not a substitute for consensus validation.
Separate internal lock vs. visible address	Keeps consensus bytes independent from UI encoding.	Developers must avoid treating display strings as canonical ledger state.

20. API and Developer Tooling

The project exposes multiple developer-facing layers:

- local launchers for `mainnet`, `testnet`, and `regnet`
- node APIs and RPC-like surfaces
- wallet tooling
- benchmarking and adversarial harness binaries
- desktop client integration

Developer tooling is valuable only when it routes into the same truth model as the node. Raw transaction submission must still hit canonical validation. Local mining helpers must still produce blocks that full validation accepts. Explorer endpoints must still reflect validated chainstate.

20.1. Practical API Boundaries

Public or semi-public APIs should:

- return validated chain and mempool information
- reject malformed raw transactions
- reject wrong-network payloads
- avoid exposing secret wallet material
- reflect the same confirmation and maturity policy enforced in code

21. Explorer and Indexer

Explorer and indexer surfaces are presentation layers over validated node data. Their role is to help users, exchanges, and operators inspect blocks, transactions, addresses, supply behavior, and network health. They do not define spendability, and they must not present invalid or legacy data as if it were canonical.

Because Atho already separates raw block archives from indexed metadata and UTXO state, explorer-style queries can be served from indexed products without reopening every raw historical block file for normal lookups.

22. Security Model

Atho’s security model combines several assumptions:

- proof-of-work ordering remains costly to rewrite without sufficient hashpower
- full nodes validate blocks and transactions independently
- Falcon-512 witness verification remains correctly implemented
- canonical ownership binding prevents loose script-based spending ambiguity
- network identities remain separated
- storage commits remain durable and coherent
- wallets protect secret material and do not leak seeds or signing keys

The security model should be described honestly. Atho is not “quantum-proof forever.” It is a payment chain whose transaction authorization model is built around a post-quantum signature scheme and whose consensus implementation is designed to fail closed on malformed or invalid state transitions.

Table 11 summarizes the main threat categories and the repository’s current defensive posture.

Table 11

Atho Threat Model

Threat Category	Defensive Posture	Remaining Dependency
Double-spend attempts	UTXO existence checks, duplicate-input rejection, and chain selection by validated proof-of-work.	Full nodes must remain widely deployed and honest enough to validate independently.
Unauthorized spend attempts	Canonical 32-byte lock binding plus Falcon-512 verification over rebuilt signing digests.	Wallets and nodes must continue to agree on exact signing-message bytes.
Wrong-network replay	Network-specific IDs, visible address prefixes, genesis anchors, and P2P magic values.	Operators must not intentionally disable network checks or mix state directories.
Malformed peer or API input	Strict decoders, size bounds, and typed validation errors.	Coverage still depends on continuing regression and adversarial testing.
Storage corruption or partial state drift	Hybrid block-file plus LMDB model with explicit chainstate indexing.	Crash-consistency and recovery behavior still need continuous validation.
Wallet secret exposure	Local signing, zeroizing memory for key material, password-based wallet encryption support, and UI/runtime defaults that require encryption.	The low-level datafile format still treats an empty passphrase as explicit plaintext mode, so production flows must keep encryption required.

23. Performance and Scalability

Performance in Atho comes from removing wasted work rather than weakening validation. Important hot paths include:

- canonical transaction decode
- UTXO lookups
- lock-digest ownership checks
- Falcon verification
- mempool conflict tracking
- block validation
- commitment-root reconstruction
- storage commits
- sync scheduling and relay

The project’s performance posture should continue to favor:

- batching safe work where possible
- caching exact validation products only when correctness is preserved
- avoiding redundant serializations
- limiting lock contention around mempool and storage access
- parallelizing independent witness verification without reordering final state transitions

24. Testing, Auditing, and Benchmarking

Atho benefits from layered verification:

- unit tests for protocol constants, encoding, hashing, signatures, and network identity
- integration tests for wallet, node, mempool, and storage paths
- adversarial tests for malformed inputs and replay attempts
- benchmark coverage for Falcon verification, decode paths, block validation, and storage commit behavior
- design reviews and audit reports for consensus correctness, Falcon handling, and production readiness

The latest local audit posture is no longer the older “mainnet blocked by known critical consensus bugs” state. The current assessment is **mainnet delay recommended**: no active invalid-block acceptance, inflation, double-spend, signature-bypass, or work-selection failure was reproduced in the normal path, but the remaining proof net is still thinner than a production release should accept. The areas that need the most proof before launch are scheduled fuzz execution, wallet/node and miner/validator differential testing, reorg crash-window fault injection, signed or otherwise stronger snapshot distribution, and final release-gate automation.

