# ATHO

Atho — The Platinum Standard of the Quantum Age.

Comprehensive Technical Whitepaper: Quantum-Era Security, Consensus Economics, and Core Implementation Architecture

Generated: 2026-04-10
Network: mainnet
Consensus Version: 1.00
Status: Technical reference; implementation-grounded (Alpha pre-release snapshot)

Disclaimer: This document is technical in nature and does not constitute legal, tax, investment, or financial advice. All monetary scenarios are model-based and depend on explicit assumptions. Consensus-critical values are authoritative only as implemented in Src/Utility/const.py and validation code paths.

# Table of Contents

# Part I - Strategic, Scientific, and Economic Foundations

This part is writing-first and research-first. It explains why Atho exists, how quantum risk changes blockchain engineering requirements [FN-01], why specific cryptographic choices were made, and how policy design aligns with long-run security and usability goals.

## How to Read This Whitepaper

This guide is designed so technical and non-technical readers can follow the same document without losing context. Use the path below that matches your role, then return to the full pass for complete coverage.

- Quick Executive Path (20-30 min): Read Executive Abstract, Threat Convergence summary, and Part II consensus tables/charts. This gives mission, risk model, and live policy numbers fast.
- Security Reviewer Path (60-90 min): Read quantum-risk chapters, Part II constants, then Part III invariants and Runtime Guard docs coverage. Focus on fail-closed behavior and upgrade discipline.
- Protocol Engineer Path (90-150 min): Read Part I context, then Part II sizing/issuance charts, then Part III module map and condensed source extracts for code-level alignment.
- Economics and Policy Path (45-75 min): Read emissions and utilization chapters, then modeling atlas figures and consensus metrics table. Validate threshold assumptions against constants.
- How to Use Footnotes: Markers [FN-xx] point to short plain-language explanations near the end of this document. Use them when terms are unfamiliar.

If a term is unfamiliar, use the explanatory footnotes and end-of-document keyword index [FN-02].

## Atho Alpha Focus: What We Solve and Why It Matters

Atho's alpha focus is straightforward: remove ambiguity from the critical path. In many blockchain systems, policy documents, node behavior, and operator assumptions drift over time. That drift creates hidden risk because users think one rule set is active while code actually enforces another. Atho addresses this by binding narrative claims to deterministic constants, deterministic validators, and reproducible logs so protocol behavior is observable rather than implied.

The first core problem Atho solves is long-horizon cryptographic exposure planning. Instead of treating post-quantum migration as a future emergency, Atho treats it as current architecture work. Falcon-based signing pathways, verifier pinning controls, and explicit release discipline are designed to reduce late-stage migration debt. This is not a marketing distinction; it is an operational one. If migration pressure arrives suddenly, systems with pre-integrated controls can respond with policy updates, while legacy-first systems may require ecosystem-wide rewrites.

The second problem is economic opacity. Atho formalizes reward schedule behavior, fee floors, burn behavior, and floor clipping using constants and integer math. That means miners, wallets, integrators, and auditors can independently verify how policy executes over time. Predictable fee logic also helps user operations because transaction cost planning is linked to byte footprint and chain policy, not opaque runtime pricing behavior. The value is not lower complexity for its own sake; the value is lower uncertainty in core financial flows.

The third problem is operational safety during growth. Alpha networks are where hidden race conditions, stale assumptions, and weak observability usually appear. Atho focuses on controls that improve real operator outcomes: bounded rollback, explicit guard rails around upgrades, structured log evidence, and clearer node lifecycle discipline. This approach keeps incident response practical. Instead of debugging by folklore, operators can follow deterministic traces and known checkpoint behavior.

Atho also prioritizes user-facing trust mechanics. Addresses, transaction states, pending behavior, and confirmation visibility are part of security communication, not only UX polish. If users cannot verify what happened, they develop unsafe habits and support overhead climbs quickly. That is why the project pairs protocol-level discipline with interface-level clarity in the GUI and Web Explorer. Good security and good operations both depend on predictable, inspectable state.

Another strategic difference is scope discipline. Atho is intentionally reducing broad narrative overhead in this alpha whitepaper and concentrating on enforceable mechanisms. The aim is to ship fewer claims and stronger guarantees. Every major section in this document maps to one of three things: an active consensus rule, an operational control with measurable output, or a model assumption that is explicitly labeled. This structure helps partners evaluate readiness without navigating unnecessary noise.

From a deployment standpoint, Atho's target is production-grade behavior built in layers. First layer: deterministic acceptance rules. Second layer: reproducible storage and recovery semantics. Third layer: runtime integrity and release discipline. Fourth layer: observability that can prove policy is active under live conditions. This layering matters because failures rarely happen in only one subsystem. Strong chains align cryptography, economics, storage, and operations under one evidence model.

The alpha objective is therefore concrete: finalize a chain where security assumptions, economic behavior, and operator workflows are coherent enough for serious integration testing. That requires continued hardening, but the direction is clear and measurable. Atho's core value proposition is not that risk disappears; it is that risk becomes explicit, bounded, and actionable with deterministic tools.

**Problem -> Mechanism -> Outcome (Alpha Lens)**

| Problem | Atho Mechanism | Expected Outcome |
|---|---|---|
| Quantum-era signature migration debt | Falcon verification path + digest pinning controls + version discipline | Lower emergency migration risk and cleaner upgrade windows |
| Policy/implementation drift | Deterministic constants + canonical validation paths + structured checks | Consistent node behavior and reduced fork-risk from ambiguity |
| Unclear monetary behavior | Explicit reward/fee/burn/floor equations under integer arithmetic | Auditable supply behavior and predictable fee economics |
| Operational incident chaos | Runtime guard + rollback bounds + high-signal audit logs | Faster triage and bounded recovery actions |
| User confidence gaps | Clear pending/confirmed UX + searchable explorer state + copy-safe flows | Lower error rate and stronger day-to-day trust signals |

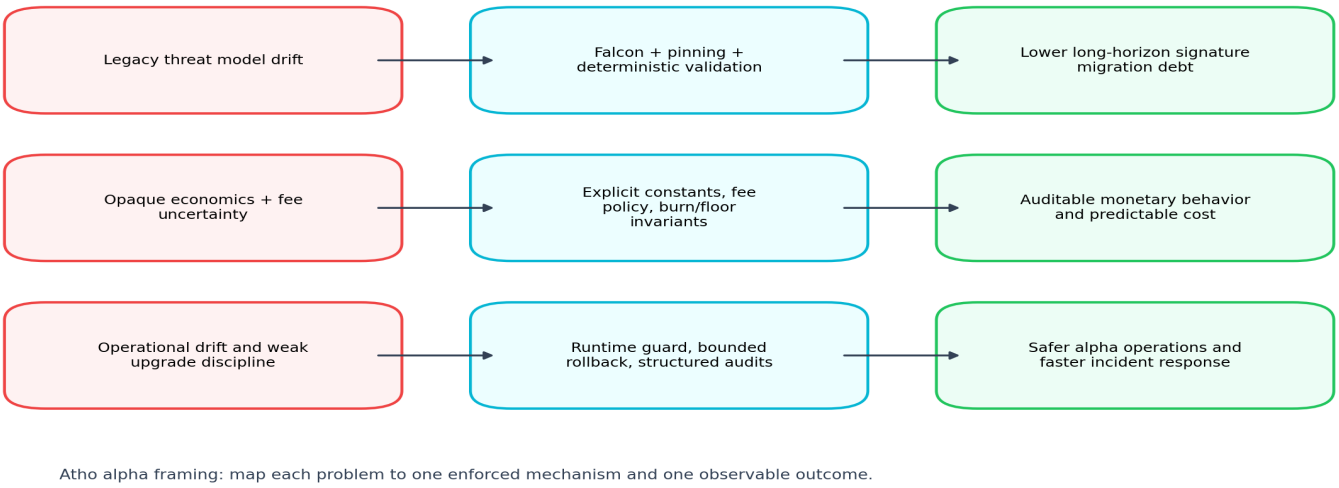



*Figure 3B: Atho alpha problem-to-mechanism mapping with operator-visible outcomes.*

## Quantum Intuition Visuals

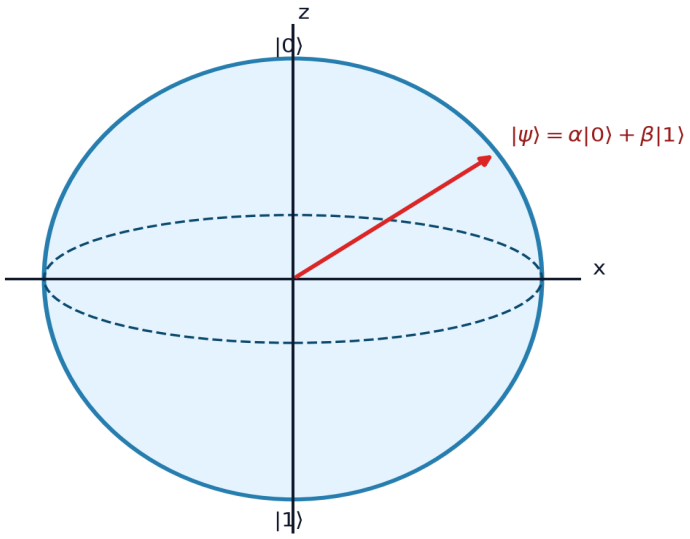



*Figure 1: Stylized Bloch-sphere view of one qubit state. This is a conceptual guide, not hardware geometry.*

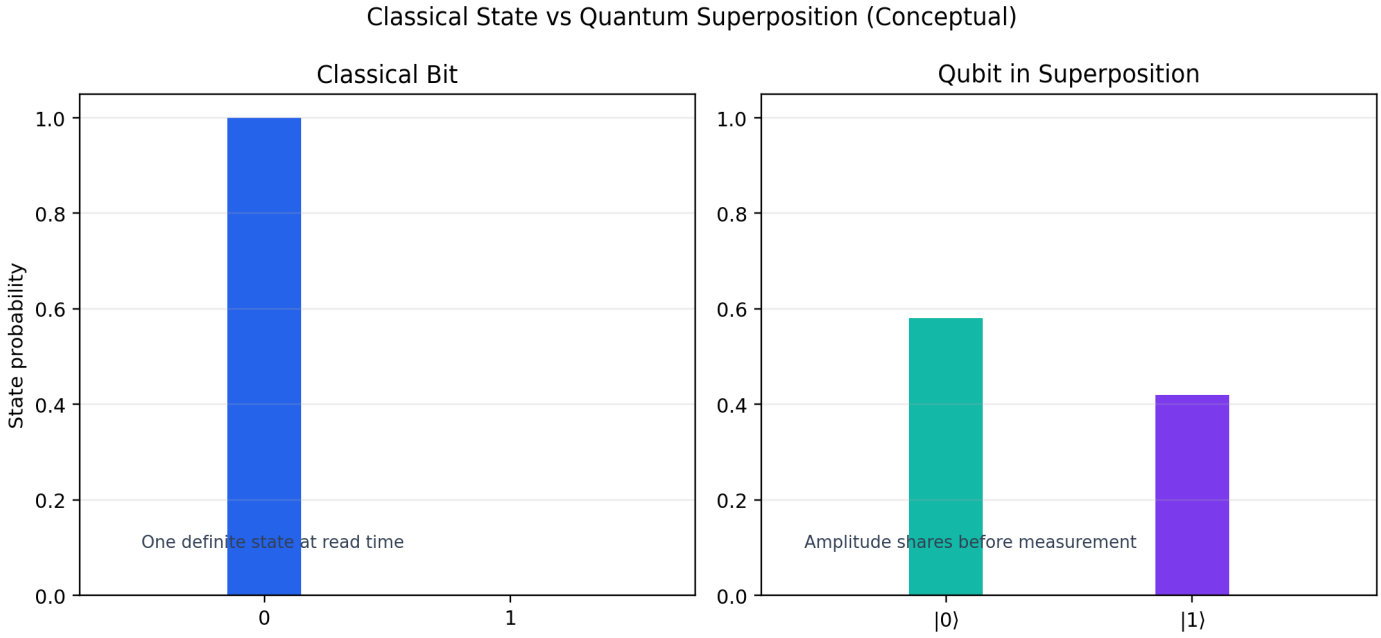



*Figure 2: Classical bit states versus qubit superposition probabilities before measurement.*

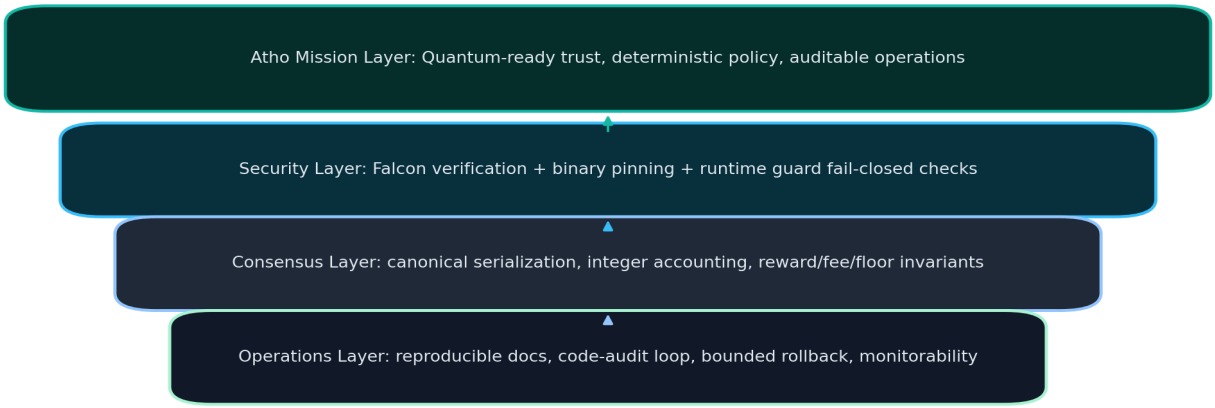



*Figure 3: Atho-focused architecture priorities for the alpha pre-release whitepaper.*

# Executive Abstract and Project Intent

Atho Labs created Atho to solve a structural mismatch in today's blockchain stack: long-duration monetary systems are still largely anchored to classical-signature assumptions while adversary capability is moving toward quantum-assisted regimes. The primary thesis is simple: if a chain is intended to survive multiple decades, post-quantum resilience must be introduced as a first-order design objective rather than a late emergency migration.

This document is deliberately top-to-bottom and research-first. It starts with quantum mechanics and cryptographic risk models, then maps those concerns to legacy chain limitations, then explains why Atho made specific protocol choices, and only after that moves into implementation-level appendix material. That separation is intentional so the strategic and scientific narrative remains readable before code-level details are presented.

Atho's identity is built on three pillars. First, cryptographic modernization through Falcon-512 signing pathways with strict binary pinning controls in required networks. Second, deterministic consensus economics with explicit tail emission, utilization-sensitive fee burn behavior, and a hard floor invariant.

The standard of quality for this whitepaper is not stylistic depth alone. It is whether each major claim can be traced to a verifiable mechanism: a consensus constant, a validator check, a storage invariant, an economic equation, or a release control. Where this document discusses external research, it does so to justify design pressure; where it discusses Atho internals, it does so to show explicit enforcement.

# How Quantum Computing Works and Why It Changes Security Planning

Quantum computing is not a faster version of classical computing; it is a different model of computation built on superposition, interference, and entanglement. Classical bits encode one state at a time, while qubits can encode amplitudes over multiple basis states. Algorithmic advantage appears when a quantum circuit uses interference to amplify amplitudes corresponding to correct solutions and suppress incorrect ones.

In practical cryptography discussions, the concern is not generic speedup across all tasks. The concern is selective speedup for mathematically structured problems that underpin common security assumptions. Discrete logarithm and integer factorization are key examples.

For symmetric primitives and hash functions, the threat is different. Grover-style search yields roughly quadratic speedup in idealized conditions, so effective security margins are reduced by about half in bits under that model. This does not instantly break modern hash functions, but it changes prudent parameter sizing and long-term policy decisions for systems expected to remain secure over decades.

Blockchain systems are uniquely exposed to this planning problem because they are transparent, durable, and value-dense. Public keys, signatures, transaction graphs, and state history are visible and archived. An adversary does not need to break everything at once; they can target high-value exposure windows and exploit weak migration readiness.

# Executive Summary: Threat Convergence and Atho's Response

The convergence of quantum computing and AI-assisted optimization is now a strategic planning input for blockchain systems with multi-decade lifetimes. Even with uncertainty in timeline estimates, the engineering direction is clear: classical elliptic-curve dependence creates accumulating migration debt, and that debt becomes harder to repay as ecosystems scale.

This whitepaper treats the problem in six layers: quantum foundations, algorithmic cryptanalysis, classical-chain exposure, migration-timeline reality, Atho architectural controls, and continuous upgrade governance. Each layer is tied to concrete protocol behavior and operational controls so the final result is verifiable rather than purely rhetorical.

Atho's solution posture is to integrate post-quantum-capable authenticity pathways, deterministic consensus arithmetic, strict binary provenance controls, and explicit monetary observability from genesis-era architecture onward. This avoids relying on emergency retrofits after infrastructure lock-in and allows security policy to tighten over time without breaking policy traceability.

The document also separates market inference from protocol fact. Where it discusses economic scenarios, it labels assumptions; where it discusses consensus behavior, it references deterministic constants and validation pathways. This distinction is critical for credible policy communication with miners, operators, and ecosystem integrators.

# Qubit Error Correction, Fault Tolerance, and Security Timelines

Qubit Error Correction, Fault Tolerance, and Security Timelines is central to Atho Labs' system design because security budgets must account for when fault-tolerant machines may become practical, not only for current hardware limits. Traditional crypto systems generally assume many existing chains assume today's classical hardness profile and defer cryptographic migration to future governance, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is security teams underestimate schedule risk, producing migration deadlines that are too late for real ecosystem coordination. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, qubit error correction, fault tolerance, and security timelines intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. many existing chains assume today's classical hardness profile and defer cryptographic migration to future governance Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

# Shor-Type Threats to Elliptic-Curve Signature Systems

Shor-Type Threats to Elliptic-Curve Signature Systems is central to Atho Labs' system design because discrete-log speedups directly challenge elliptic-curve signature assumptions used in major chains. Traditional crypto systems generally assume Bitcoin, Litecoin, and Ethereum are rooted in classical elliptic-curve signature ecosystems, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is exposed public keys become forgery targets once practical key extraction pathways emerge. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, shor-type threats to elliptic-curve signature systems intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. Bitcoin, Litecoin, and Ethereum are rooted in classical elliptic-curve signature ecosystems Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Grover-Type Effects on Hash Security Margins

Grover-Type Effects on Hash Security Margins is central to Atho Labs' system design because hash function choices should include future effective security margin analysis, not only present-day collision statistics. Traditional crypto systems generally assume chains often document hash usage without explicitly discussing quantum-adjusted margins, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is underestimated preimage margins can weaken confidence in long-horizon archival and commitment structures. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, grover-type effects on hash security margins intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. chains often document hash usage without explicitly discussing quantum-adjusted margins Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Harvest-Now, Forge-Later Exposure Model

Harvest-Now, Forge-Later Exposure Model is central to Atho Labs' system design because adversaries can archive public artifacts now and exploit them later if migration is delayed. Traditional crypto systems generally assume many systems treat signature risk as a same-day event instead of a deferred exploitation pipeline, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is archived key material and transaction metadata can become actionable under future capabilities. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, harvest-now, forge-later exposure model intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. many systems treat signature risk as a same-day event instead of a deferred exploitation pipeline Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Address and Public-Key Exposure Windows in UTXO Systems

Address and Public-Key Exposure Windows in UTXO Systems is central to Atho Labs' system design because risk depends on when public keys are exposed, how often they are reused, and how tooling handles key lifecycle. Traditional crypto systems generally assume wallet behavior in legacy chains can still leak or reuse key material in operationally weak ways, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is key reuse and delayed migration expand attack windows even before full cryptanalytic breakpoints. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, address and public-key exposure windows in utxo systems intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic

migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. wallet behavior in legacy chains can still leak or reuse key material in operationally weak ways Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Bitcoin and Litecoin: Resilience Strengths and Quantum Migration Gaps

Bitcoin and Litecoin: Resilience Strengths and Quantum Migration Gaps is central to Atho Labs' system design because classical PoW chains are robust operationally but still face signature-layer migration complexity. Traditional crypto systems generally assume security assumptions center on secp256k1 ECDSA and ecosystem-wide compatibility inertia, and that assumption held under classical adversary models.

The technical concern in this area is network-scale migration can be socially and operationally harder than protocol-level cryptographic updates. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Ethereum and Account-Model Signature Surfaces

Ethereum and Account-Model Signature Surfaces is central to Atho Labs' system design because account-based ecosystems carry different exposure dynamics because signatures and key recovery are deeply integrated. Traditional crypto systems generally assume EOA workflows rely on classical elliptic-curve signatures with extensive external tooling dependence, and that assumption held under classical adversary models.

The technical concern in this area is migration complexity compounds across wallets, contracts, relayers, and exchange infrastructure. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Why Falcon-512: Compactness, Verification Path, and Practical Deployment

Why Falcon-512: Compactness, Verification Path, and Practical Deployment is central to Atho Labs' system design because signature scheme selection must balance security assumptions, size, and implementation ergonomics. Traditional crypto systems generally assume classical schemes are mature operationally but tied to assumptions quantum research targets, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is choosing only for familiarity can produce long-term security debt. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, why falcon-512: compactness, verification path, and practical deployment intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. classical schemes are mature operationally but tied to assumptions quantum research targets Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## NIST Standardization Context and Cryptographic Governance

NIST Standardization Context and Cryptographic Governance is central to Atho Labs' system design because standards-track alignment reduces bespoke risk and improves external auditability. Traditional crypto systems generally assume custom cryptographic stacks without standards context increase review burden and trust assumptions, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is non-standard choices can create maintenance traps and interoperability friction. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for

implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, nist standardization context and cryptographic governance intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. custom cryptographic stacks without standards context increase review burden and trust assumptions Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Why SHA3-384 in Core Hashing Paths

Why SHA3-384 in Core Hashing Paths is central to Atho Labs' system design because hash selection should reflect long-horizon preimage margin, implementation clarity, and deterministic serialization. Traditional crypto systems generally assume many ecosystems rely on earlier hash families with different design assumptions, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is under-documenting hash rationale weakens confidence in commitment semantics. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, why sha3-384 in core hashing paths intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. many ecosystems rely on earlier hash families with different design assumptions Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Checksum Surfaces and Human-Facing Safety

Checksum Surfaces and Human-Facing Safety is central to Atho Labs' system design because human-readable addressing still needs robust checksum validation to reduce operational loss. Traditional crypto systems generally assume address formatting errors remain a persistent source of user-side loss across chains, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is weak checksum and normalization handling produces silent routing failures. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, checksum surfaces and human-facing safety intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. address formatting errors remain a persistent source of user-side loss across chains Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Supply-Chain Security: Binary Pinning as a Consensus-Adjacent Control

Supply-Chain Security: Binary Pinning as a Consensus-Adjacent Control is central to Atho Labs' system design because cryptographic correctness depends on binary integrity, not only algorithm labels. Traditional crypto systems generally assume path-based binary discovery without strict pinning enables replacement attacks, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is malicious signer/verifier binaries can invalidate trust even when protocol math appears sound. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, supply-chain security: binary pinning as a consensus-adjacent control intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. path-based binary discovery without strict pinning enables replacement attacks Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Deterministic Serialization and Canonical Transaction Identity

Deterministic Serialization and Canonical Transaction Identity is central to Atho Labs' system design because distributed consensus requires byte-equivalent interpretation under independent implementations. Traditional crypto systems generally assume ambiguous serialization can create silent forks or divergent txid computation, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is non-canonical field ordering and witness confusion can destabilize network convergence. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, deterministic serialization and canonical transaction identity intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. ambiguous serialization can create silent forks or divergent txid computation Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Consensus Invariants: Exact Coinbase Payout and Fee Accounting

Consensus Invariants: Exact Coinbase Payout and Fee Accounting is central to Atho Labs' system design because monetary policy is only credible when block-level payouts are exact and checkable. Traditional crypto systems generally assume loosely enforced payout logic invites drift between intended and actual issuance, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is silent overpay or accounting mismatch erodes monetary trust quickly. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, consensus invariants: exact coinbase payout and fee accounting intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. loosely enforced payout logic invites drift between intended and actual issuance Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Inflation, Deflation, and Floor-Clipped Burn Mechanics

Inflation, Deflation, and Floor-Clipped Burn Mechanics is central to Atho Labs' system design because a usage-linked monetary response should remain bounded under adverse or extreme scenarios. Traditional crypto systems generally assume fixed narratives of inflation or deflation often ignore utilization and fee-volume variability, and that assumption held under

classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is unbounded burn logic can create pathological supply outcomes. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, inflation, deflation, and floor-clipped burn mechanics intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. fixed narratives of inflation or deflation often ignore utilization and fee-volume variability Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Miner Security Budget and Network Safety

Miner Security Budget and Network Safety is central to Atho Labs' system design because chain security depends on sustainable validator/miner economics, not only issuance aesthetics. Traditional crypto systems generally assume fee-only end states can create budget volatility and uncertain attack-cost floors, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is insufficient security budget increases reorg and censorship risk. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, miner security budget and network safety intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. fee-only end states can create budget volatility and uncertain attack-cost floors Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## Mempool, API, and P2P Attack Surface Management

Mempool, API, and P2P Attack Surface Management is central to Atho Labs' system design because network exposure paths must be considered alongside cryptography and economics. Traditional crypto systems generally assume many systems under-document operational limits for API and gossip abuse paths, and that assumption held under classical adversary models. However, the security planning horizon for a monetary protocol is not one release cycle; it spans decades.

The technical concern in this area is flooding, replay, and transport abuse can degrade liveness without violating consensus math. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes. A purely theoretical discussion is insufficient; design choices must account for implementation friction, release workflow, and the tendency of production systems to drift from ideal assumptions.

From a quantum-threat perspective, mempool, api, and p2p attack surface management intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives. The appropriate response is not panic migration or denial; it is structured risk budgeting.

Legacy chain behavior illustrates the challenge clearly. many systems under-document operational limits for API and gossip abuse paths Bitcoin and Litecoin rely on secp256k1-based ECDSA flows, and Ethereum relies on classical elliptic-curve signatures across externally owned accounts. When quantum-capable key extraction becomes practical, any exposed public key can become a potential forgery target if migration has not already occurred.

## LMDB State Durability and Consensus Write Discipline

LMDB State Durability and Consensus Write Discipline is central to Atho Labs' system design because state integrity is a consensus concern when storage behavior can diverge under load or failure. Traditional crypto systems generally assume improper storage guardrails can corrupt node-local truth even with valid chain data, and that assumption held under classical adversary models.

The technical concern in this area is map exhaustion, stale writes, or weak backup procedures can cause operational forks. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, lmdb state durability and consensus write discipline intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Key Management and Operational Cryptography Hygiene

Key Management and Operational Cryptography Hygiene is central to Atho Labs' system design because the strongest cryptography fails if key lifecycle controls are weak. Traditional crypto systems generally assume plaintext keys, broad file permissions, and ad-hoc export flows are common operational weaknesses, and that assumption held under classical adversary models.

The technical concern in this area is key theft and signing abuse bypass on-chain policy entirely. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, key management and operational cryptography hygiene intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Upgrade Policy: Tighten-Only Security Posture

Upgrade Policy: Tighten-Only Security Posture is central to Atho Labs' system design because security-sensitive consensus systems should prioritize tightening controls over loosening them. Traditional crypto systems generally assume frequent policy loosenings increase ambiguity and downgrade risk, and that assumption held under classical adversary models.

The technical concern in this area is inconsistent upgrade discipline can fragment trust and validator behavior. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, upgrade policy: tighten-only security posture intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Why Atho Uses a UTXO Model Instead of Generalized Smart-Contract VM Complexity

Why Atho Uses a UTXO Model Instead of Generalized Smart-Contract VM Complexity is central to Atho Labs' system design because state-model simplicity is a security strategy when deterministic auditability is a priority. Traditional crypto systems generally assume highly expressive execution layers increase surface area for semantic and economic exploits, and that assumption held under classical adversary models.

The technical concern in this area is execution complexity can obscure root-cause analysis during incident response. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, why atho uses a utxo model instead of generalized smart-contract vm complexity intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Fee Predictability and User-Cost Transparency

Fee Predictability and User-Cost Transparency is central to Atho Labs' system design because adoption requires costs that are predictable enough for users and integrators to model. Traditional crypto systems generally assume dynamic gas auctions and opaque priority markets can be difficult for users to reason about, and that assumption held under classical adversary models.

The technical concern in this area is unpredictable fees reduce trust and operational planning quality. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, fee predictability and user-cost transparency intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Auditable Burn Tracking and Supply Monitoring

Auditable Burn Tracking and Supply Monitoring is central to Atho Labs' system design because supply claims must be observable in block data and monitoring artifacts. Traditional crypto systems generally assume insufficient accounting visibility creates room for confusion or misinformation, and that assumption held under classical adversary models.

The technical concern in this area is stakeholders cannot independently verify monetary behavior. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, auditable burn tracking and supply monitoring intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Atho Labs Product Thesis: Platform Standard Over Hype Cycles

Atho Labs Product Thesis: Platform Standard Over Hype Cycles is central to Atho Labs' system design because the project is designed as infrastructure with long-lived guarantees rather than short-cycle narrative features. Traditional crypto systems generally assume many ecosystems optimize for near-term speculation rather than long-term protocol assurance, and that assumption held under classical adversary models.

The technical concern in this area is misaligned incentives can deprioritize security upgrades until incidents force them. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, atho labs product thesis: platform standard over hype cycles intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Quantum Attack Taxonomy for Public Blockchains

Quantum Attack Taxonomy for Public Blockchains is central to Atho Labs' system design because defense planning is stronger when quantum-era threats are categorized by objective and required capability. Traditional crypto systems generally assume many projects discuss quantum risk in generic terms without mapping concrete exploit classes, and that assumption held under classical adversary models.

The technical concern in this area is without taxonomy, teams mis-prioritize controls and leave high-impact windows unprotected. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, quantum attack taxonomy for public blockchains intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## How Quantum Hardware Roadmaps Influence Protocol Design Timing

How Quantum Hardware Roadmaps Influence Protocol Design Timing is central to Atho Labs' system design because protocol teams need timing strategy that accounts for long lead-time migration and ecosystem coordination. Traditional crypto systems generally assume classical chains often wait for immediate break signals before executing migration work, and that assumption held under classical adversary models.

The technical concern in this area is late migration compresses testing windows and increases probability of rushed governance decisions. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, how quantum hardware roadmaps influence protocol design timing intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Falcon-512 Versus Alternative Post-Quantum Signature Families

Falcon-512 Versus Alternative Post-Quantum Signature Families is central to Atho Labs' system design because scheme selection should be justified by explicit tradeoffs in signature size, verification behavior, and deployment friction. Traditional crypto systems generally assume many discussions pick a scheme based on popularity rather than workload fit, and that assumption held under classical adversary models.

The technical concern in this area is misaligned selection can harm throughput, storage cost, or implementation reliability. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, falcon-512 versus alternative post-quantum signature families intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Signature Size, Block Capacity, and Real Throughput Economics

Signature Size, Block Capacity, and Real Throughput Economics is central to Atho Labs' system design because cryptographic security choices directly interact with byte-level fee and capacity models. Traditional crypto systems generally assume throughput narratives often ignore signature payload impact at scale, and that assumption held under classical adversary models.

The technical concern in this area is systems can overstate TPS or understate user cost when signature overhead is omitted. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, signature size, block capacity, and real throughput economics intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Classical Chain Upgrade Friction: Governance and Ecosystem Coordination

Classical Chain Upgrade Friction: Governance and Ecosystem Coordination is central to Atho Labs' system design because security migration in large ecosystems is constrained by social and operational coordination capacity. Traditional crypto systems generally assume Bitcoin, Litecoin, and Ethereum each have heterogeneous wallet, exchange, and custody dependencies, and that assumption held under classical adversary models.

The technical concern in this area is even sound cryptography can fail deployment if ecosystem synchronization lags protocol intent. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Why Deterministic Integer Arithmetic Matters in Monetary Consensus

Why Deterministic Integer Arithmetic Matters in Monetary Consensus is central to Atho Labs' system design because floating-point ambiguity is unacceptable in distributed monetary validation. Traditional crypto systems generally assume human-facing docs can drift into decimal reasoning that is unsafe for consensus math, and that assumption held under classical adversary models.

The technical concern in this area is rounding variance can produce payout mismatches and chain divergence. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, why deterministic integer arithmetic matters in monetary consensus intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Fee Burn Visibility and Audit Integrity

Fee Burn Visibility and Audit Integrity is central to Atho Labs' system design because burn policy credibility depends on observable on-chain accounting, not only narrative claims. Traditional crypto systems generally assume systems with opaque accounting fields can create uncertainty about actual supply effects, and that assumption held under classical adversary models.

The technical concern in this area is stakeholders cannot independently validate burn execution and floor interactions. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, fee burn visibility and audit integrity intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Supply Floor Rationale and Stability Envelope

Supply Floor Rationale and Stability Envelope is central to Atho Labs' system design because a floor creates a bounded stability envelope for aggressive burn policies. Traditional crypto systems generally assume fixed-scarcity narratives without floor logic can become brittle under extreme assumptions, and that assumption held under classical adversary models.

The technical concern in this area is over-burn scenarios can damage monetary confidence if not clipped deterministically. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, supply floor rationale and stability envelope intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Censorship Resistance and Security Budget Under Tail Regimes

Censorship Resistance and Security Budget Under Tail Regimes is central to Atho Labs' system design because long-run censorship resistance depends on durable miner economics, not only issuance headlines. Traditional crypto systems generally assume fee-only end states can become highly variable and sensitive to cyclical activity, and that assumption held under classical adversary models.

The technical concern in this area is security budget shocks can lower attack cost and increase censorship feasibility. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, censorship resistance and security budget under tail regimes intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Mempool Policy, Relay Behavior, and Adversarial Traffic

Mempool Policy, Relay Behavior, and Adversarial Traffic is central to Atho Labs' system design because real-world resilience includes handling spam, replay, and malformed propagation at scale. Traditional crypto systems generally assume many protocol discussions underweight relay-layer abuse and edge-case flooding, and that assumption held under classical adversary models.

The technical concern in this area is degraded liveness and delayed confirmations can emerge without consensus break. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, mempool policy, relay behavior, and adversarial traffic intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Binary Provenance, Release Discipline, and Operator Trust

Binary Provenance, Release Discipline, and Operator Trust is central to Atho Labs' system design because operators need evidence that cryptographic binaries are authentic, version-bound, and immutable in deployment. Traditional crypto systems generally assume unverified binary updates create silent security debt in cryptographic systems, and that assumption held under classical adversary models.

The technical concern in this area is supply-chain manipulation can bypass protocol intent by altering signer/verifier behavior. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, binary provenance, release discipline, and operator trust intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Atho as a Platform Standard: Interoperability, Auditability, and Longevity

Atho as a Platform Standard: Interoperability, Auditability, and Longevity is central to Atho Labs' system design because a platform standard must remain understandable to engineers, operators, auditors, and external reviewers. Traditional crypto systems generally assume rapidly changing ecosystems often lose institutional memory and policy traceability, and that assumption held under classical adversary models.

The technical concern in this area is fragmented understanding increases implementation error and governance conflict. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

From a quantum-threat perspective, atho as a platform standard: interoperability, auditability, and longevity intersects with known algorithmic pressure points. Shor-type speedups threaten discrete-log style assumptions in elliptic-curve systems, and Grover-type speedups alter effective preimage margins for symmetric primitives.

## Bitcoin: Public-Key Exposure Windows Under Quantum Transition

Bitcoin: Public-Key Exposure Windows Under Quantum Transition is central to Atho Labs' system design because public-key exposure windows are often the decisive variable in post-quantum forgery timelines. Traditional crypto systems generally assume Bitcoin uses classical secp256k1-based ECDSA and a conservative governance culture that makes wide cryptographic migration slower to coordinate., and that assumption held under classical adversary models.

The technical concern in this area is operators fail to prioritize key-rotation and exposure minimization controls before migration pressure spikes. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Litecoin: Public-Key Exposure Windows Under Quantum Transition

Litecoin: Public-Key Exposure Windows Under Quantum Transition is central to Atho Labs' system design because public-key exposure windows are often the decisive variable in post-quantum forgery timelines. Traditional crypto systems generally assume Litecoin inherits most security and migration assumptions from Bitcoin, including classical signature dependence and operational coordination constraints., and that assumption held under classical adversary models.

The technical concern in this area is operators fail to prioritize key-rotation and exposure minimization controls before migration pressure spikes. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Ethereum: Public-Key Exposure Windows Under Quantum Transition

Ethereum: Public-Key Exposure Windows Under Quantum Transition is central to Atho Labs' system design because public-key exposure windows are often the decisive variable in post-quantum forgery timelines. Traditional crypto systems generally assume Ethereum combines account-model complexity, diverse tooling, and deep ecosystem coupling around classical signature assumptions for externally owned accounts., and that assumption held under classical adversary models.

The technical concern in this area is operators fail to prioritize key-rotation and exposure minimization controls before migration pressure spikes. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Lattice Signatures: Implementation Correctness and Determinism

Lattice Signatures: Implementation Correctness and Determinism is central to Atho Labs' system design because lattice assumptions provide a practical balance between performance and post-quantum confidence when implemented with side-channel discipline; additionally, implementation correctness, canonicalization, and deterministic acceptance rules are foundational for distributed trust. Traditional crypto systems generally assume classical signature systems remain optimized for present-day efficiency but face structural quantum fragility, and that assumption held under classical adversary models.

The technical concern in this area is non-deterministic behavior can create divergent outcomes even with identical high-level policies. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

## Hash-Based Signatures: Implementation Correctness and Determinism

Hash-Based Signatures: Implementation Correctness and Determinism is central to Atho Labs' system design because hash-based constructions provide conservative security framing with strong transparency in assumption modeling; additionally,

implementation correctness, canonicalization, and deterministic acceptance rules are foundational for distributed trust. Traditional crypto systems generally assume many production systems underutilize hash-based options because of historical size and workflow constraints, and that assumption held under classical adversary models.

The technical concern in this area is non-deterministic behavior can create divergent outcomes even with identical high-level policies. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

# AI-Assisted Cryptanalysis and Circuit Optimization: Protocol Layer

AI-Assisted Cryptanalysis and Circuit Optimization: Protocol Layer is central to Atho Labs' system design because AI-assisted workflow can accelerate circuit search, resource estimation, and adversarial strategy iteration; in this domain, protocol assumptions should be mapped to explicit invariant checks and deterministic acceptance behavior. Traditional crypto systems generally assume legacy ecosystems often separate strategic planning from implementation constraints, creating blind spots under adversarial pressure, and that assumption held under classical adversary models.

The technical concern in this area is security planning that ignores AI acceleration may understate practical risk timelines. In public blockchain settings, attack feasibility is shaped by key exposure windows, mempool visibility, signature reuse patterns, and operational heterogeneity between nodes.

# Research Context and Standards References (Narrative)

Atho's research posture synthesizes established cryptography literature, NIST post-quantum standardization context, and observed operational lessons from production blockchain networks. The key principle is to avoid single-point reasoning.

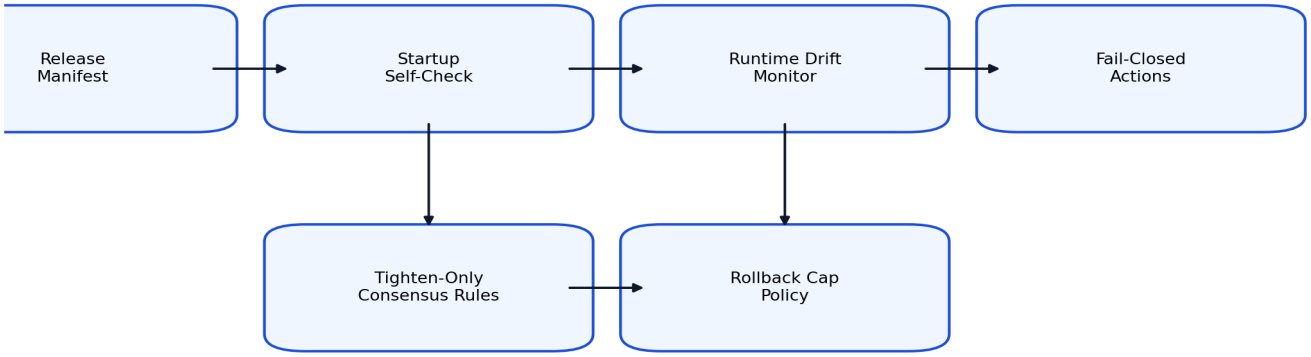
Research-grounded design means translating external findings into explicit controls. For signatures, this means selecting and operationalizing post-quantum-compatible pathways.

## Operational Security and Upgrade Visuals

These diagrams summarize how transactions move from wallet to final state transition [FN-03], and how runtime integrity and update controls are enforced [FN-08][FN-09][FN-15].


Pipeline focus: canonical serialization, strict signature verify, deterministic fee and state transitions.

*Figure 4: Consensus processing path from wallet construction through mempool, miner selection, and block verification.*




Upgrade safety model: verify artifacts, reject unsafe drift, and bound emergency rollback windows.

*Figure 5: Runtime guard and upgrade discipline flow with fail-closed controls and bounded rollback policy [FN-10][FN-26].*

### Part I in Plain Language

- Quantum risk planning is about future safety windows, not today's benchmark speed [FN-01].
- Atho keeps transaction and signature rules deterministic so nodes agree on one truth [FN-02][FN-16].
- Runtime guard and version rules are there to stop unsafe drift during upgrades [FN-08][FN-09].
- The target is simple: clear rules, measurable behavior, and fast fault detection [FN-15].

## Part II - Protocol Constants, Models, and Diagrams

This section bridges narrative and enforcement. It presents active constants, deterministic economic thresholds, and visual system models so readers can verify how policy and implementation connect [FN-14].