Table 12 lists the minimum testing areas that should remain in view for public network operation.

Table 12

Required Test Coverage Matrix

Test Area	What Must Hold	Why It Matters
Transaction decode	Malformed bytes reject deterministically	Prevents parser ambiguity and bypasses.
Ownership binding	Wrong key or wrong lock always fails	Protects spend authorization.
Monetary validation	Overpaying coinbase blocks reject	Preserves emission rules.
Network isolation	Wrong-network addresses and blocks reject	Prevents cross-network contamination.
Storage durability	Accepted state is reconstructable after restart	Preserves node trust in local chainstate.
Falcon verification	Malformed signatures and keys reject safely	Preserves authorization correctness.

Rust safety requirements also deserve explicit long-form tracking because the implementation is part of the security model, not just a vehicle for the protocol. Table 13 identifies the coding standards most relevant to consensus-critical review.

Table 13

Consensus-Critical Rust Review Standards

Rust Review Requirement	Why It Matters	Current Application Area
`#![forbid(unsafe_code)]` or equivalent in core crates	Shrinks the memory-unsafety surface for validation and state code.	`atho-core`, `atho-storage`, `atho-node`, and related binaries.
Typed enums for network and validation state	Prevents stringly typed rule branching.	Network IDs, versions, launch modes, and error surfaces.
Canonical serialization functions owned by protocol types	Prevents drift between wallet, node, and storage byte layouts.	Transactions, witnesses, blocks, addresses, and digests.
No panic-driven validation logic on hostile input	Keeps peer and API input from crashing the process.	P2P codec, address decode, raw transaction decode, and witness parsing.
Explicit activation scheduling for future rules	Makes dormant upgrades inspectable before activation.	Ruleset V2 placeholder and version checks.

25. Governance and Upgrade Philosophy

The strongest version of Atho is one where changes are explicit, reviewed, and traceable to code. Consensus-critical changes should not be smuggled through convenience flags, silent compatibility layers, or conflicting documentation. If the network changes a constant, changes a genesis profile, changes confirmation depth, changes maturity, or changes monetary rules, the change should be treated as a first-class protocol event.

The upgrade philosophy should therefore stay conservative:

- keep consensus rules centralized
- avoid ambiguous compatibility layers
- treat genesis and network identity changes as real resets when necessary
- keep wallet, API, explorer, and mining behavior aligned with node validation
- update public documentation only after the code truth is known

26. Roadmap

The current codebase is already beyond a toy prototype, and the one-month stable testnet signal is meaningful. The roadmap should now be read as a launch-hardening sequence rather than a feature wishlist:

- **Phase 1 — Current stable testnet:** continue public testnet operation, monitor propagation, mempool pressure, restart behavior, wallet UX, and explorer/API latency.

- **Phase 2 — Final hardening pass:** close fuzz runtime, differential, reorg-crash, snapshot-trust, and release-gate gaps.
- **Phase 3 — Mainnet release candidate:** verify live seed infrastructure, final operator configuration, reward-address procedures, release signing, and external review notes.
- **Phase 4 — Mainnet target:** late July to late August 2026 if the prior gates remain green.
- **Phase 5 — Ecosystem growth:** support exchange integrations, merchant tools, payment infrastructure, and long-term maintenance practices.

27. Conclusion

Atho is designed as a proof-of-work payment network whose strongest traits are not maximal complexity, but explicitness, deterministic validation, post-quantum-aware transaction authorization, and operator-reviewable rules. The current implementation uses SHA3-384 mining, Falcon-512 witnesses, canonical ownership binding, Base56 visible addresses, network-specific genesis anchors, 100-second blocks, 1-confirmation normal transaction consensus spendability, 100-block coinbase maturity, and a 50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 emission path with perpetual tail issuance.