### Consensus Constants Snapshot

| Parameter | Current Value |
|---|---|
| Consensus Version | 1.00 |
| Block Time | 120 seconds |
| Difficulty Retarget Interval | 180 blocks |
| MTP Window | 13 blocks |
| Future Timestamp Drift | 30 seconds |
| Tx Confirmations (Regular) | 10 blocks |
| Coinbase Maturity | 150 blocks |
| Blocks Per Era | 1,000,000 |
| Tail Start Height | 8,000,000 |
| Tail Reward | 0.19531250 ATHO/block |
| Pre-Tail Supply | 100,000,000.000 ATHO |
| Hard Max Supply | 150,000,000 ATHO |
| Supply Floor | 21,000,000 ATHO |
| Fee Floor | 350 atoms/byte (0.00000035 ATHO/byte) |
| Minimum Tx Fee | 100,000 atoms (0.000100 ATHO) |
| Dust Threshold | 250 atoms (0.000000250 ATHO) |
| BPoW Enforcement Height | 10,000 |
| Bond Requirement | 25.000 ATHO |

| | |
|---|---|
| Bond Activation Confirmations | 25 blocks |
| Slash Penalty | 2.500 ATHO |
| Bootstrap Allocation (Premine) | 390,625 ATHO @ block 1 |
| Max Block Size | 3,500,000 bytes |
| Max Block Weight | 14,000,000 |
| Annual Tail Issuance | 51,328.125 ATHO/year |
| Max Annual Fees (100% Utilization) | 321,930.000 ATHO/year |
| Max Annual Burn-Path Fees | 144,868.500 ATHO/year |
| Max Annual Miner-Routed Fees | 0.000 ATHO/year |
| Max Annual Consensus-Pool Routing | 177,061.500 ATHO/year |
| Deflation Threshold Utilization | 35.431% |

## Recent Accepted TX Size Samples (Mainnet)

Sizing is shown using vsize and weight so policy, miner selection, and UI reporting stay aligned [FN-04][FN-05].

| Inputs | Outputs | vsize (vB) | Weight (wu) | Fee (ATHO) | Est TPS @120s |
|---|---|---|---|---|---|
| 1 | 2 | 566 | 2,264 | 0.000113200 | 51.53 |
| 2 | 2 | 615 | 2,460 | 0.000123000 | 47.43 |
| 3 | 2 | 664 | 2,656 | 0.000132800 | 43.93 |
| 4 | 2 | 713 | 2,852 | 0.000142600 | 40.91 |

## Protocol Mechanics Illustrations

These diagrams show concrete mechanics that are easy to miss in text-only explanations: verification path, UTXO transformation, fee burn, relay, and persistence flow [FN-17][FN-19][FN-23].

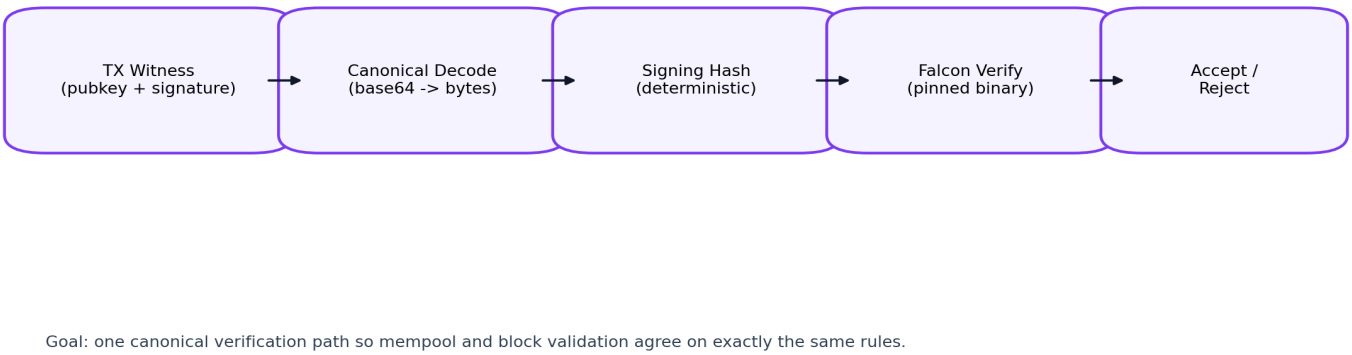
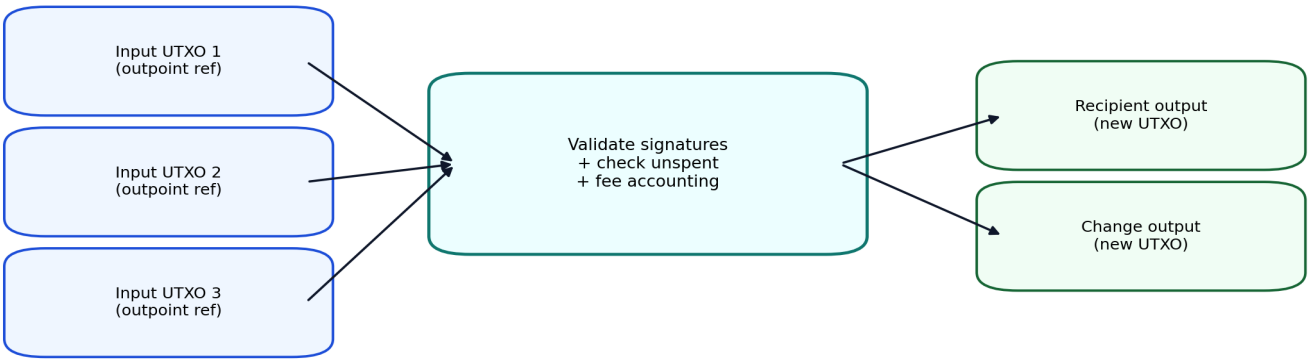
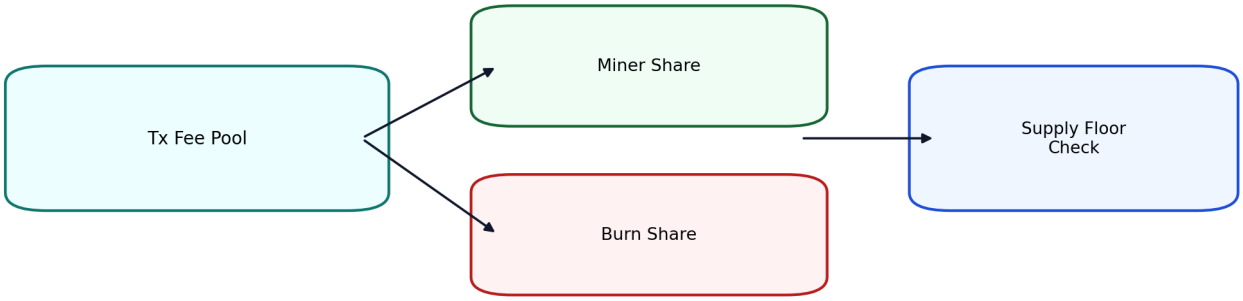



*Figure 6: Canonical signature verification path used by transaction validation [FN-11][FN-16].*



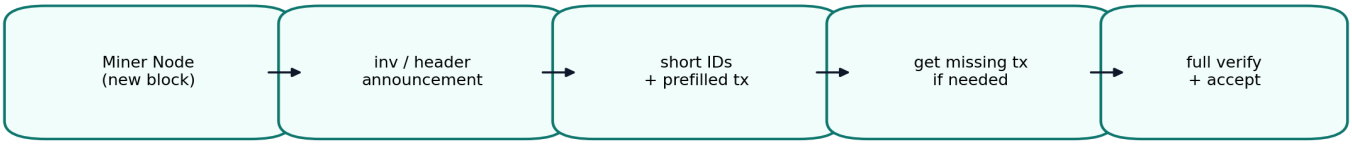

In plain terms: old coins are consumed once, then replaced by new coins.

*Figure 7: UTXO spend lifecycle from referenced inputs to newly created outputs [FN-03][FN-17].*




Burn is applied deterministically, then clipped if needed so supply cannot go below floor.

*Figure 8: Fee handling model showing miner share, burn share, and floor clipping [FN-07][FN-27].*




Compact relay reduces payload but peers still fetch full missing tx data before validation.

*Figure 9: Compact relay sequence for faster propagation with missing-tx fetch behavior [FN-12][FN-19].*

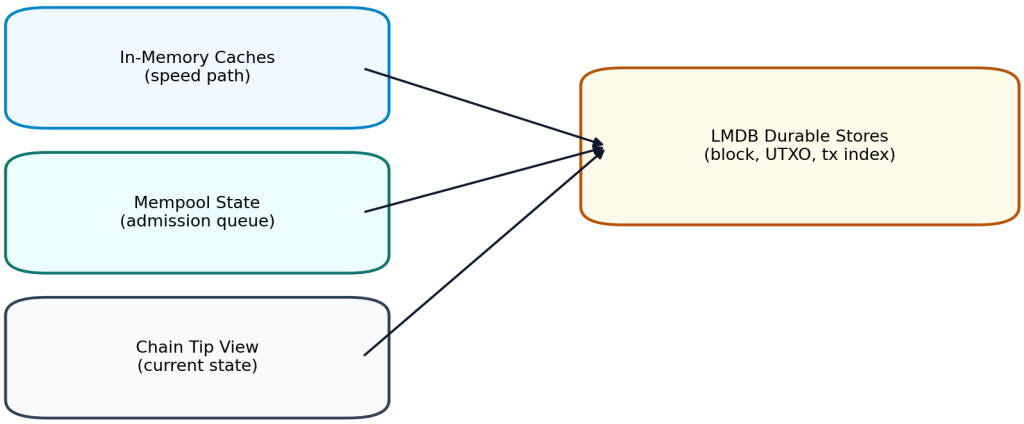



*Figure 10: RAM acceleration layered on top of LMDB durability and restart safety [FN-22].*

## PQC Key Encryption at Rest (Kyber + AES-256)

Atho's wallet lockbox encrypts secret payload data with AES-256-GCM and then protects the DEK with two independent control planes: a password-derived path and a Kyber KEM path. This is the practical full-stack posture: Falcon for signatures, Kyber for post-quantum DEK protection, and AES for efficient authenticated payload encryption.

- Payload encryption: AES-256-GCM over Falcon private key material and mnemonic secret fields.
- Password plane: Argon2id-derived KEK used to unwrap DEK for normal unlock UX.
- PQC plane: Kyber wrap of the same DEK for advanced recovery and post-quantum custody depth.

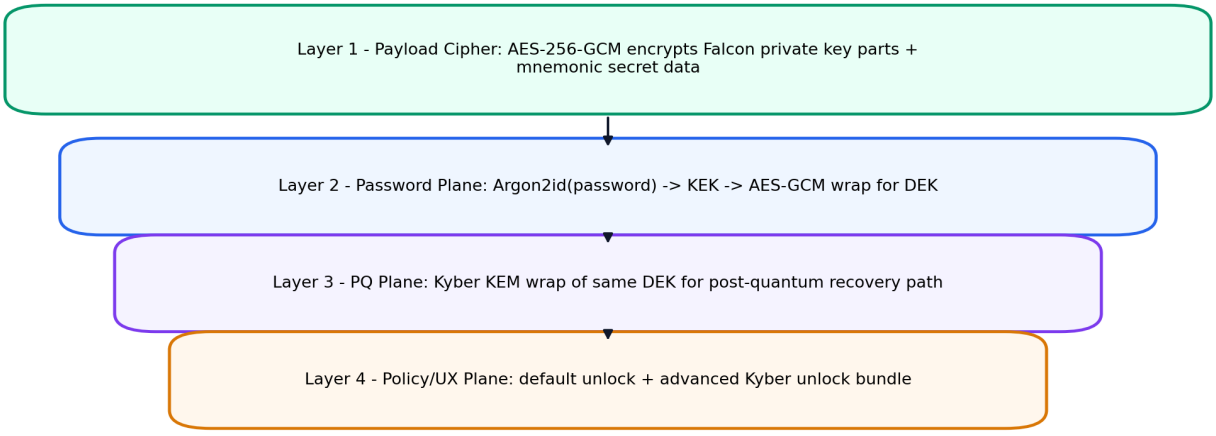
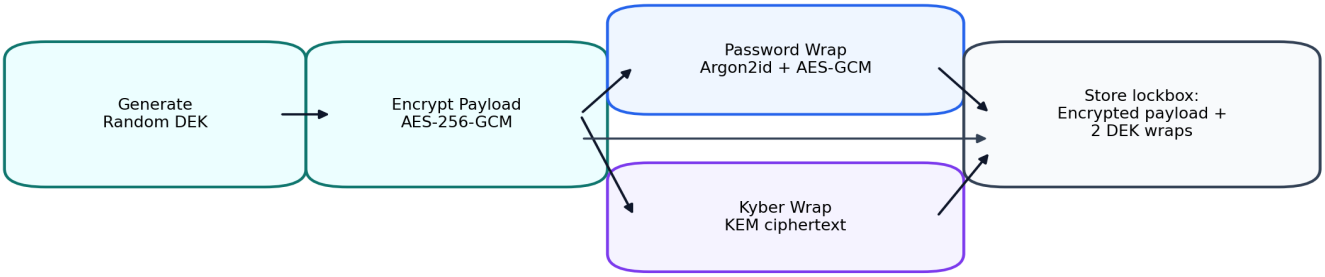
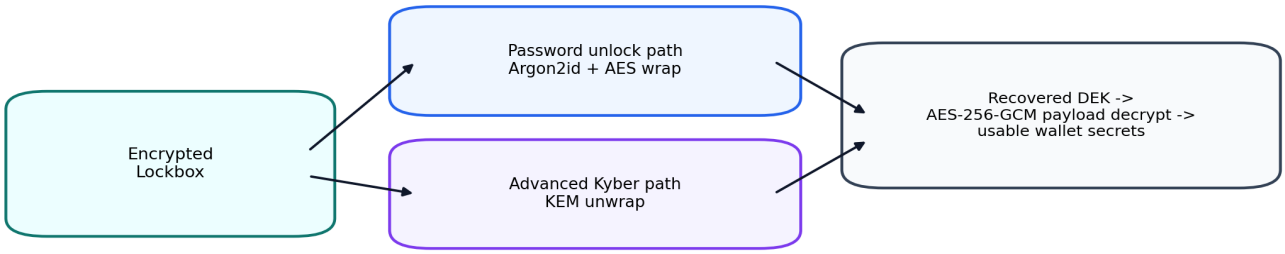



*Figure 10A: Layered Atho key-at-rest stack showing AES payload encryption with password and Kyber DEK protection planes.*




Both wraps reference the same DEK; either authorized path can recover it under policy.

*Figure 10B: DEK lifecycle from generation through dual-wrap storage records (password wrap + Kyber wrap).*




Operational view: one encrypted payload, two authorized key-unlock planes, one deterministic decrypt result.

*Figure 10C: Unlock routing model where authorized password or Kyber path recovers DEK for the same deterministic decrypt output.*

## System Diagrams and Economic Charts

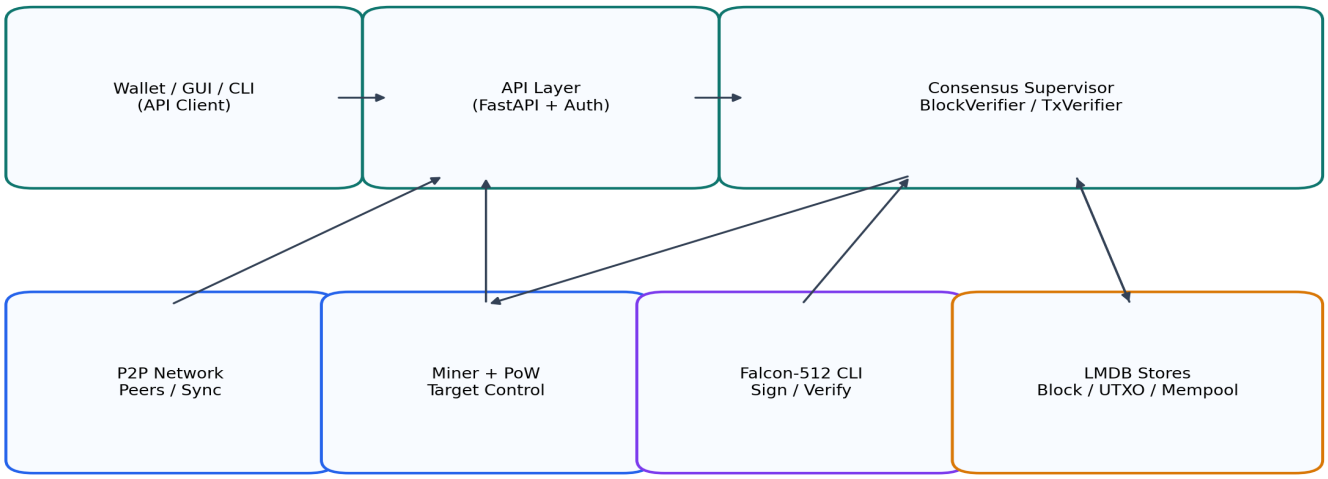



Figure: Atho runtime architecture from client entry points through consensus, cryptography, and storage.

*Figure 11: End-to-end Atho system architecture and data/control paths.*

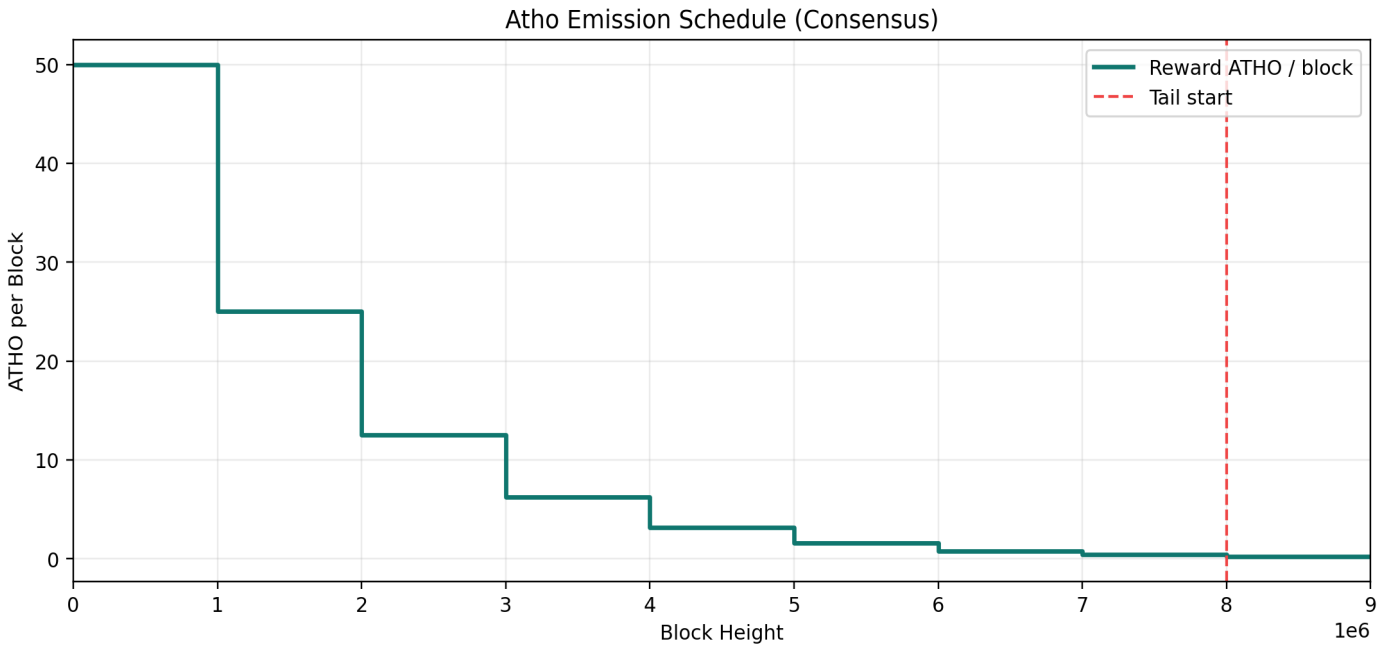



*Figure 12: Consensus reward schedule with pre-tail eras and tail transition [FN-06][FN-25].*

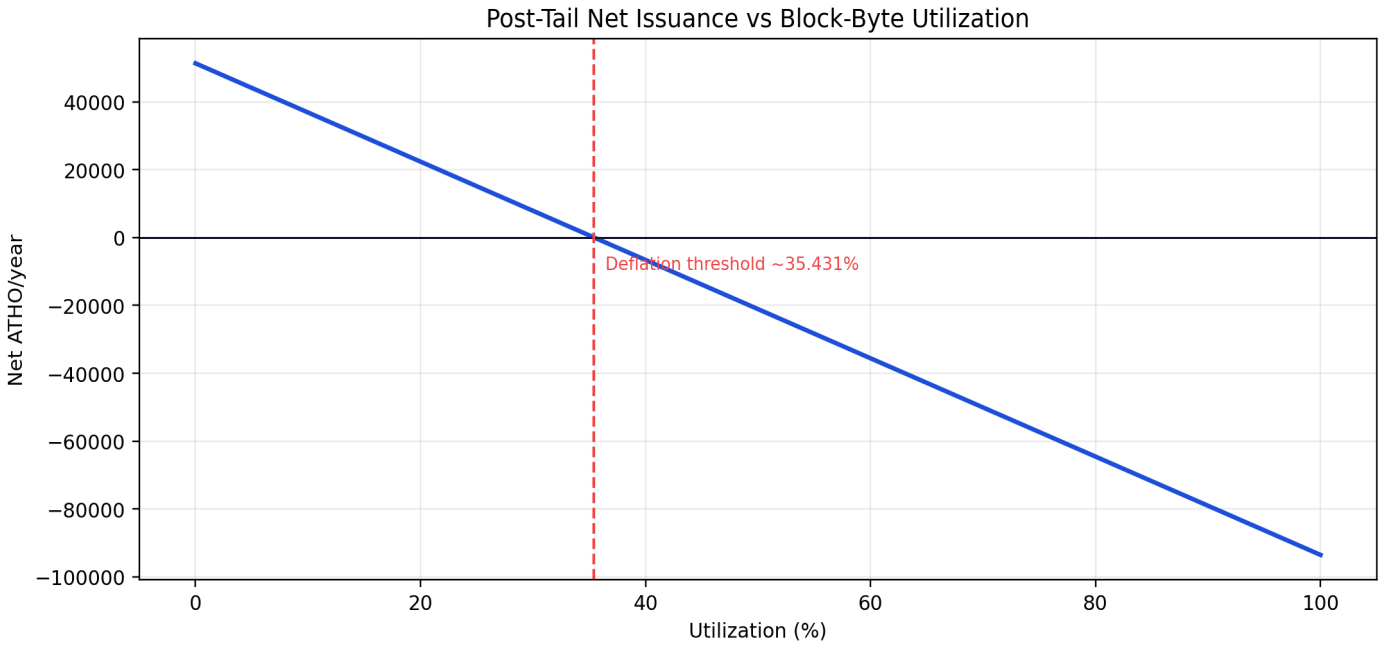



*Figure 13: Net issuance response to block-byte utilization under active fee policy [FN-07].*

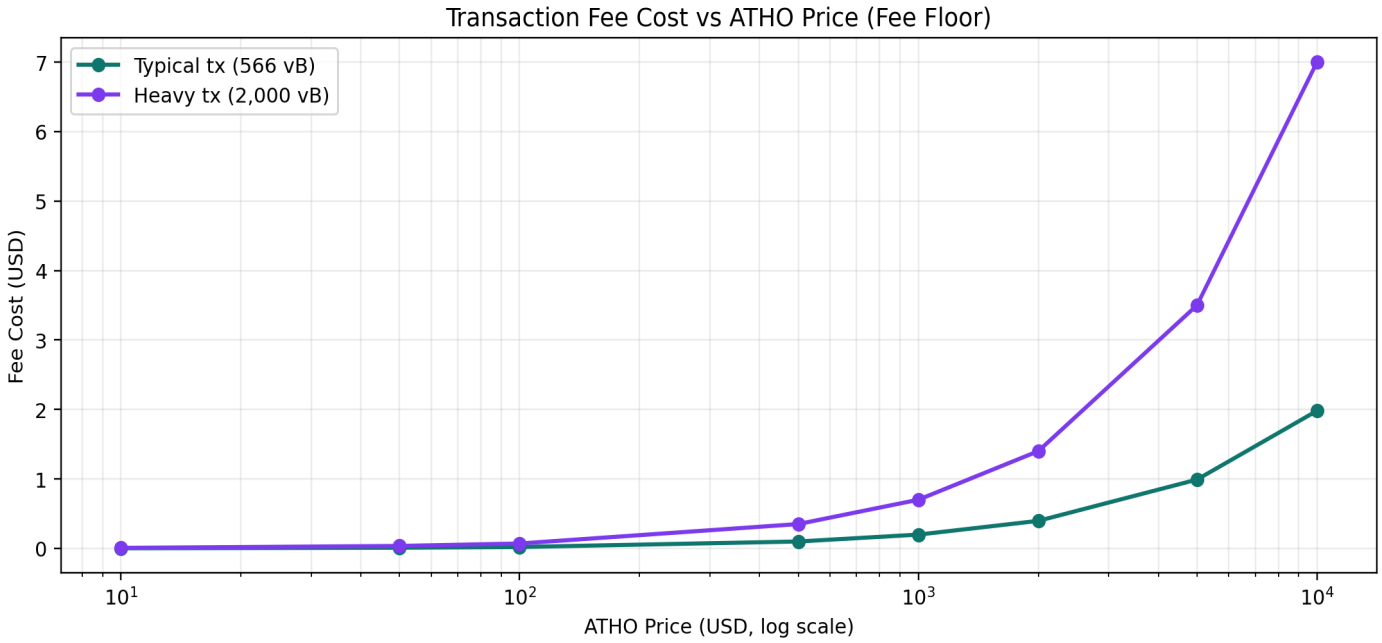



*Figure 14: Fee cost profile for representative transaction sizes across ATHO price levels [FN-24].*

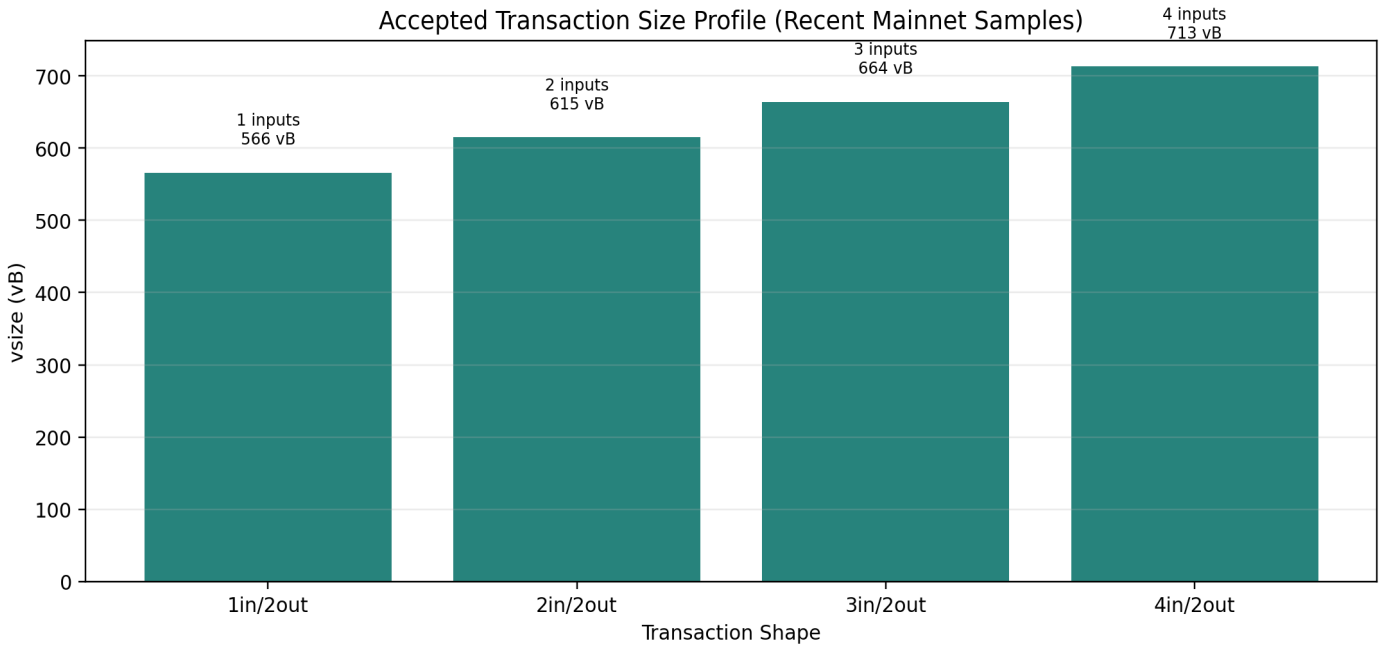



*Figure 15: Accepted transaction vsize profile from recent mainnet samples (optimized wire format era) [FN-04].*

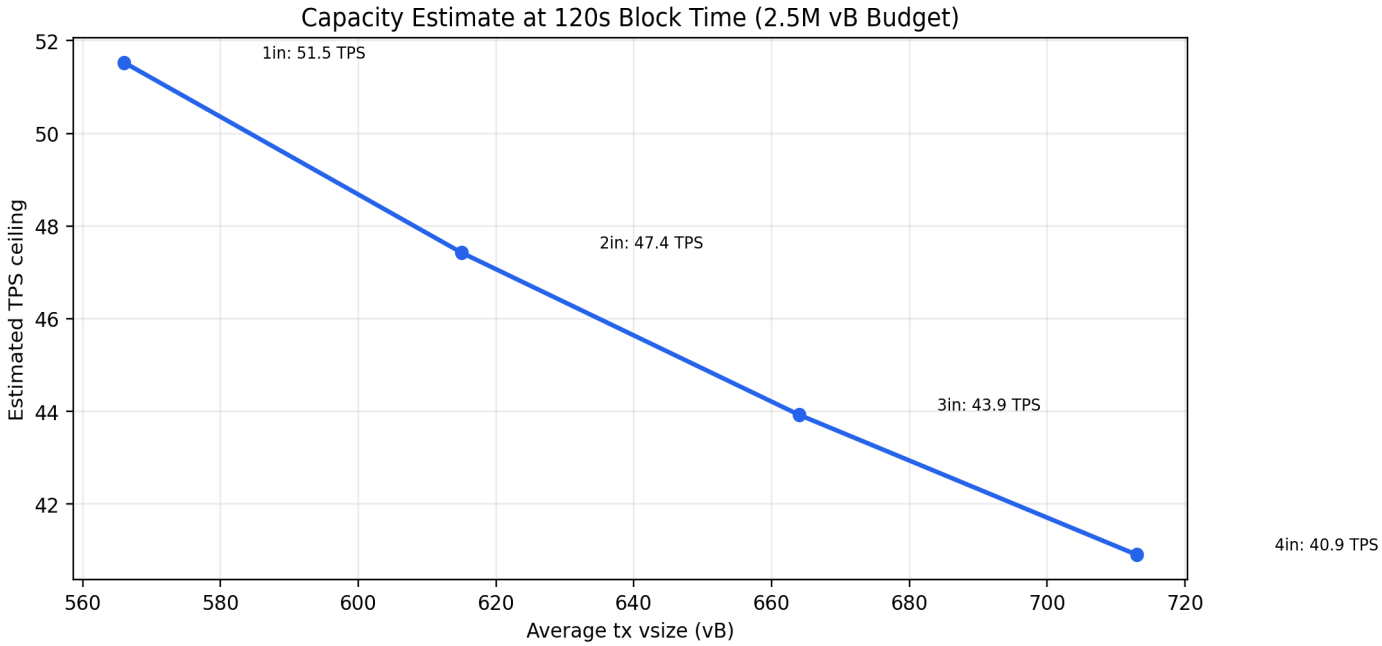



*Figure 16: Estimated TPS envelope at 120-second blocks as a function of average transaction vsize.*

## Emissions Modeling Visual Atlas

This atlas replaces text-heavy modeling dumps with chart-first analysis using generated CSV outputs. Each figure below is derived directly from the emissions modeling datasets and presents scenario behavior in visual form for fast comparison [FN-06][FN-07].

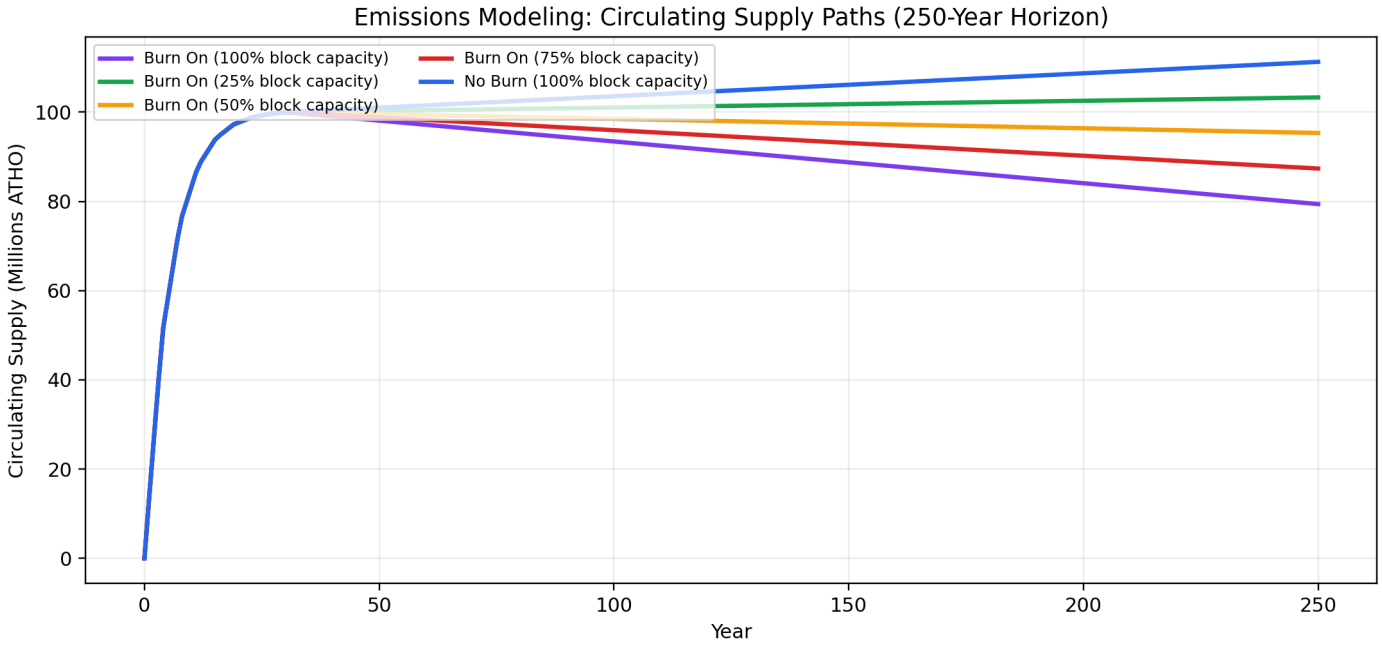



*Figure 17: 250-year circulating supply trajectories by scenario (no burn and burn-at-capacity levels).*

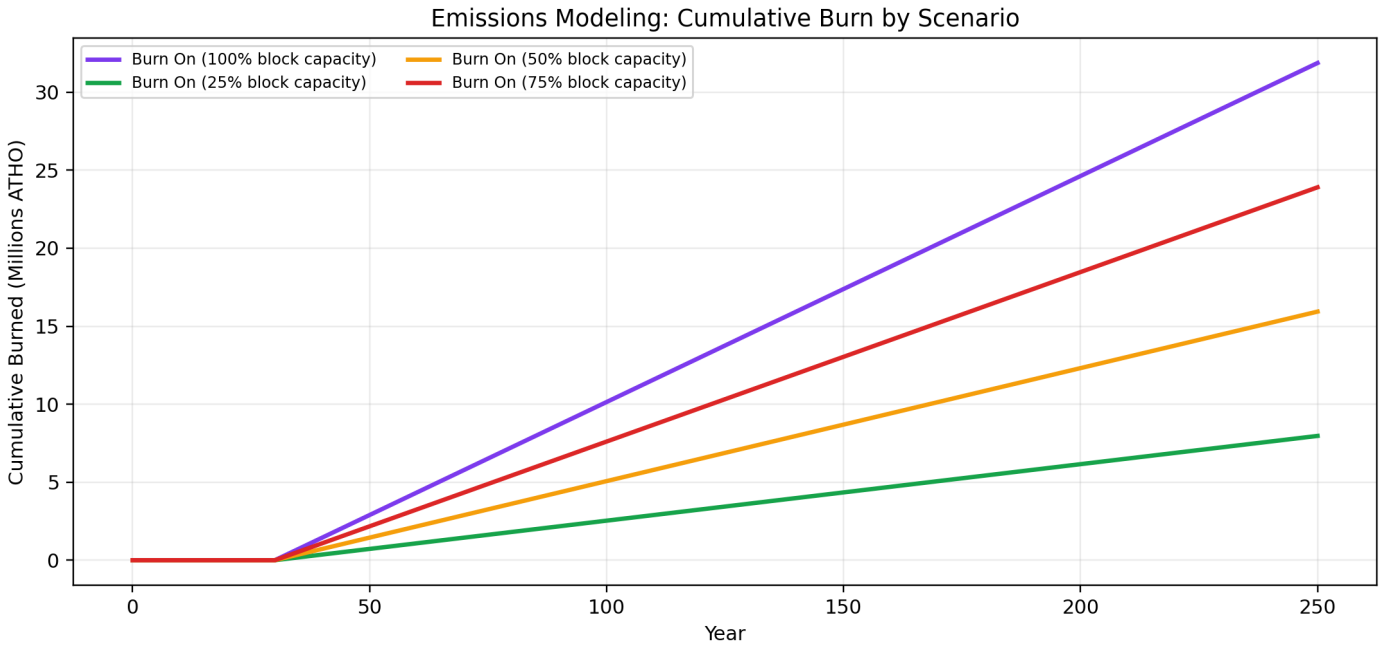



*Figure 18: Cumulative burned ATHO over time for burn-enabled scenarios, showing monotonic divergence by utilization.*

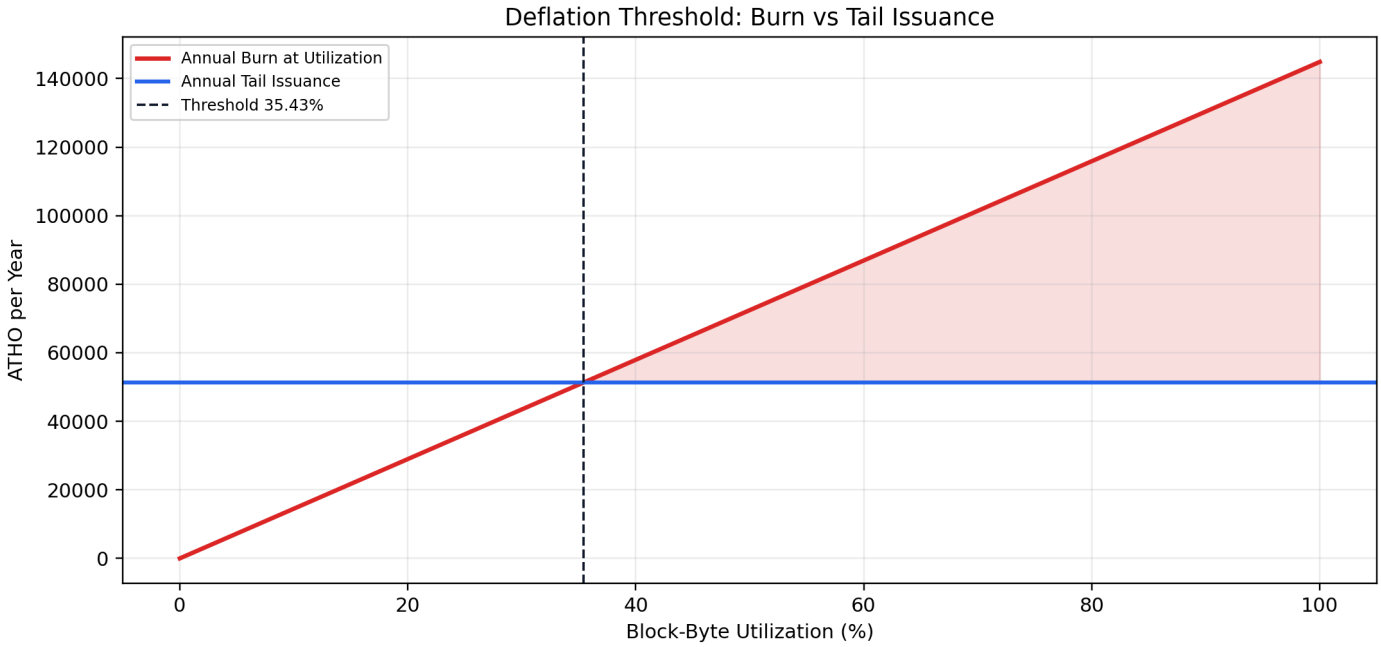



*Figure 19: Deflation threshold as the intersection of annual burn and annual tail issuance.*