The value proposition is therefore clear: Atho is trying to become durable digital money for a quantum-aware era by being strict, understandable, and locally verifiable. The public testnet's roughly one month of stable operation supports the current direction, but mainnet should remain gated by proof rather than optimism. If the project keeps code, policy, documentation, and network behavior aligned through the final two-to-three-month hardening window, it will remain much easier to audit than systems whose complexity outgrew their operator visibility.

References

1 Atho source repository, current code and documentation in this workspace.

2 National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (FIPS 202).*

3 Falcon Team. *Falcon: Fast-Fourier Lattice-Based Compact Signatures Over NTRU.*

4 Rust Project Developers. *The Rust Programming Language* and related reference documentation.

5 LMDB project documentation for memory-mapped key-value storage behavior.

Appendix A. Glossary

- **atom:** the smallest indivisible accounting unit in Atho; 1 ATHO equals 100,000,000 atoms.
- **Base56:** the user-facing visible address alphabet and encoding used by Atho addresses.
- **coinbase maturity:** the number of confirmations required before a mined subsidy output may be spent.
- **Falcon-512:** the lattice-based post-quantum signature scheme used for Atho transaction authorization.
- **founder-hash metadata:** fixed `SHA3-384` and `SHA3-512` fields included in canonical Atho block headers.
- **LMDB:** the key-value store used for indexed chainstate and metadata persistence.
- **tail emission:** the perpetual 0.390625 ATHO reward floor that begins at block 8,820,000.
- **UTXO:** unspent transaction output; the accounting object that can later be spent exactly once.

Appendix B. Protocol Constants

Table 14

Protocol Constants by Network

Table 14

Protocol Constants by Network

Constant	Mainnet	Testnet	Regnet	Prunetest
Consensus identity	Distinct mainnet ID	Distinct testnet ID	Distinct regnet ID	Distinct prunetest ID
Target block time	100 seconds	100 seconds	100 seconds	100 seconds
Max block virtual size	3,000,000 vbytes	3,000,000 vbytes	3,000,000 vbytes	3,000,000 vbytes
Max block weight	12,000,000 weight units	12,000,000 weight units	12,000,000 weight units	12,000,000 weight units
Max serialized block size	12,000,000 bytes	12,000,000 bytes	12,000,000 bytes	12,000,000 bytes
Sizing model	SegWit-style vbytes	SegWit-style vbytes	SegWit-style vbytes	SegWit-style vbytes
Normal transaction consensus spendability	1 confirmation	1 confirmation	1 confirmation	1 confirmation
Default wallet confirmation filter	3 confirmations	3 confirmations	3 confirmations	3 confirmations
Coinbase maturity	100 blocks	100 blocks	100 blocks	100 blocks
Proof-of-work hash	SHA3-384	SHA3-384	SHA3-384	SHA3-384
Signature scheme	Falcon-512	Falcon-512	Falcon-512	Falcon-512
Address format	Base56 with mainnet prefix	Base56 with testnet prefix	Base56 with regnet prefix	Base56 with prunetest prefix
Founder hashes	Fixed canonical header values	Fixed canonical header values	Fixed canonical header values	Fixed canonical header values

Appendix C. Code Reference Map

Table 15

Code Reference Map

Table 15

Code Reference Map

Module or File	Role in the Current Repository
`crates/atho-core/src/constants.rs`	Consensus constants for timing, subsidy, confirmations, and units.
`crates/atho-core/src/network.rs`	Network IDs, visible prefixes, ports, P2P magic values, and launch policy boundaries.
`crates/atho-core/src/block.rs`	Canonical block header encoding, founder-hash fields, and block serialization.
`crates/atho-core/src/transaction.rs`	Canonical transaction and witness object model.
`crates/atho-core/src/address.rs`	Base56 address derivation, checksum logic, and decode rules.
`crates/atho-core/src/consensus/rules.rs`	Active protocol/ruleset versions and future placeholder scheduling.
`crates/atho-core/src/consensus/subsidy.rs`	Reward schedule, yearly emission logic, and absence of a finite max-supply cap.
`crates/atho-core/src/consensus/pow.rs`	Difficulty targeting and proof-of-work validation rules.
`crates/atho-core/src/consensus/signatures.rs`	Canonical signing-message construction and witness verification context.
`crates/atho-crypto/src/falcon.rs`	Falcon-512 key and signature wrappers plus verification behavior.
`crates/atho-storage/src/validation.rs`	Contextual transaction and block validation against chainstate.
`crates/atho-storage/src/block_files.rs`	Flat block archive layout, file rotation, and offset bookkeeping.
`crates/atho-wallet/src/wallet.rs`	Wallet state, address derivation, and transaction construction.
`crates/atho-node/src/api.rs`	Node-facing API surface and reporting.
`crates/atho-p2p/src/config.rs`	Network bootstrap configuration and peer limits.