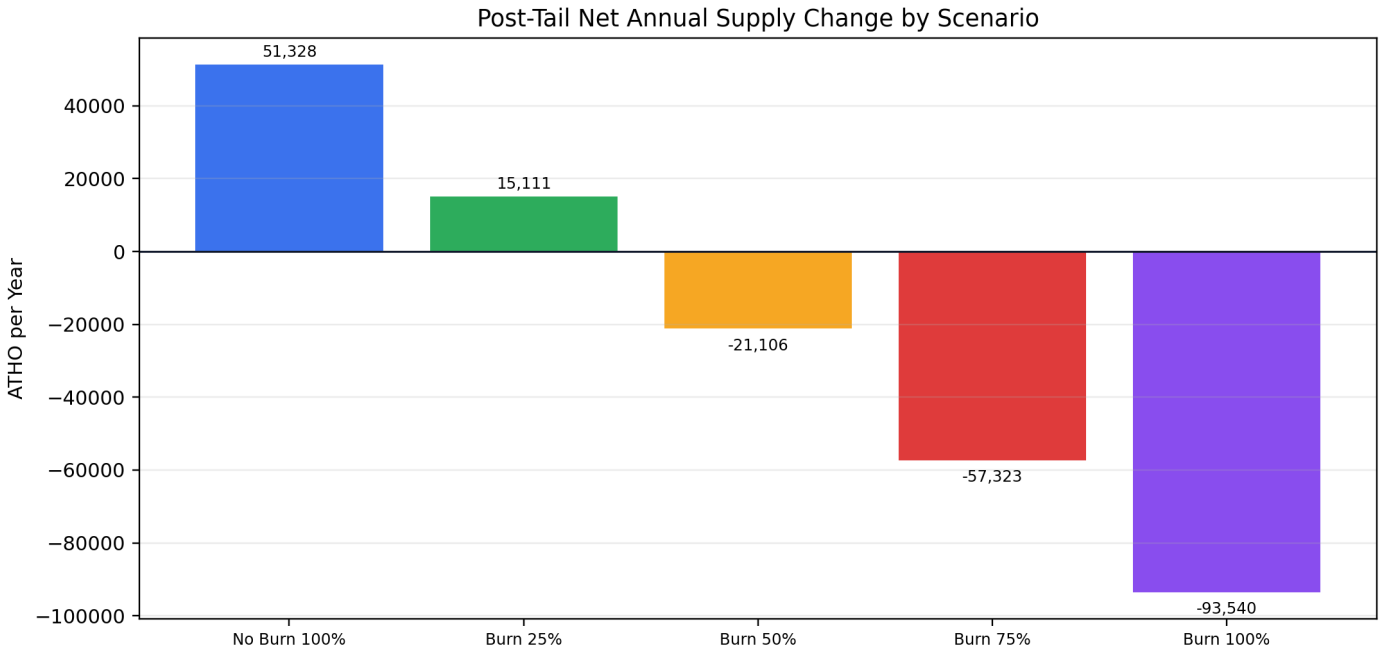



*Figure 20: Post-tail net annual supply change by scenario (positive inflationary vs negative deflationary ranges).*

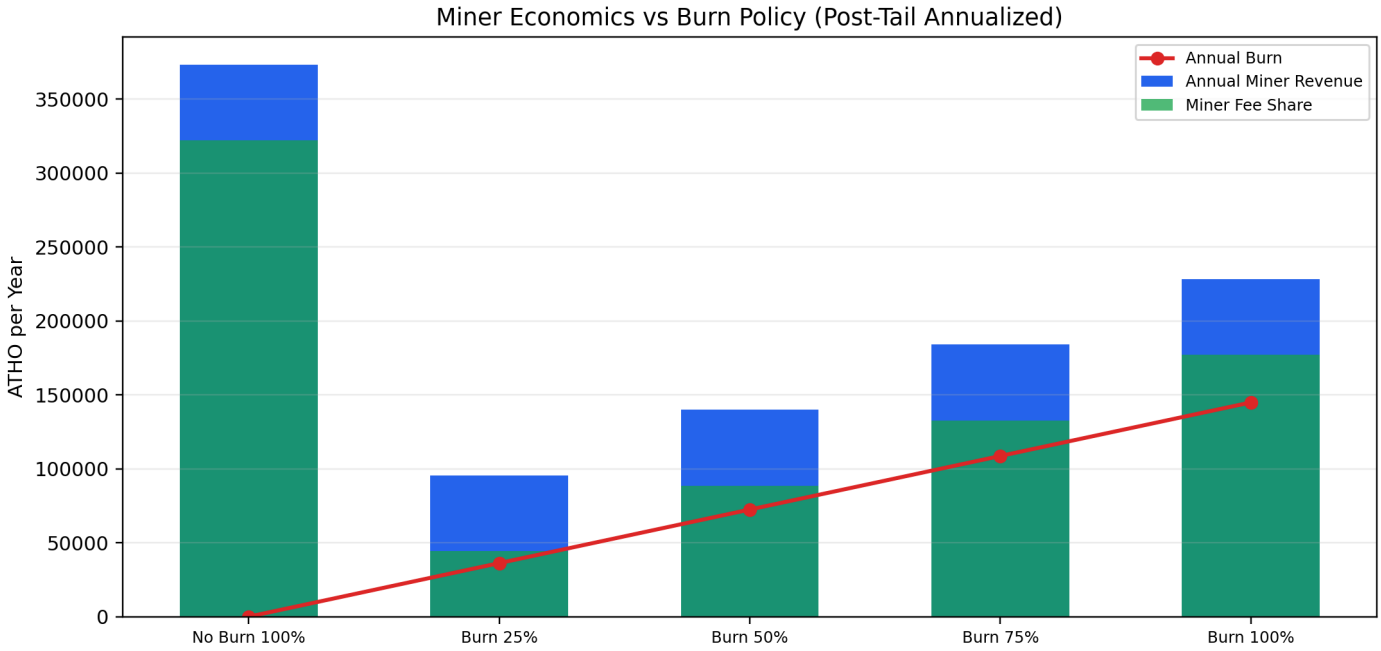



*Figure 21: Miner economics comparison across scenarios, including annual miner totals, miner fee share, and annual burn.*

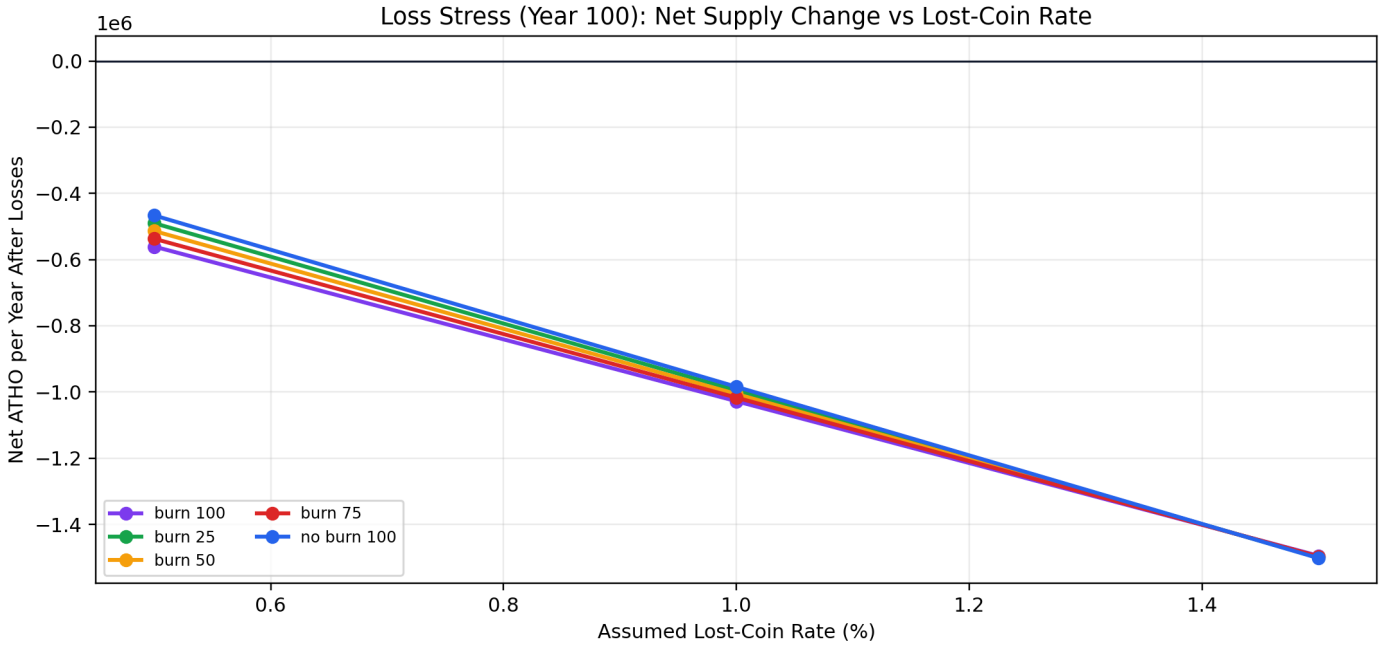



*Figure 22: Lost-coin stress curves at year 100 showing when protocol net issuance is offset by loss assumptions.*

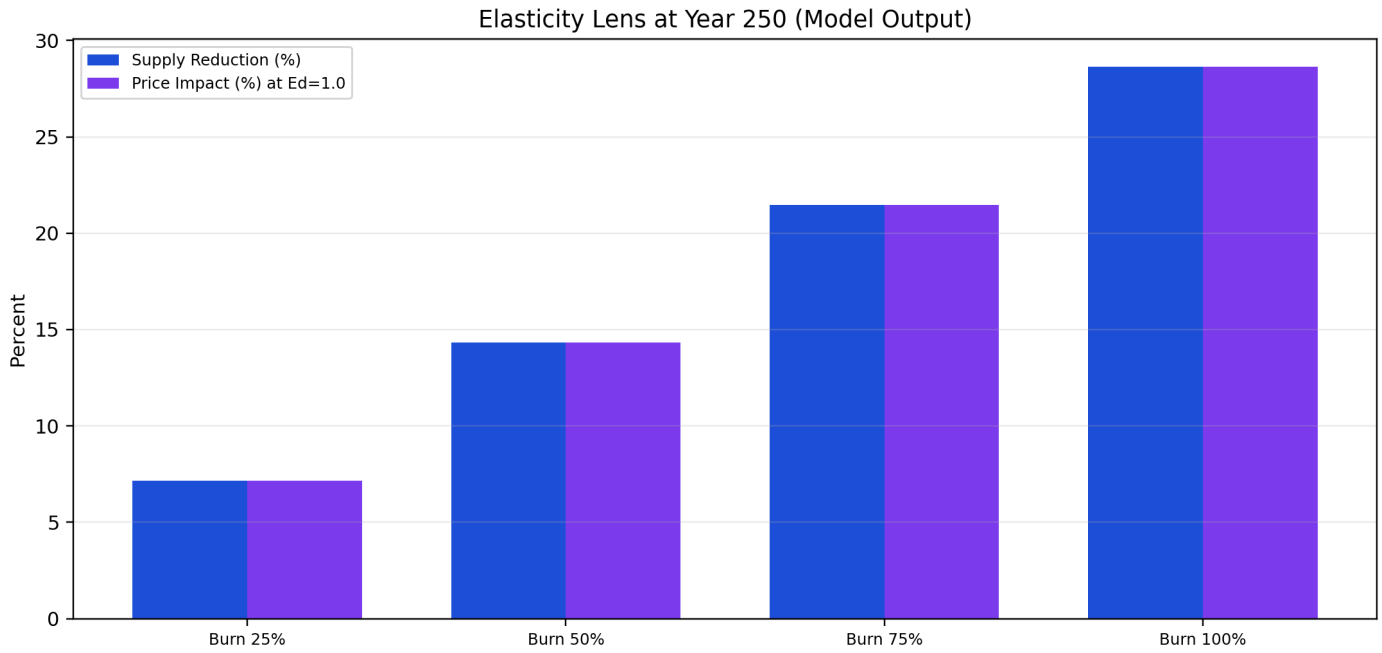



*Figure 23: Year-250 elasticity lens comparing supply reduction with modeled price impact at elasticity 1.0.*

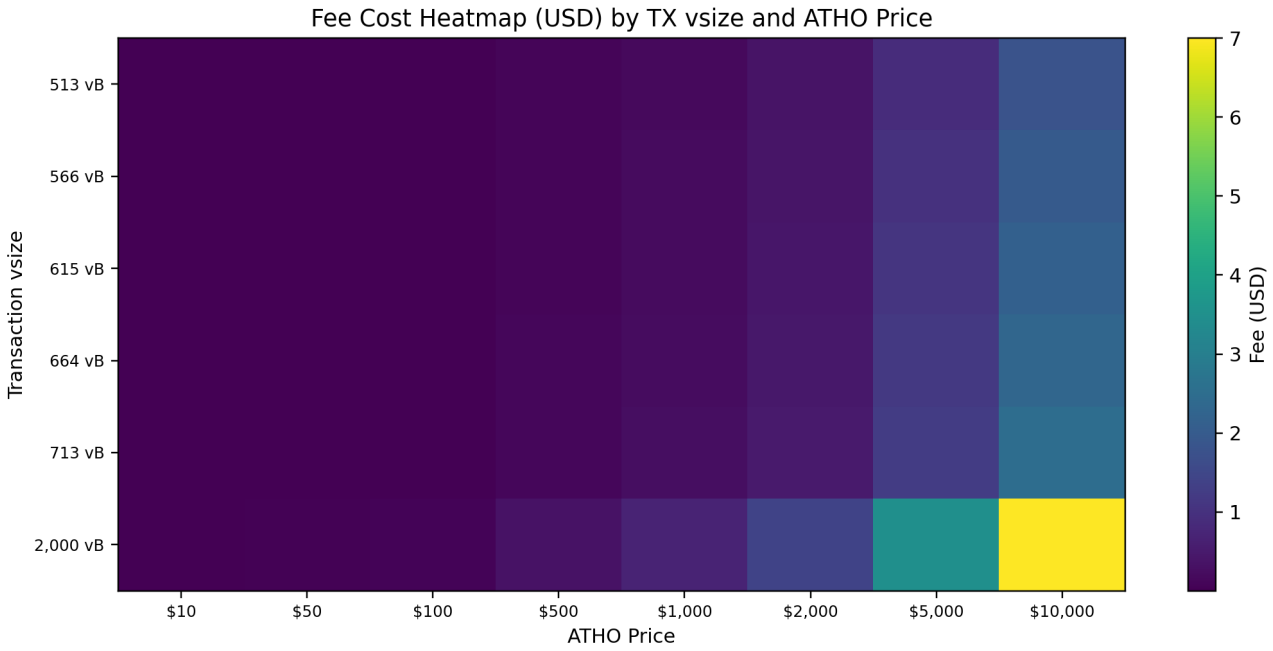



*Figure 24: Fee-cost heatmap by transaction size and ATHO spot price under active byte-fee policy.*

# Plain-Language FAQ for Non-Technical Readers and Partners

This FAQ translates technical protocol language into business and operations language. It is formatted as a numbered Q/A reference so readers can scan quickly and revisit specific concerns.

| # | Question | Answer |
|---|---|---|
| **1** | **What is Atho in one sentence?** | Atho is a blockchain designed to stay understandable, auditable, and secure over long time horizons, including post-quantum risk planning. |
| **2** | **What is the premine/bootstrap allocation and what is it for?** | Atho's bootstrap allocation is 390,625 ATHO at block 1. Its purpose is controlled network bootstrapping (initial distribution and miner/bond onboarding runway) and it remains consensus-accounted and auditable on-chain. |

| # | Question | Answer |
|---|---|---|
| 3 | **What core network parameters should operators memorize first?** | Target block time is 120 seconds, difficulty retarget interval is 180 blocks, regular transaction confirmations are 10, coinbase maturity is 150 blocks, fee floor is 350 atoms/byte, and BPoW bond requirement is 25.000 ATHO with 25 confirmations. |
| 4 | **Why does this whitepaper talk so much about quantum risk now?** | Because migration becomes harder as ecosystems grow. Planning early is cheaper and safer than emergency migration after large value and infrastructure are already locked in. |
| 5 | **Does post-quantum design mean Atho is magically future-proof forever?** | No. It means Atho starts from stronger assumptions and keeps an upgrade path that can tighten security over time. No protocol should claim permanent immunity. |
| 6 | **What does 'consensus-critical' actually mean for a non-engineer?** | It means every honest node must calculate the same result. If two nodes use different rules, they can split into incompatible chains. |
| 7 | **What are vsize and weight in simple terms?** | They are accounting tools for block capacity and fees. They let the network measure transaction footprint consistently, especially when witness data is involved. |
| 8 | **Are fees random or controlled?** | They are policy-controlled. Atho uses explicit constants and deterministic checks so fee minimums, burn behavior, and acceptance logic are predictable and auditable. |
| 9 | **What is fee burn, and why do it?** | Fee burn removes a configured portion of fees from supply. It is used to shape long-run net issuance behavior while still preserving miner incentives. |
| 10 | **Could fee burn accidentally destroy too much supply?** | The design includes supply floor clipping logic. Burn behavior is bounded so supply cannot pass below the configured floor. |
| 11 | **What does 'tail issuance' mean?** | After halving eras, reward becomes fixed. That fixed long-run issuance is called tail issuance and is part of Atho's steady-state economic design. |
| 12 | **How should partners interpret TPS numbers in this paper?** | Treat them as scenario-based capacity estimates tied to block interval, block budget, and transaction mix. They are not universal guarantees under all network conditions. |
| 13 | **Why can high-input transactions still be larger?** | Each input must prove spending authority. Even after wire-format optimization, more inputs generally require more verification data and therefore larger transactions. |
| 14 | **If RAM caching exists, why keep LMDB storage?** | RAM improves speed; LMDB preserves durability and recovery. If a node restarts or crashes, durable state is required for reliable operation. |
| 15 | **What is runtime guard and why should operators care?** | Runtime guard watches critical integrity assumptions (files, policy controls, and upgrade discipline). In enforce mode, it can stop unsafe operation instead of continuing silently. |
| 16 | **What is binary pinning in practical terms?** | It means the node checks the expected digest of a critical binary before trusting it. This reduces supply-chain and tampering risk. |
| 17 | **Why does Atho emphasize 'tighten-only' updates?** | To prevent accidental or hidden weakening of core safety policy. Future changes should increase rigor, not quietly relax critical constraints. |
| 18 | **What does rollback cap mean during a crisis?** | It defines a hard maximum historical rewind window. This helps incident response stay bounded and predictable rather than open-ended. |
| 19 | **Can a transaction be accepted by mempool but later rejected in a block?** | It should be rare. The system aims for one canonical validation path so mempool and block verification agree. Mismatches are treated as bugs to eliminate. |
| 20 | **What does compact relay improve?** | Network efficiency. Peers can exchange smaller block hints first and fetch only missing transaction data, reducing bandwidth and propagation time. |
| 21 | **How should non-technical partners read this whitepaper?** | Read the executive sections, diagrams, FAQ, and constants tables first. Then use the appendix only for deeper technical validation. |
| 22 | **What is the most important trust principle in this design?** | Do not rely on claims alone. Rely on deterministic rules, observable logs, reproducible artifacts, and verifiable enforcement. |
| 23 | **How does Atho differ from a typical 'fast chain' pitch?** | Atho prioritizes verifiability and policy determinism first, then performance optimization. Speed claims are tied to measurable assumptions, not marketing-only numbers. |

| # | Question | Answer |
|---|---|---|
| 24 | **Is Atho trying to replace all existing chains?** | No. The design focus is to provide a clear, security-forward architecture and operational discipline for long-horizon reliability. |
| 25 | **Why is deterministic behavior repeated so often in this paper?** | Because deterministic behavior is what keeps independent nodes in agreement. If behavior is ambiguous, consensus safety degrades. |
| 26 | **Can transactions be reversed if sent to the wrong address?** | No protocol-level undo exists for normal transfers. Safety depends on careful address validation before sending. |
| 27 | **What does 'confirmed' mean versus 'pending'?** | Pending means accepted by policy and waiting for block inclusion. Confirmed means included in an accepted chain block. |
| 28 | **Why can pending status sometimes last longer than expected?** | Block timing, fee competitiveness, input complexity, and mempool conditions all affect inclusion speed. |
| 29 | **Can one bad transaction block miner progress?** | If policy mismatch exists, it can cause repeated candidate rejection loops. Atho addresses this by enforcing one canonical validation path. |
| 30 | **How does Atho handle large multi-input transactions safely?** | Each input must be independently validated as unspent and authorized. Optimization reduces redundancy, but security checks remain per input. |
| 31 | **Does making transactions smaller reduce security?** | Not if reductions remove redundant serialization only. Security-critical data and verification checks remain mandatory. |
| 32 | **Does compact relay mean nodes trust partial data?** | No. Compact relay improves transport efficiency, but full required data is fetched before final validation. |
| 33 | **Can a peer lie about telemetry like hashrate?** | Telemetry can be spoofed unless attested. Atho treats telemetry as operational signal, not consensus truth. |
| 34 | **What if a node runs outdated software?** | Compatibility depends on consensus version and active policy. Outdated nodes risk rejection or forked behavior if rules changed. |
| 35 | **How do upgrades avoid weakening security?** | The tighten-only approach prevents silent safety relaxation. Upgrades should add checks or clarity, not remove core protections. |
| 36 | **What is the practical purpose of signed release manifests?** | They let operators verify that release artifacts match approved metadata before trusting them in production. |
| 37 | **What happens if runtime guard detects critical drift?** | In enforce mode, the node can stop mining, reject risky processing, and emit explicit security events instead of continuing blindly. |
| 38 | **Why are rollback limits important in incidents?** | They bound recovery actions so crisis response stays controlled, auditable, and resistant to ad-hoc overreach. |
| 39 | **Can rollback policy be unlimited for flexibility?** | Unlimited rollback increases abuse and uncertainty risk. Bounded rollback is safer for operator coordination and user trust. |
| 40 | **Does Atho rely on floating-point math for money?** | Consensus accounting is integer-based. This avoids precision drift and keeps monetary checks reproducible across nodes. |
| 41 | **What does the supply floor protect against?** | It prevents excessive deflation from driving spendable supply below a configured lower bound. |
| 42 | **Can fee policy change over time?** | Yes, through explicit governance and versioned updates. Changes should remain auditable and consistent across nodes. |
| 43 | **Why does the paper include both technical and plain-language sections?** | Because adoption requires both engineering correctness and stakeholder comprehension. One without the other creates execution risk. |
| 44 | **How should exchanges evaluate readiness?** | Check deterministic validation behavior, upgrade discipline, rollback policy, monitoring coverage, and incident-response clarity. |
| 45 | **How should institutional partners evaluate readiness?** | Use an evidence checklist: cryptographic posture, consensus invariants, artifact provenance, operational controls, and reproducible testing. |

| # | Question | Answer |
|---|---|---|
| **46** | **Is Atho private by default?** | It is a transparent ledger model. Privacy expectations should match that transparency unless additional privacy layers are explicitly deployed. |
| **47** | **What data should operators monitor daily?** | Chain tip progress, mempool pressure, rejected block reasons, validation error rates, storage health, and security-log anomalies. |
| **48** | **Why are broad exception catches considered risky?** | They can hide root causes and delay incident diagnosis. Narrow exceptions plus structured logs improve reliability and security response. |

# Part III - Implementation and Code Appendix

The following appendix intentionally includes only core implementation surfaces instead of dumping the entire repository. Its purpose is to map the critical consensus, cryptography, storage, and modeling modules that enforce policy and network safety [FN-11][FN-15].

## Code Audit Section - How to Read It

Read this section as a practical operator checklist: what can break consensus or liveness, what user impact appears first, and what deterministic fix pattern closes it. The objective is clarity over volume [FN-20][FN-28].

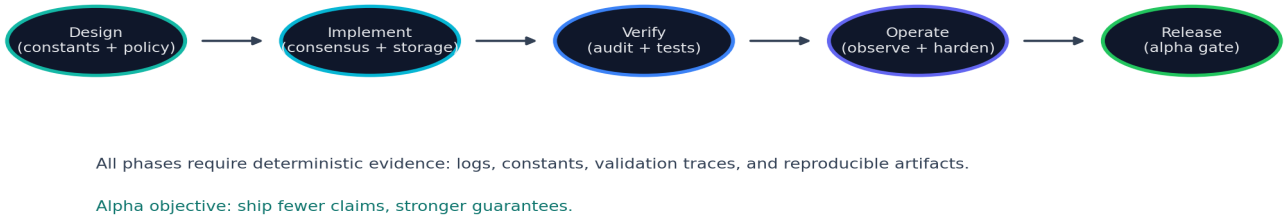



*Figure 25: Alpha delivery flow from design to enforceable release controls.*


*Figure 26: Code-audit remediation lifecycle used for performance and security hardening.*

## Code Audit Findings Summary (Organized)

| Priority | Finding Theme | Primary Effect | Remediation Pattern |
|---|---|---|---|
| Critical | Consensus verify drift | Fork risk, conflicting acceptance | Single canonical verify path for mempool + block [FN-16] |
| High | Non-atomic persistence paths | State mismatch after interruptions | Transactional commit boundaries + rollback tests [FN-23] |
| High | Slow miner template assembly | Stale blocks, weak liveness | Indexed lookups + incremental candidate accounting [FN-21] |
| Medium | API scan-heavy endpoints | User-facing lag and timeout spikes | Pagination defaults + summary caches + strict caps |
| Operational | Low-signal logging patterns | Slow incident triage | Structured security logs with deterministic event tags |

## Core Implementation Map

| Module | Core Responsibility |
|---|---|
| Src/Utility/const.py | Consensus constants, reward schedule, fee floor, burn/floor invariants, and network policy toggles. |
| Src/Main/blockveri.py | Block-level validity checks: structure, linkage, witness commitment, payout equations, and fee/burn accou |
| Src/Main/txveri.py | Transaction-level checks: canonicalization, signature verification, fee policy, output validity, and anti-bypas |
| Src/Storage/blockstorage.py | Persistent acceptance path for validated blocks with post-validation guardrails and invariant checks. |
| Src/Storage/utxostorage.py | UTXO set state transitions, spend/create rules, and deterministic integer accounting. |
| Src/Storage/mempool.py | Mempool policy and transaction admission controls prior to block inclusion. |
| Src/Accounts/falcon_cli.py | Falcon CLI execution wrapper, digest pinning verification, and startup validation behavior. |
| Src/Main/esim.py | Emissions modeling engine that produces long-horizon scenario outputs and comparative economics data |
| Src/Main/runnode.py / Src/Main/stop.py | Operational orchestration for node lifecycle management, process visibility, and controlled shutdown. |

## Core Security Invariants (Implementation-Backed)

These invariants are consensus-critical checkpoints; violating them must trigger rejection rather than warning-level behavior [FN-02][FN-15].

- Coinbase payout arithmetic must match consensus reward equations and any active fee-share policy.
- Post-tail fee burn and supply floor clipping are enforced through deterministic integer checks, not optional policy hooks.
- Signature verification pathways are explicit and auditable; startup pinning controls bind cryptographic binaries to expected digests where required.
- UTXO transitions are deterministic and replay-safe under canonical transaction serialization and strict validation gates.
- Emissions and burn tracking must remain reorg-consistent and cross-checkable against block-level accounting fields.

# Documentation Coverage Matrix (Condensed)

To keep this complete whitepaper concise, source documentation is covered through a matrix and targeted highlights rather than full inline duplication. This preserves complete topical coverage while reducing document size and stale-content drift [FN-14][FN-28].

| Document | Words | Coverage Summary |
|---|---|---|
| README.md | 1,127 | Repository structure, component map, and operator-facing entry points. |
| Docs/INDEX.md | 654 | Documentation index and authoritative navigation. |
| Docs/ONBOARDING.md | 726 | Developer Onboarding: protocol/operations reference. |
| Docs/quickstart.md | 698 | Operational startup, run commands, and baseline deployment steps. |
| Docs/WhitePaper.md | 2,733 | Technical overview and protocol summary. |
| Docs/Consensus.md | 1,043 | Block/tx acceptance, policy gates, and reorg behavior. |
| Docs/Falcon512.md | 587 | Falcon integration details and verification posture. |
| Docs/Sha3-384.md | 694 | Hashing rationale, canonicalization, and checksum rules. |
| Docs/Tx.md | 820 | Transaction model, fee policy, and wire/validation behavior. |
| Docs/Sigwit.md | 633 | Witness handling, weight/vsize accounting, and policy integration. |
| Docs/ApiAuth.md | 938 | API key scopes, auth controls, and security layering. |
| Docs/LMDB.md | 594 | Persistence model, map sizing, and recovery/operational constraints. |
| Docs/Base56.md | 746 | Address encoding, checksum behavior, and normalization rules. |
| Docs/KeyManager.md | 1,408 | Key lifecycle and signing/verification management. |
| Docs/Emissions.md | 892 | Issuance model, burn/floor policy, and emission audits. |
| Docs/Inflation_Deflationary.md | 756 | Long-horizon utilization and supply scenarios. |
| Docs/Threat.md | 2,200 | Security threat model and mitigation priorities. |
| Docs/Docker.md | 1,058 | Containerized deployment guidance and operational setup. |
| Docs/dockertest.md | 930 | Docker test harness and verification routines. |
| Docs/Binaries.md | 553 | Binary build/release provenance and constraints. |
| Docs/Troubleshooting.md | 2,000 | Failure-mode diagnostics and operator playbooks. |
| Docs/Node_Stop.md | 681 | Node process lifecycle control utility. |
| Docs/Emissions Modeling/Entire_Overview.md | 2,821 | Model methodology and scenario outputs. |

## Condensed Source Extracts (All Major Docs)

This appendix retains all major documentation topics with bounded excerpts per file. It is intentionally compressed to reduce repetition while preserving broad technical coverage [FN-28].

### Repository Overview

Source: README.md

### Atho Network

Date: 2026-04-05

Website: Atho.io

Atho is a development-stage post-quantum UTXO blockchain focused on deterministic execution, strong accounting guarantees, and practical operator tooling.

The project combines:

- Falcon-512 signature security for post-quantum transaction and block signing paths.
- SHA3-384 hashing across identity and consensus-critical digest paths.
- strict integer-atom accounting in consensus (no float math in value rules).
- BPoW + wallet staking state machines with deterministic validation logic.
- Platinum Shield private transaction layer with block-level STARK proof gating.
- Python orchestration with native C/C++ acceleration in performance-critical paths.

### Why Atho

Atho is built for teams that want:

- clear consensus behavior that is easy to audit,
- a chain architecture that can scale via native hot paths without abandoning Python ergonomics,
- a modern cryptographic direction with practical runtime controls (binary pinning, deterministic codecs, strict policy checks).

Platinum Shield (Private Layer)

Platinum Shield is Atho's private-transaction layer and is designed to coexist with the public UTXO path. It combines private note commitments/nullifiers with block-level STARK proof gating and fail-closed validation behavior.

Reference:

- Platinum Shield Privacy Architecture

Start Here

If this is your first run, go straight to:

- Quickstart

That guide covers environment setup, required binary builds, runtime flags, and first node launch.

Atho Network Parameters Overview

This section summarizes the current consensus/economic profile in one place for operators, integrators, and reviewers.

Consensus and Blockchain Parameters

| Parameter | Current Value | | --- | --- | | Consensus version | 1.00 | | Target block time | 120s | | Difficulty retarget interval | 180 blocks | | Median-time-past window | 13 blocks | | Max future timestamp drift | 30s | | Standard transaction confirmations | 10 blocks | | Coinbase maturity | 150 blocks | | Max block size (base bytes) | 3,500,000 bytes | | Max block weight | 14,000,000 wu | | Max transaction size | 250,000 bytes |

Transaction Sizing and TPS Envelope

These are measured tx-size values under the active binary/witness path (serializetxv1 + Constants.txpolicymetrics). Real throughput depends on tx mix and mempool conditions.

| Tx Shape | Typical vsize | Est. TPS @ 120s blocks | | --- | --- | --- | | 1 in / 1 out (public) | 500 vB | ~58.33 TPS | | 1 in / 2 out (public) | 553 vB | ~52.74 TPS | | 2 in / 2 out (public) | 602 vB | ~48.45 TPS | | 3 in / 2 out (public) | 651 vB | ~44.80 TPS | | public -> private (1 in / 1 private out) | 2471 vB | ~11.80 TPS | | private -> public (1 private in / 1 public out) | 2708 vB | ~10.77 TPS.

Current measured private-flow multiplier versus public 1 in / 2 out (553 vB) is approximately:

- public -> private: 4.47x
- private -> public: 4.90x
- private -> private: 8.45x

### Docs Index

Source: Docs/INDEX.md

### Atho Documentation Index

Date: 2026-04-10

This is the canonical documentation map for the current Atho repository state. Use this file as the first checkpoint before reading topic-specific docs, because it reflects the active production policy profile and the most recent rewrite pass.

### Documentation Goals

The docs in this folder are organized around four outcomes:

- explain consensus and economic policy in plain language,
- map policy to concrete code paths and runtime controls,
- provide operational runbooks for node/wallet/miner workflows,
- preserve a traceable relationship between whitepaper claims, constants, and generated model artifacts.

Current baseline profile referenced across docs:

- target block time 120s,
- difficulty retarget interval 180 blocks,
- tx confirmations 10, private tx confirmations 10, coinbase maturity 150,
- fee floor 350 atoms/vB, minimum fee 100,000 atoms,

- pre-tail base target 100,000,000 ATHO, hard max 150,000,000 ATHO, supply floor 21,000,000 ATHO,
- BPoW bond requirement 25 ATHO with activation at 25 confirmations,
- fee pool routing pre-tail/post-tail 40% / 55%.

Start Here

If you are new to the repository, read in this order:

- quickstart.md: environment setup, native binaries, first node launch.
- Binaries.md: binary build outputs, pin metadata, strict loading controls.
- Troubleshooting.md: common failures and known fixes.
- Consensus.md: block validity, policy constants, deterministic checks.
- Tx.md: tx encoding, fee rules, witness-size behavior.

This sequence minimizes confusion by placing runtime prerequisites before protocol detail.

Core Protocol References

Primary technical references:

- WhitePaper.md
- Consensus.md
- Tx.md
- TransactionFlowDoc.md
- PrivateKeyDerivationandRestore.md
- WalletHierarchyVisual_Guide.md
- PlatinumShield_Privacy.md
- Sigwit.md
- Sha3-384.md
- Falcon512.md
- Emissions.md
- Inflation_Deflationary.md
- AthoFinalSpecs.md
- Emissions Modeling/Entire_Overview.md

For wallet-stake and bonded mining policy, use this focused path:

- WhitePaper.md (economic and governance context)
- Consensus.md (validation and payout mechanics)
- Tx.md (role-aware transaction behavior)

Runtime and Operations

Operational docs for day-to-day deployment and support:

- Network_Stack.md
- ApiAuth.md
- LMDB.md
- KeyManager.md
- Base56.md
- GPU_Miner.md
- Docker.md
- dockertest.md
- Packaging.md
- Node_Stop.md
- gui.md

Use these for system administration, API integrations, mining operations, packaging, and UI runtime control.

Developer References

Codebase orientation and audit support:

- ONBOARDING.md

- SRCFILEMAP.md
- Threat.md
- TransactionSecurityAudit_FAQ.md

SRCFILEMAP.md is especially useful when tracing a rule from docs to implementation modules.

Whitepaper and Modeling Artifacts

Canonical investor and technical narrative sources:

- WhitePaper.md (technical markdown source)
- AthoWhitepaperv3.pdf (new technical v3 PDF artifact)
- AthoWhitepaperCondensed_v3.pdf (condensed high-level investor/casual overview)

Modeling outputs are regenerated from live constants through Src/Main/esim.py and published under Docs/Emissions Modeling/ as both CSV and PDF.

Maintenance Rules for This Folder

### Developer Onboarding

Source: Docs/ONBOARDING.md

### Atho Network — Developer Onboarding

Last refresh: 2026-04-04.

This doc is a practical, minimal path to get a new developer productive fast.

All detailed docs live in Docs/. Use the documentation index in this folder as the central map, and keep this file for the developer path.

### 0) Prerequisites

- Python 3.9+
- Docker Desktop (optional, for multi-node testing)
- macOS/Linux/WSL recommended

### 1) Repo layout (high■level)

- Src/ — core code (nodes, blockchain, storage, P2P, miner, API, CLI)
- Docs/ — protocol and component docs
- docker/docker-compose.yml — Docker services (fullnode/miner/etc.)
- Src/Main/runnode.py — local launcher for full/miner/wallet nodes
- logs/ — runtime logs (ignored in onboarding)

### 2) Local setup (venv)

### 3) Local run (single machine)

Launcher:

Pick: 1 = Full node, 2 = Miner, 3 = Wallet/API, 4 = All.

### 4) Docker run (multi■node test)

Start full node in one terminal:

Start miner in another terminal:

### 5) Logs to watch

Key signals:

- handshake_ok — peers connected
- syncverifiedblocks — blocks downloaded and verified
- blockparentmissing / ORPHAN — parent not yet available
- Mining tip height=... — miner following chain tip

6) Common env knobs

Set per shell (or docker-compose override):

- ATHOP2PPORT — P2P TCP port
- ATHODBROOT — storage root
- ATHOLOGROOT — logs root

- ATHOMINERSYNC_WAIT — delay before mining
- ATHOMNEMONICPBKDF2_ITERS — optional override for mnemonic PBKDF2-HMAC-SHA3-512 iterations (default is secure baseline in code)

6.1) Wallet/key model (current)

- Key generation is mnemonic-first (atho-mnemonic-v1) with default 24 words.
- Supported phrase lengths: 12 / 24 / 48.
- Mnemonic restore is deterministic by phrase + passphrase + path (network/account/role/index).
- Wallet APIs now include:
- POST /wallet/create
- POST /wallet/recover_mnemonic
- POST /wallet/export
- POST /wallet/import
- CLI commands include:
- wallet new [12|24|48]
- wallet recover
- wallet export
- wallet import

7) Where to start in code

- Src/Main/runnode.py — launcher
- Src/Node/fullnode.py / minernode.py — node entrypoints
- Src/Network/* — P2P, peers, sync
- Src/Storage/* — LMDB stores
- Src/Miner/* — mining loop + PoW
- Src/Blockchain/* — block/chain validation

8) Src/ module map (one■by■one)

This project is modular by domain. Below is a quick map of every top■level module under Src/ and its role.

- Src/Accounts/
- Key management (Falcon keys, export/import, format enforcement).
- Src/Api/
- FastAPI app, auth, API routes (wallet, chain, tx, mining, peers).
- Src/Blockchain/
- Core blockchain logic (block model, chain validation, genesis, reorg/orphan handling).
- Src/Config/
- Config files (API keys, port mappings, bootstrap seeds, etc.).
- Src/Falcon/
- Falcon signature integration and CLI/binaries.
- Src/Kyber/
- Vendored Kyber KEM implementation and wrappers used by wallet lockbox workflows.
- Src/GUI/
- Desktop GUI, CLI UI, Web Explorer (web_explorer.html), and local explorer bridge server.
- Src/Main/
- Launchers and main entrypoints (runnode.py, emission bootstrap, etc.).
- Src/Miner/
- Mining loop, PoW manager, candidate block building.
- Src/Network/

### Quickstart

Source: Docs/quickstart.md

### Atho Quickstart

Date: 2026-04-04

This guide is the canonical setup path for current development builds.

### 0) Prerequisites

- Python 3.9+
- git
- C/C++ compiler (clang or gcc)
- Bash shell for build scripts (.sh) on macOS/Linux/WSL
- For GPU OpenCL build: OpenCL runtime + headers
- For CUDA build (optional): NVIDIA GPU + nvcc (non-macOS)

LMDB native headers are required for the UTXO batch checker build.

- macOS: brew install lmdb
- Ubuntu/Debian: sudo apt-get install liblmdb-dev

### 1) Clone + virtualenv + dependencies

macOS/Linux:

Windows PowerShell:

### 2) Build and register all binaries

All native outputs are placed under:

- Binaries/<platform-tag>/...
- with hash metadata/pinning auto-updated in Binaries/pinregistry.json and Binaries/<platform-tag>/binarymeta.json

### 2.1 Falcon CLI (required)

Build + auto-register falcon_cli into Binaries/<platform-tag>/:

If you already have a prebuilt falcon_cli, register it directly:

### 2.2 Falcon FFI verify shared library (required for fastest verify path)

This prints export lines for:

- ATHOFALCONFFI_ENABLE=1
- FALCONVERIFYSO=<path>
- FALCONVERIFYSOSHA3384=<hash>

### 2.3 TX signing-body native bridge

Exports printed by script:

- ATHOTXSIGNING_SO=<path>
- ATHOTXSIGNINGSOSHA3_384=<hash>

2.4 UTXO batch checker native library

Exports printed by script:

- ATHOUTXOBATCHCHECKSO=<path>
- ATHOUTXOBATCHCHECKSOSHA3384=<hash>

2.5 CPU PoW native bridge

Exports printed by script:

- ATHOPOWNATIVE_SO=<path>
- ATHOPOWNATIVESOSHA3_384=<hash>

2.6 GPU miner binaries (OpenCL + optional CUDA)

Outputs:

- gpumineropencl (always attempted)
- gpuminercuda (only when nvcc exists and OS supports CUDA)

3) Strict binary mode (default)

Runtime now uses only Binaries/<platform-tag>/ paths by default. Explicitly set for clarity:

4) Create API credentials

5) Run node launcher

Recommended first run:

- start full + wallet/API
- then add miner role after baseline sync is healthy

Scripted terminal-only launch (no prompts):

6) Run GUI (optional)

6.1) Terminal sender + bond/stake flows

7) Performance-related runtime knobs

- ATHOTXVERIFY_WORKERS=<n>: parallel tx verification workers.
- ATHOUTXOBATCHCHECKENABLE=1: enable native UTXO batch path (default on).
- ATHOGPUBACKEND=auto|opencl|cuda: GPU backend selection override.
- ATHOBINARIESSTRICT=1: require canonical binary locations.

8) Basic health checks

9) Common failure points

- Falcon binary not installed for current platform tag.
- LMDB headers missing when building libutxobatchcheck.
- CUDA expected on macOS (CUDA build is skipped there).
- API auth not initialized.
- Stale background processes holding ports.

10) Current Consensus Runtime Defaults

Useful verification values for deployment checks:

- block target: 120s
- difficulty retarget interval: 180 blocks
- transaction confirmations required: 10
- coinbase maturity: 150 blocks
- fee floor: 350 atoms/vB
- min tx fee: 100,000 atoms
- BPoW enforcement height: 10,000
- bond requirement: 25 ATHO

Use:

- Troubleshooting.md
- Binaries.md
- Node_Stop.md

### Technical Whitepaper (Project)

Source: Docs/WhitePaper.md

### Atho Technical Whitepaper v3

Date: 2026-04-10

Technical PDF artifact: Docs/AthoWhitepaperv3.pdf Condensed overview PDF: Docs/AthoWhitepaperCondensed_v3.pdf

### 1. Executive Summary

Atho is a post-quantum UTXO blockchain that uses Falcon-512 signatures, SHA3-384 hashing, and a 120-second block target. The system is optimized around a hybrid architecture:

- Python for orchestration, APIs, and operator control.
- Native C/C++ binaries for cryptographic and performance-critical loops.

The current direction is throughput and determinism:

- compact binary transaction wire format,
- integer atom accounting end-to-end,
- hash-pinned native binary loading,
- CPU/GPU miner backends under one consensus path.

- Bonded Proof of Work (BPoW) with wallet staking as a separate role.