Module or File	Role in the Current Repository
`crates/atho-rpc`	Typed local RPC transport and request/response model.
`crates/atho-installer`	Release installation and distribution tooling.

Appendix D. Transaction Validation Pseudocode

```
function validate_transaction(raw_bytes, network, utxo_view):
    tx = decode_canonical_transaction(raw_bytes)
    reject_if_trailing_bytes(raw_bytes, tx)
    reject_if_wrong_network(tx, network)
    reject_if_duplicate_inputs(tx.inputs)
    reject_if_invalid_output_amounts(tx.outputs)
    reject_if_invalid_fee_floor(tx)

    resolved_inputs = batch_lookup_utxos(tx.inputs, utxo_view)
    reject_if_missing_inputs(resolved_inputs)
    reject_if_immature_coinbase_spend(resolved_inputs)

    for input in tx.inputs:
        witness = parse_witness(input.witness)
        reject_if_malformed_witness(witness)
        reject_if_lock_not_canonical(resolved_inputs[input].lock_digest)
        reject_if_public_key_does_not_match_lock(witness.public_key, resolved_inputs[input].lock_diges
        digest = rebuild_signing_digest(tx, input.index, resolved_inputs[input], network)
        reject_if_falcon_verification_fails(witness.public_key, witness.signature, digest)

    reject_if_created_value_exceeds_available_value(tx, resolved_inputs)
    return validated_transaction
```

Appendix E. Block Validation Pseudocode

```
function validate_block(raw_block_bytes, network, chainstate):
    block = decode_canonical_block(raw_block_bytes)
    reject_if_wrong_network(block.header, network)
    reject_if_bad_header_structure(block.header)
    reject_if_invalid_pow(block.header)
    reject_if_bad_timestamp_or_target(block.header, chainstate)
    reject_if_bad_coinbase_position(block.transactions)
    reject_if_duplicate_txids(block.transactions)
    reject_if_duplicate_spends(block.transactions)

    utxo_batch = batch_lookup_all_inputs(block.transactions, chainstate)
    fees = 0
    for tx in block.transactions excluding coinbase:
        validated_tx = validate_transaction(tx.raw_bytes, network, utxo_batch)
        fees += validated_tx.fee

    reject_if_coinbase_overpays(block.coinbase, block.height, fees)
    reject_if_bad_commitment_roots(block)
    atomically_apply_block(block, utxo_batch, chainstate)
    return accepted_block
```

Appendix F. Mempool Admission Pseudocode

```
function admit_to_mempool(raw_tx_bytes, network, chainstate, mempool):
    validated_tx = validate_transaction(raw_tx_bytes, network, chainstate.utxo_view)
    reject_if_mempool_conflict(validated_tx.inputs, mempool)
    reject_if_policy_invalid(validated_tx)
    mempool.insert(validated_tx)
    return admitted
```

Appendix G. Flowchart Source Text

The figures in this edition are rendered as monochrome report-style diagrams directly from the local PDF generator rather than external colored assets. Their intended meanings are:

- **Figure 1:** wallet creation to mempool admission to mining to durable storage and downstream sync.
- **Figure 2:** separation between P2P, API/RPC, mempool, consensus, storage, mining, and wallet boundaries.
- **Figure 3:** canonical signing flow from UTXO selection to node-side digest reconstruction and Falcon verification.
- **Figure 4:** full transaction lifecycle from wallet creation through settlement.
- **Figure 5:** ordered block validation pipeline from decode to atomic commit.
- **Figure 6:** subsidy path across the active 50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 schedule.
- **Figure 7:** mempool admission path from raw input to conflict tracking and admit/reject decision.
- **Figure 8:** node synchronization and block propagation sequence.
- **Figure 9:** wallet-to-node interaction from seed material to status updates.
- **Figure 10:** parallelizable validation work distribution and ordered final commit.
- **Figure 11:** hybrid block-file plus LMDB commit model.