### 2. Current Protocol Direction

Compared with earlier snapshots, the current implementation formalizes these changes:

- Binary-first transaction transport:
- Canonical transport codec is compact binary (ATX2, with ATX1 decode compatibility).
- Witness is handled as raw bytes internally.
- Integer accounting standardization:
- Consensus-critical value math is integer atoms.
- Display formats remain user-facing (ATHO, decimal strings, Base56 addresses) without changing internal math semantics.
- Native acceleration and binary governance:
- Native bridges for signing-body hashing, UTXO batch checks, and CPU mining loops.
- GPU miner binaries (OpenCL baseline, CUDA where available).
- Per-binary SHA3-384 pinning and metadata in Binaries/.
- Updated throughput model:
- Capacity estimates now use measured/derived vsize distributions rather than a fixed baseline tx size.
- BPoW + wallet staking economics:
- miner eligibility is controlled by deterministic bond state,
- staking remains separate from block production,
- fee routing is deterministic and auditable at block level.

3. Protocol Snapshot

- Block interval target: 120 seconds.
- Difficulty retarget interval: 180 blocks.
- Transaction confirmations required: 10.
- Coinbase maturity: 150 blocks.
- Block cap: 3,500,000 base bytes, 14,000,000 weight units.
- Transaction policy cap: 250,000 vB per transaction.
- Transaction policy metric: SegWit-style vsize.
- Addressing: Base56 for users, HPK-based internal destination binding.
- Signature system: Falcon-512.
- Hashing: SHA3-384 for canonical tx/block identity operations.
- Active fee floor: 350 atoms/vB.
- Active min tx fee: 100,000 atoms.

3.1 Network Parameters (Operator Reference)

- Network mode set: mainnet, testnet, regnet
- Target block cadence: 120s
- Difficulty retarget window: 180 blocks
- Median-time-past window: 13 blocks
- Future drift bound: 30s
- Tx confirmations (standard): 10
- Coinbase maturity: 150
- Max block size: 3,500,000 bytes
- Max block weight: 14,000,000
- Max transaction size policy: 250,000 vB
- Fee floor: 350 atoms/vB
- Minimum transaction fee: 100,000 atoms
- Dust threshold: 250 atoms
- BPoW enforcement height: 10,000
- Bond requirement: 25 ATHO
- Bond activation confirmations: 25

- Slash penalty: 2.5 ATHO
- Epoch/finalization windows: 720 / 3,600 blocks

4. Transaction Encoding and Capacity

Atho no longer relies on one static tx-size assumption. Effective tx size depends on:

### Consensus

Source: Docs/Consensus.md

### Atho Consensus (Current)

Date: 2026-04-05

This document defines active consensus-enforced behavior for blocks, transactions, fee routing, and BPoW state transitions.

### 1) Core Constants

From Src/Utility/const.py:

- Block target time: 120 seconds
- Difficulty retarget interval: 180 blocks
- Transaction confirmations required: 10 blocks
- Coinbase maturity: 150 blocks
- Max block base bytes: 3,500,000
- Max block weight: 14,000,000
- Max tx policy size (vsize): 250,000
- Fee floor: 350 atoms per policy byte (vsize)
- Min tx fee: 100,000 atoms
- Dust limit: 250 atoms
- Pre-tail base supply target: 100,000,000 ATHO
- Hard max supply cap: 150,000,000 ATHO
- Supply floor: 21,000,000 ATHO

### 2) BPoW and Stake Constants

- BPoW enforcement height: 10,000 (mainnet, testnet, regnet)
- Bond requirement: 25 ATHO
- Bond activation confirmations: 25
- Bond unbond delay: 10,080 blocks
- Slash penalty: 2.5 ATHO
- Epoch length: 720 blocks
- Finalization buffer: 3,600 blocks
- Bootstrap allocation: 390,625 ATHO at block 1
- Stake minimum: 20 ATHO
- Stake max per address: 500 ATHO
- Stake max new entry (rolling 30 days, network-wide): 25,000 ATHO
- Stake max total locked (network-wide): 25,000,000 ATHO
- Stake unbond delay: 129,600 blocks (180 days)
- Stake rewards stop on unstake request: true

Primary data stores:

- bond.lmdb for miner bond lifecycle state
- stake.lmdb for wallet staking lifecycle state
- utxo.lmdb for standard spendable outputs

2.1 Deterministic Role Address Rules

Consensus role derivation is domain-separated from raw Falcon pubkey bytes:

- regular: SHA3-384("ATHOADDRV1" || network || pubkey)
- bond: SHA3-384("ATHOBONDV1" || network || pubkey)

- stake: SHA3-384("ATHOSTAKEV1" || network || pubkey)

Consensus does not trust user-facing string prefix alone. It validates deterministic derivation rules plus network role domains.

2.2 State Machines

Bond state machine:

- pending -> active -> exiting -> unlockable -> withdrawn

Stake state machine:

- pending -> active -> exiting -> unlockable -> withdrawn

Both are enforced by deterministic height and confirmation checks. No state transition is accepted if preconditions fail.

3) Fee Routing and Pool Rules

- Fee uplift policy: +25%
- Fee pool routing:
- pre-tail (height < 8,000,000): 40% of total fees (20% miner-side, 20% stake-side),
- post-tail (height >= 8,000,000): 55% of total fees (25% miner-side, 30% stake-side).
- Miner-side split:
- pre-tail: 0% winner-proportional, 20% bonded-idle split (of total fees),
- post-tail: 20% winner-proportional, 5% bonded-idle split (of total fees).
- Burn policy at tail: 100% burn on routed non-pool fees (45% of total fees at post-tail routing, subject to floor clipping).

Consensus-managed pool address is deterministic and network-separated:

- mainnet: "P" + Base56(SHA3-384("ATHOPROTOCOLPOOL_MAINNET"))
- testnet: "L" + Base56(SHA3-384("ATHOPROTOCOLPOOL_TESTNET"))
- regnet: "L" + Base56(SHA3-384("ATHOPROTOCOLPOOL_REGNET"))

4) Transaction Consensus Path

Primary code:

- Src/Main/txveri.py
- Src/Transactions/txvalidation.py

Consensus checks include:

### Falcon-512 Integration

Source: Docs/Falcon512.md

### Falcon-512 Integration (Current)

Date: 2026-04-10

This document is the active Falcon-512 integration reference for Atho. It covers signing and verification flow, byte-policy enforcement, runtime binary controls, and operational checks required to keep cryptographic behavior deterministic across nodes.

### 1) Runtime Model

Signing and verification are orchestrated through:

- Src/Accounts/falconcli.py

Consensus validation path references:

- Src/Transactions/txvalidation.py::validatesignature
- KeyManager.verifywithpubkey(...)
- FalconCLI.verify(...)

Runtime binary resolution prefers canonical pinned artifacts in Binaries/<platform-tag>/.

### 2) Key and Signature Byte Policy

Current policy constants in Src/Utility/const.py:

- canonical Falcon public key size: 897 bytes (FALCONPUBKEYBYTES),
- legacy public key size compatibility: 1024 bytes (FALCONPUBKEYLEGACY_BYTES), disabled by default,
- compressed signature target: 666 bytes (FALCONSIGCOMPRESSEDTARGETBYTES),
- accepted signature window: 600..690 bytes (FALCONSIGMINBYTES..FALCONSIGMAXBYTES).

Consensus checks enforce these bounds before invoking cryptographic verification.

### 3) Encoding and Wire Behavior

Atho uses a binary-first tx transport (ATX2) with witness as raw bytes. API compatibility layers may expose witness fields using canonical text-safe encodings, but validation always decodes back to byte form and applies policy checks on bytes.

Practical implication:

- display format can vary by API surface,
- consensus acceptance rules remain byte-deterministic.

4) Signing Flow

Standard signing pipeline:

- construct canonical no-witness signing body,
- hash signing body with SHA3-384,
- sign digest via Falcon CLI bridge,
- attach signature/public key witness fields.

Determinism requirements:

- canonical field ordering,
- stable serialization separators,
- explicit decimal/atom normalization before digesting.

5) Verification Flow

Standard verification pipeline:

- strip witness fields and reconstruct canonical signing body,
- recompute SHA3-384 digest,
- decode witness signature/public key,
- enforce byte-length policy,
- run Falcon verification via configured verifier path.

Any mismatch in canonical body reconstruction invalidates the signature regardless of witness payload shape.

6) Optional FFI Verification Path

An optional shared-library verifier can be enabled with explicit env configuration:

- ATHOFALCONFFI_ENABLE=1
- FALCONVERIFYSO=<absolute-path>
- FALCONVERIFYSOSHA3384=<expected-digest>

If FFI is unavailable or fails pin checks, runtime can fall back to CLI verification according to current policy flags.

7) Binary Pinning and Integrity Controls

Integrity metadata locations:

- Binaries/pin_registry.json
- Binaries/<platform-tag>/binary_meta.json

Strict mode:

With strict mode enabled, unexpected binary paths or digest mismatches are rejected. This protects against accidental drift and malicious binary substitution.

8) Operational Checklist

After Falcon-related changes:

- rebuild Falcon CLI/shared verifier artifacts,
- refresh pin metadata,
- run tx signing/verification smoke tests,
- run full unit/integration suite for tx validation paths,
- confirm explorer/API endpoints still decode witness fields correctly.

Command references:

9) Migration and Compatibility Notes

Legacy pubkey acceptance (1024-byte) should remain disabled unless a deliberate migration window is approved.
Re-enabling compatibility broadens acceptance surface and should be time-bounded with explicit network coordination.

When compatibility mode is temporarily enabled:

### SHA3-384 and Addressing

Source: Docs/Sha3-384.md

### SHA3-384, HPK, and Addressing

Date: 2026-04-10

This document explains how Atho uses SHA3-384 across identity, transaction digesting, block identity, and runtime integrity checks. It focuses on the active implementation profile, not historical variants.

### 1) Hash Primitive Role in Atho

Atho uses SHA3-384 as the default high-assurance digest primitive for consensus-relevant identity and signing flows. The practical result is a consistent 48-byte digest shape (96 hex characters) across major subsystems.

Core implementation entrypoints:

- Src/Utility/hash.py
- Src/Utility/txdhash.py
- Src/Transactions/tx.py
- Src/Transactions/txvalidation.py

SHA3-256 remains present for specific helper/checksum use cases where shorter digest width is sufficient and consensus behavior is unaffected.

### 2) HPK (Hashed Public Key) Model

HPK is derived from raw Falcon public key bytes through SHA3-384. This gives deterministic, fixed-length identity material for internal destination binding.

Key properties:

- digest size: 48 bytes,
- hex form: 96 chars,
- deterministic for identical input bytes,
- stable across display encoding changes.

Why this matters:

- internal storage and validation remain hash-native,
- user-facing address encodings can evolve without changing canonical identity semantics,
- code paths that operate on HPK avoid variable-length public key edge cases.

3) Public Key Length Policy

Current production constants in Src/Utility/const.py define:

- canonical Falcon public key length 897 bytes,
- optional legacy compatibility length 1024 bytes (disabled in normal operation unless migration policy enables it).

HPK derivation always hashes the raw byte payload accepted by validation policy. Display encoding is not part of HPK digest input.

4) Addressing Layer: HPK vs Base56

Atho separates internal and external address representations:

- internal: HPK-based deterministic identity,
- external: Base56 user-facing strings with network and role-aware prefixes.

Relevant code:

- Src/Blockchain/base56.py
- Src/Utility/const.py (network/role prefix policy)

Runtime behavior:

- Base56 decode maps back to canonical HPK form,
- Base56 encode wraps HPK with checksum/display affordances,

- consensus and accounting remain HPK-driven.

This separation keeps user ergonomics without weakening deterministic backend identity checks.

5) Transaction Hashing Roles

Atho distinguishes digest roles clearly:

- txid: deterministic no-witness identity hash.
- wtxid: full transaction hash including witness payload.
- signing digest: canonical no-witness signing body hash.

Operational impact:

- tx indexing and validation use stable digest semantics,
- witness malleability is isolated from no-witness identity where intended,
- signing verification always rebuilds canonical signing body before hashing.

6) Block and Runtime Identity

Block-level identity paths also rely on SHA3-384 conventions so digest width and tooling are consistent. The same primitive family is reused in binary pinning metadata, creating a cohesive integrity model across consensus and runtime binaries.

Binary integrity files:

- Binaries/pin_registry.json
- Binaries/<platform-tag>/binary_meta.json

Pin validation logic:

- Src/Utility/binarypin.py

### Transactions

Source: Docs/Tx.md

### Atho Transaction Reference (Current)

Date: 2026-04-05

This is the authoritative transaction behavior reference for current development builds.

### 1) Canonical Policy Metric

Atho uses SegWit-style policy metrics:

- weight = basebytes * 3 + totalbytes
- vsize = ceil(weight / 4)

vsize is used for:

- fee floor checks,
- mempool admission,
- miner packing,
- transaction size policy limits.

Active policy constants:

- fee floor: 350 atoms/vB
- minimum tx fee: 100,000 atoms
- tx confirmations required for regular spend visibility: 10

### 2) Binary Transport Codec

Current serializer emits compact binary payload magic:

- ATX2 (current)

Decoder accepts:

- ATX2
- ATX1 (legacy decode compatibility)

Code:

- Src/Transactions/txbinary.py
- Src/SigWit/sigwit.py

### 3) Field Strategy (Compact)

Inputs:

- spend reference: txoutid (<96-hex-txid>:<vout>)
- optional non-zero sequence via bitmap + varint in binary codec
- no full signature embedded per-input

Outputs:

- amount encoded from integer atoms
- destination binding encoded as 48-byte raw HPK digest
- lock flags compacted with bitmap

Witness:

- signature and public key are raw bytes in binary payload
- API/UI wire display remains canonicalized for compatibility/human tooling

4) Signature Reference Token

Input script_sig is a compact reference token, not a full signature blob.

Current canonical prefix:

- SR:<16-hex>

Legacy SIG_REF:* acceptance is policy-gated and disabled in current defaults.

5) Integer Accounting

Consensus-critical tx value logic uses integer atoms. Display conversion to decimal strings is presentation-only.

Examples:

- amount_atoms
- fee_atoms
- exact conservation checks in verifier paths

6) Witness Size Policy

Current Falcon witness policy bounds:

- signature target ~666 bytes (compressed range accepted by policy)
- canonical public key length 897 bytes
- legacy public key length acceptance is migration-policy controlled

7) Measured Size Profile (Hard Data)

There is no single fixed tx size. Size varies with:

- input count
- output count
- private vectors
- metadata
- witness payload length

The table below is measured from the live serializer + policy path:

- Transaction.to_dict()
- serializetxv1(...)
- Constants.txpolicymetrics(...)

Snapshot date: 2026-04-05

| Flow Shape | Base Bytes | Total Bytes | Weight | vsize | Binary Bytes | | --- | ---: | ---: | ---: | ---: | ---: | | coinbase (1 out) | 290 | 290 | 1160 | 290 | 290 | | public 1 in / 1 out | 121 | 1634 | 1997 | 500 | 1634 | | public 1 in / 2 out | 174 | 1687 | 2209 | 553 | 1687 | | public 2 in / 2 out | 223 | 1736 | 2405 | 602 | 1736 | | public 3 in / 2 out | 272 | 1785.

8) Private Multipliers vs Standard Public Send

Using public 1 in / 2 out (553 vB) as baseline:

- public -> private: 2471 / 553 = 4.47x
- private -> public: 2708 / 553 = 4.90x
- private -> private: 4673 / 553 = 8.45x

- private -> public with private change: 4680 / 553 = 8.46x

This confirms current private flows are not ~21x for representative single-input/single-recipient shapes. They are currently ~4.5x to ~8.5x, with larger multipliers when private input/output counts increase.

9) Private Size Sensitivity (Measured)

### SegWit and Witness Rules

Source: Docs/Sigwit.md

### Atho SegWit Sizing and Throughput Reference

Date: 2026-04-10

This document describes the active Atho sizing model for transaction weight, virtual size (vsize), and throughput estimation. It is intended for node operators, wallet/API integrators, and reviewers validating fee/tps assumptions.

### 1) Active Sizing Limits

From Src/Utility/const.py:

- MAXBLOCKSIZE_BYTES = 3,500,000
- MAXBLOCKWEIGHT = 14,000,000
- MAXTRANSACTIONSIZE_BYTES = 250,000 (policy vsize cap)
- WITNESSSCALEFACTOR = 4

These values establish the hard envelope for mempool admission and block packing. Any throughput estimate that ignores these limits is not aligned with current consensus policy.

### 2) Canonical Metric Definitions

Atho uses SegWit-style accounting:

- weight = basebytes * 3 + totalbytes
- vsize = ceil(weight / 4)

Interpretation:

- base bytes are weighted heavier,
- witness bytes are discounted relative to base,
- fee policy and packing policy are both driven by vsize.

Current fee baseline tied to vsize:

- fee floor 350 atoms/vB,
- minimum fee 100,000 atoms.

3) Base vs Witness Counting Rules

Base view (no-witness form):

- remove top-level witness fields,
- blank script_sig in input entries for deterministic no-witness sizing.

Total view:

- includes witness payload and all fields.

Implementation references:

- Src/SigWit/sigwit.py::SegWit.sizes_bytes
- binary tx serialization paths when available (ATX2 transport)

The important point is deterministic counting, not ad-hoc JSON length heuristics.

4) Throughput Formula

Let:

- Bv = 3,500,000 effective vbytes per block,
- T = 120 seconds per block target,
- F = packing factor (1.0 ideal, 0.95 conservative),
- Savg = average non-coinbase tx vsize.

Then:

- TPS ~= (Bv F) / (T Savg)

Use this as a planning model, not a guarantee. Real block composition, propagation, and tx mix always influence observed throughput.

5) Current Reference vsize Profiles

Representative estimates (compressed witness, no metadata-heavy edge cases):

- 1 in / 1 out: ~513 vB
- 1 in / 2 out: ~566 vB
- 2 in / 2 out: ~615 vB
- 3 in / 2 out: ~664 vB
- 4 in / 2 out: ~713 vB

These are useful baseline points for fee estimation and capacity planning. Production applications should still measure real rolling averages from their own observed workload.

6) Example TPS Bands

With T = 120:

- at 566 vB: 51.5 TPS ideal, 49.0 TPS at F=0.95
- at 615 vB: 47.4 TPS ideal, 45.1 TPS at F=0.95
- at 664 vB: 43.9 TPS ideal, 41.7 TPS at F=0.95
- at 713 vB: 40.9 TPS ideal, 38.8 TPS at F=0.95

Private flows are intentionally larger than public-only flows and therefore reduce effective TPS for privacy-heavy block compositions.

7) Why vsize Discipline Matters

Sizing discipline directly affects:

- fair fee market behavior,
- predictable miner packing,
- mempool anti-spam controls,

### API Authentication

Source: Docs/ApiAuth.md

### API Authentication & Permissions

Last refresh: 2026-04-04.

This note explains how Atho secures its HTTP API, how passwords/permissions work, and why this matters for protecting your node and funds.

### Model overview

- Every API request must include:
- X-API-Key – the token from Src/Config/Api_Keys.json
- X-User – username associated with the key (default atho)
- X-Pass – password for the user (hashed in the key file)
- Optional HMAC: when enabled, requests also carry X-TS and X-Sig to sign method/path/body with a derived secret.
- Permissions are per-key; missing scopes yield 403s. Keys are stored in Src/Config/Api_Keys.json.

### Permissions

- Required scopes include:
- read – general info endpoints
- explorer – block/tx explorer-like data
- send_tx – submit transactions
- mining – mining control/status
- walletadmin – wallet/key mutation endpoints (/wallet/create, /wallet/recovermnemonic, /wallet/import, /wallet/export, rename/delete/default)
- node_admin – node/config/security mutation endpoints
- Keys can have a subset; defaults are full permissions if a password is set, or read-only if no password is provided.
- The code migrates existing keys to include required scopes on startup.

Passwords and safety

- Passwords are hashed (SHA3-256 with a salt) in the key file; the plain password is never stored.
- Without X-Pass, a key cannot authorize sensitive actions even if the token is known.
- If you leave the password blank on creation, the key is restricted to read-only scopes, reducing blast radius for non-admin usage.

HMAC (optional but recommended when exposed)

- If REQUIRE_HMAC is enabled, every request must include X-TS and X-Sig; the server verifies freshness (±60s) and integrity over method/path/body.
- HMAC secrets are derived per key (or you can store an explicit secret).
- Protects against replay and tampering; use when exposing APIs beyond localhost.

File locations

- API keys: Src/Config/Api_Keys.json
- Auto-creation: the entrypoint (docker/docker-entrypoint.sh) calls ensureapikey_auto() if no keys exist; cliui/runnode call interactive setup.

Why this matters

- The API can submit transactions, manage mining, and expose node data. Without authentication, anyone could double-spend, drain wallets, or disrupt your node.
- Strong passwords + per-key scopes + optional HMAC provide layered defense. Keep Api_Keys.json private (chmod 600) and rotate keys periodically.
- API authentication protects request authorization. Release integrity checks (checksums/signatures) are a separate control layer and should be used for distributed binaries.

Operational tips

- Local dev: HMAC off, HTTPS off, bind to localhost; use strong passwords anyway.
- Exposed/production: enable HMAC, run behind TLS/reverse proxy, limit IPs via firewall, use least-privilege keys (separate read-only vs. send/mine).
- Back up Api_Keys.json securely; losing it locks you out, leaking it compromises your node.

Performance behavior (wallet endpoints)

### LMDB Storage

Source: Docs/LMDB.md

### Atho LMDB Storage Overview

Date: 2026-04-09

This document defines which LMDB stores are consensus-critical vs local/runtime state, and why each store exists.

### 1) Database Root

Root path:

- blockchain_storage/<network>/

Configured in:

- Src/Utility/const.py (Constants.DATABASES)

### 2) Store Classification

### Consensus-Critical Stores (network state)

These stores are replayed/validated from canonical blocks and directly affect validation outcomes.

- fullblockchain*.lmdb
- Code: Src/Storage/blockstorage.py
- Role: canonical block bodies, tx location index, tip metadata.
- Consensus impact: highest.
- utxo*.lmdb
- Code: Src/Storage/utxostorage.py
- Role: spendable output set + apply/unspend markers.
- Consensus impact: highest.

- private_notes*.lmdb
- Code: Src/Storage/privatenotesstore.py
- Role: shielded note set, nullifier sets, Merkle tree nodes, anchor roots.
- Consensus impact: highest for private tx validity.
- bond*.lmdb
- Code: Src/Storage/bondstake.py
- Role: bond lock state machine, miner eligibility linkage.
- Consensus impact: high (bonded mining checks).
- stake*.lmdb
- Code: Src/Storage/bondstake.py
- Role: stake lock state machine + network totals used by staking logic.
- Consensus impact: high (staking constraints/rewards).
- payout_journal*.lmdb
- Code: Src/Storage/bondstake.py
- Role: deterministic payout idempotency/event history.
- Consensus impact: high for safe replay/reorg payout behavior.
- Why it is required:
- prevents duplicate payouts across retries/restarts,
- preserves ordered payout transitions (pending -> submitted -> confirmed),
- enables deterministic reprocessing after crashes/reorg windows.

Local Runtime/Propagation Stores

Important for networking and performance, but not canonical chain truth.

- mempool*.lmdb
- Code: Src/Storage/mempool.py
- Role: pending tx propagation, fee/ranking indexes, tombstones.
- Canonical source remains accepted blocks, not mempool records.
- orphan_blocks*.lmdb
- Code: Src/Blockchain/orphreorgs.py
- Role: orphan/side-branch staging for bounded reorg handling.
- Becomes canonical only when blocks are accepted into block store.

3) Runtime Deletion/Recovery Expectations

- Deleting a consensus store while node is running is unsafe by default.
- Current hardening behavior:
- LMDB begin paths detect missing backing paths,
- affected stores attempt environment recovery/reopen,
- consensus code still treats missing/corrupt required state as validation

failure rather than accepting uncertain state.

- Operational recommendation:
- stop node before manual DB deletion unless intentionally testing recovery.

4) Map Growth / Rollover

Atho uses bounded LMDB map growth:

- on MapFullError, stores may grow map size (if enabled),
- growth is capped by per-store limits,
- near-limit pressure is logged.

Important:

- LMDB page reuse does not shrink map size automatically.
- pruning/cleanup frees pages for reuse but does not compact file size.

5) Operational Checks

- Inspect store health:
- Watch logs for map pressure and recovery events.
- Increase max map size before sustained high-write load if near limits.

6) Related Files

- Src/Storage/lmdbutils.py
- Src/Storage/blockstorage.py
- Src/Storage/utxostorage.py
- Src/Storage/privatenotesstore.py
- Src/Storage/mempool.py

### Base56 Addressing

Source: Docs/Base56.md

### Base56 Address Encoding

Last refresh: 2026-04-04.

This note explains why Atho uses Base56 for addresses instead of Base58 or other alphabets.

### Why Base56

- Ambiguity reduction: Drops lookalike characters to cut down on copy/paste and transcription errors.
- Human-friendly: Better readability in CLI, logs, and printed/exported materials; fewer mistakes when reading aloud or over screenshares.
- Consistent casing: One unambiguous alphabet across all components (CLI, API responses, wallet UI) so there is one canonical representation.

### How it’s built

- HPK (hashed public key) is derived from the Falcon public key (SHA3-384).
- HPK bytes are encoded with the Base56 alphabet (digits/letters chosen to avoid lookalikes).
- Decoding reverses the process to recover the HPK for validation and address matching.

### Characters removed (6) and why

Removed from the larger Base62 set because they are visually confusing:

- O (uppercase O) – looks like zero
- 0 (zero) – looks like letter O
- I (uppercase i) – looks like l/1
- l (lowercase L) – looks like I/1
- 1 (one) – looks like I/l
- i (lowercase i) – too similar to I/l

Atho Base56 alphabet (safe set)

Uppercase: A B C D E F G H J K L M N P Q R S T U V W X Y Z Lowercase: a b c d e f g h j k m n p q r s t u v w x y z Digits: 2 3 4 5 6 7 8 9

Benefits of removing lookalikes

- Avoids misreading addresses and wrong transfers
- Reduces fraudulent lookalike addresses
- Cuts typing errors on mobile and across fonts/screen sizes
- Mirrors the rationale of Base58 and other error-resistant encodings

Why not Base58 or Bech32?

- Base58 still includes ambiguous glyphs for some fonts/locales; Base56 removes more of them.
- Bech32 adds checksums and HRPs, but is longer; Atho prioritizes shorter, readable strings with a controlled alphabet. A checksum can be layered separately if desired.

Security: encoding vs. strength

- Base56 is purely an encoding choice for human readability; it does not reduce cryptographic strength. The underlying security comes from Falcon-512 signatures and SHA3-384 hashing of public keys.

- SHA3-384 offers a larger digest than SHA-256, providing stronger collision resistance and headroom against preimage/collision attacks; the Base56 text is just a view of those bytes.
- Collusion/lookalike resistance: by removing ambiguous characters and using explicit network prefixes, it's harder to trick users with visually similar addresses, while the HPK (SHA3-384) remains the authoritative identifier internally.

How addresses are derived internally (HPK)

- Public key → SHA3-384 → HPK (hashed public key) → Base56 address.

### Key Manager

Source: Docs/KeyManager.md

### Key Manager (Atho)

Last refresh: 2026-04-04.

This note explains how Atho's key manager stores, loads, and uses keys for wallets, mining rewards, and API authentication.

### What it does

- Generates Falcon-512 keypairs for wallet/miner addresses.
- Defaults to mnemonic-backed deterministic key generation (atho-mnemonic-v1) with 24 words (12/24/48 supported).
- Derives hashed public keys (HPK) and Base56 addresses for display/UX.
- Persists keys atomically to Keys/KeyManagerPublicPrivate_Keys.json (one file per node/instance).
- Supplies keys to transaction signing, mining coinbase outputs, and API auth where needed.
- Imports existing Falcon keys (see below) with hash/base56 verification and a test-sign before storing.

### Storage layout

- Path (by default): Keys/KeyManagerPublicPrivate_Keys.json
- Structure:
- miner1, miner2, …: labels for each managed key.
- public_key: Falcon public key (stored as normalized hex; canonical policy target is 897 bytes, legacy 1024-byte compatibility may still be accepted by current policy).
- private_key: Falcon private key components (F, G, f, g).
- mnemonicphrase, mnemonicwords, mnemonictext, mnemonicmeta for mnemonic-derived keys.
- Metadata: creation time, role (e.g., miner), network.
- Files are written atomically to avoid partial writes and corruption.

Encrypted-at-rest lockbox mode

- Key file now supports encrypted-at-rest mode with one-password unlock flow.
- Kyber DEK wrap is mandatory in lockbox mode (no Kyber opt-out).
- Lockbox unlock modes:
- default: password-only UX (Kyber secret wrap stored in wallet metadata).
- advanced: password + Kyber unlock text file import on unlock (.txt bundle).
- Encryption stack:
- password KDF: Argon2id
- payload cipher: AES-256-GCM
- DEK wrap: AES-256-GCM
- required PQ wrap: Kyber (kyber1024_ref by default) wrapping the same DEK for hybrid/PQ backup integrity.
- Plaintext fields (kept visible for wallet UX while locked):
- identifier, role, network, defaults, hashedpublickey, address_base56, source/version.
- Encrypted fields:
- raw publickey / clipublic_key
- all private key parts (f/g/F/G)
- mnemonic fields (mnemonicphrase, mnemonicwords, mnemonictext, mnemonicmeta).
- Security metadata is stored in walletsecurity (schema athokey-lockbox-v1).
- Advanced-mode Kyber unlock file format:
- schema: athokey-kyber-unlock-v1

- includes strict integrity hash and binding metadata
- importer fails closed on tampered/invalid format.

Kyber + AES-256 together (why both)

Key point: Kyber does not replace AES for payload encryption. They serve different roles.

- AES-256-GCM role:
- Encrypt the actual wallet secret payload (private key parts + mnemonic data).
- Provide confidentiality + integrity tag over the payload.
- Kyber role:
- Wrap/encapsulate the DEK used by AES payload encryption.
- Provide a post-quantum recovery/unlock control plane for key material.

Practical model:

- Generate random DEK.
- Encrypt wallet secret payload with AES-256-GCM(DEK).
- Wrap DEK with password-derived KEK (Argon2id + AES-256-GCM wrap).
- Also wrap the same DEK with Kyber KEM metadata/ciphertext.
- Store encrypted payload + both wrap records in lockbox metadata.

This gives:

- fast encryption performance (AES),
- password UX for daily use,

## Emissions and Monetary Policy

Source: Docs/Emissions.md

## Atho Emissions and Fee Routing Policy

Date: 2026-04-05

This document defines active monetary policy and fee routing under current consensus constants.

### 1) Accounting Units

- All consensus accounting is integer-atom based.
- 1 ATHO = 1,000,000,000 atoms.

### 2) Reward Schedule (Current)

- Block time target: 120 seconds.
- Blocks/year: 262,800.
- Era size: 1,000,000 blocks (~3.805 years).

Pre-tail eras:

- Era 1: 50 ATHO/block
- Era 2: 25 ATHO/block
- Era 3: 12.5 ATHO/block
- Era 4: 6.25 ATHO/block
- Era 5: 3.125 ATHO/block
- Era 6: 1.5625 ATHO/block
- Era 7: 0.78125 ATHO/block
- Era 8: 0.390625 ATHO/block

Bootstrap and pre-tail totals:

- One-time bootstrap allocation: 390,625 ATHO at block 1.
- Pre-tail subsidy target: 99,609,375 ATHO.
- Total pre-tail base target = 99,609,375 + 390,625 = 100,000,000 ATHO.
- Hard max supply cap = 150,000,000 ATHO.

Tail:

- Tail reward 0.1953125 ATHO/block from height 8,000,000 onward.

- Tail activation occurs at ~30.44 years from genesis.
- Tail annual issuance: 51,328.125 ATHO/year.

Hard-cap clipping rule:

- Subsidy is clipped at coinbase path by remaining cap headroom.
- Once totalemittedatoms reaches 150,000,000 ATHO, subsidy is 0 for all later heights.
- Fee routing/burn/pool accounting remains unchanged (fees do not mint new supply).

3) Why This Bounded Model Exists

Atho uses three monetary anchors for stability and auditability:

- 100,000,000 ATHO base issuance path:
- defines the long-form launch distribution and predictable early-to-mid lifecycle issuance,
- gives operators and integrators a deterministic emission curve to model from genesis.
- 150,000,000 ATHO hard max supply cap:
- enforces a strict upper bound on minted supply at consensus level,
- prevents perpetual subsidy expansion and guarantees issuance eventually terminates (subsidy = 0 after cap is reached).
- 21,000,000 ATHO protected supply floor:
- ensures burn logic cannot reduce effective circulating supply below a minimum economic base,
- protects against pathological over-burn in sustained high-fee periods.

Design intent:

- The base path controls structured distribution,
- the max cap bounds long-run inflation risk,
- the floor bounds long-run deflation pressure from fee burn.

Result:

- supply is constrained inside a deterministic corridor and remains consensus-verifiable on every block.

4) How This Differs and Why It Matters

How Atho differs:

- It combines a finite hard cap (150M) with a structured base path (100M) and an enforced lower floor (21M).
- Many networks implement only one side of this envelope:
- cap-only systems constrain inflation but do not include a protocol burn-floor guardrail,
- perpetual-tail systems preserve ongoing issuance but do not hard-stop minting at a fixed maximum.

### Inflation/Deflation Model

Source: Docs/Inflation_Deflationary.md

### Atho Inflationary/Deflationary Network Model

Last refresh: 2026-04-05.

### Scope

This file explains how Atho transitions between inflationary and deflationary behavior under active consensus policy.

Canonical references:

- Src/Utility/const.py
- Src/Main/blockveri.py
- Src/Main/consensus.py
- Src/Main/emission.py

### Executive Summary

- Block time: 120 seconds (262,800 blocks/year).
- Era size: 1,000,000 blocks (~3.805 years).
- Eight pre-tail eras: 50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 ATHO/block.
- Bootstrap allocation: 390,625 ATHO at block 1 (included in total pre-tail base).
- Tail starts at height 8,000,000 (~30.44 years) at 0.1953125 ATHO/block.
- Tail annual issuance: 51,328.125 ATHO/year.

- Hard max supply cap: 150,000,000 ATHO.
- Fee floor: 350 atoms/vB (3.5e-7 ATHO/vB).
- Fee routing: pre-tail 40% to consensus pool, post-tail 55% to consensus pool.
- Tail burn target on non-pool share: 100% burn / 0% miner, with floor clipping.
- Supply floor: 21,000,000 ATHO.
- Protocol deflation threshold: ~35.43% sustained utilization.

Emission Schedule

| Phase | Height range | Reward (ATHO/block) | Issuance (ATHO) | Cumulative (ATHO) | |---|---|---:|---:|---:| | Era 1 | 0 .. 999,999 | 50.0 | 50,000,000 | 50,000,000 | | Era 2 | 1,000,000 ..

Total pre-tail base including bootstrap:

- 99,609,375 ATHO subsidy path + 390,625 ATHO bootstrap at block 1 = 100,000,000 ATHO.
- Remaining mintable headroom after pre-tail base: 50,000,000 ATHO (tail subsidy only, then subsidy suppresses to zero).

Hard-cap rule:

- total_supply <= 150,000,000 ATHO.
- Subsidy is clipped at coinbase by remaining cap headroom.
- After cap is reached, subsidy remains 0 and fees continue through the same burn/pool paths.

Fee and Burn Math

Per block at full utilization

- Total fee capacity at floor: 3,500,000 * 350 = 1,225,000,000 atoms = 1.225 ATHO
- Routed-to-pool share at tail (55%): 0.67375 ATHO
- Burn-path share at tail (45%): 0.55125 ATHO
- Tail issuance per block: 0.1953125 ATHO
- Net per-block formula: Delta_block = 0.1953125 - (0.55125 * utilization)

Per year at full utilization

- Tail issuance: 0.1953125 * 262,800 = 51,328.125 ATHO/year
- Total annual fees at floor: 321,930 ATHO/year
- Annual burn-path max (45%): 144,868.5 ATHO/year
- Annual net formula: Delta_year = 51,328.125 - (144,868.5 * utilization)
- Formula above applies while subsidy is still positive; after hard-cap clip, subsidy term is 0.

Deflation Threshold

Set annual net to zero:

51,328.125 - 144,868.5u = 0 u = 51,328.125 / 144,868.5 ~= 0.3543

So:

- <35.43%: inflationary protocol net.
- =35.43%: neutral.

### Threat Model

Source: Docs/Threat.md

### Threat Model and Security Posture

Last refresh: 2026-04-04.

This document reviews current security posture across keys, signing, networking/auth, consensus, storage, and operational controls. It calls out weaknesses, impacts, and recommended improvements with code references.

### Overview: high-risk areas

- Key files can still be left unencrypted in legacy/plain mode; lockbox encryption should be enabled for production wallets.
- Falcon CLI integrity depends on maintaining the pinned digest map per release; misconfigured/empty pins on required networks block startup.
- Runtime guard exists but may be left in audit mode; live networks should run enforce mode + signed manifest requirement.
- API auth relies on key+user+pass; HMAC optional; no TLS termination described.

- Signature policy currently allows legacy 1024-byte pubkey compatibility in some paths; keep rollout policy explicit and minimize compatibility windows.
- Verbose logging leaks metadata (key lengths/previews, tx details).
- Tail issuance is active, with post-tail non-pool fee burn target (100% of the non-pool 45% share, floor-clipped) and a hard circulating-supply floor (21M ATHO).
- LMDB map-size/permissions misconfiguration can cause DoS; no WAL/backup guidance in code.
- P2P now has basic per-IP request rate limits, but API-level throttling and global mempool caps are still limited.

Cryptography and key management

- Key storage: Keys/KeyManagerPublicPrivate_Keys.json supports encrypted lockbox mode, but legacy/plain mode is still possible. Impact if plaintext mode is used: disk compromise → key theft.
- Key generation/import: Src/Accounts/key_manager.py, wimport.py accept CLI-format Falcon-512 keys. Some import paths log previews.
- Signing pipeline: KeyManager.sign_transaction → FalconCLI.sign now passes key material via stdin (--key-stdin) instead of temp files. Impact reduced vs disk temp files; residual risk remains in process memory and local host compromise.
- Signature verification: transaction validation enforces signature byte bounds and pubkey length policy from Constants (compressed-signature range with canonical/legacy pubkey rules). Impact: overly broad compatibility windows can increase ambiguity.
- Hashing: SHA3-384 for txid/signing/HPK and SHA3-256 for address checksum. Consistent; risk is mostly around normalization (handled in txdhash.py).

Falcon CLI binary integrity

- Binary discovery: Src/Accounts/falconcli.py searches PATH and repo locations. On required networks, strict pinning + file-permission checks are applied before use.
- Hash pinning: version-bound pinning is enforced with FALCONCLIPINDOMAIN + FALCONCLIPINNEDVERSIONDIGESTS[CONSENSUSVERSION]. If pinning is required and no digest is configured, startup fails.
- FFI fallback: loads FALCONVERIFYSO if present. Impact: loading untrusted shared object can run arbitrary code.

Runtime integrity controls (implemented)

- Runtime guard: Src/Utility/versioning.py snapshots critical constants and critical file hashes at startup and checks drift periodically.
- Modes:
- audit (default): logs violations, does not stop node.
- enforce: fail-closed behavior in full/miner/wallet entrypoints.
- Manifest verification:

### Docker Operations

Source: Docs/Docker.md

### Docker Guide (Atho)

Last refresh: 2026-04-04.

This guide walks through building and running Atho nodes with Docker/Compose on Linux, macOS, and Windows (Docker Desktop). It also shows how to run multiple nodes without port conflicts and how to use the CLI container.

### Prereqs

- Install Docker (Docker Desktop on macOS/Windows; Docker Engine on Linux).
- Optional: docker-compose v1/v2 (Compose v2 is built into recent Docker Desktop).

### What’s in the repo

- docker/Dockerfile: builds a Python 3.11 image, installs deps, builds falcon_cli, and runs Src/Node/fullnode.py by default.
- docker/docker-entrypoint.sh: simple entrypoint wrapper.
- docker/docker-compose.yml: services fullnode, miner, miner2, optional fullnode2, and cli. P2P is fixed to 56000 in-container; host ports are distinct (56000/56001/56002/56003).
- Bootstrap defaults: fullnode sets ATHO_BOOTSTRAP="" (no external seed); miners bootstrap to fullnode:56000 on the compose network.
- P2P bootstrap seeds are now env-only (ATHO_BOOTSTRAP as comma-separated host:port); no hardcoded seeds.

Quick start (full node + miners + CLI, internal networking)

This builds the image and starts:

- fullnode: P2P 56000 and API 10100 inside the compose network.
- miner: P2P 56000 (host bind 56001) and API 10200 in-container (host port randomized); bootstraps to fullnode:56000.
- miner2: P2P 56000 (host bind 56003) and API 10250 in-container (host port randomized); bootstraps to fullnode:56000.
- cli: interactive CLI pointing at http://fullnode:10100.

Stop: docker compose -f docker/docker-compose.yml -f docker/docker-compose.override.yml down

Run just a full node (no miner/cli)

Run just a miner (no full/cli)

Be sure there's a peer to bootstrap to (in this repo the compose default is fullnode:56000):

Run both nodes (no cli)

Accessing the CLI

If the CLI container exits (after a session), rerun it:

Running multiple nodes (no host port conflicts)

Use the built-in services; they run on internal ports (P2P 56000, APIs 10100/10200/10250) with separate volumes. For more nodes, duplicate a service with a new name and distinct volumes.

Now run docker compose -f docker/docker-compose.yml -f docker/docker-compose.override.yml up --build and both nodes run on internal ports (56000/10100/10300). CLI can target either by setting ATHOAPIURL to http://fullnode:10100 or http://fullnode2:10300.

Exposing ports to the host (optional)

If you want to reach a node from the host, add ports: with unique host mappings:

Use different host ports per node (e.g., 56001/10101, 56002/10102, etc.). Then set CLI ATHOAPIURL to http://localhost:10101 for that node.

Running the image manually (without compose)

Configuring CLI target

- Internal compose network: ATHOAPIURL=http://fullnode:10100 (default in CLI service).

### Docker Test Harness

Source: Docs/dockertest.md

### Docker P2P Test Harness

Last refresh: 2026-04-04.

Run two full nodes plus two miners and CLI locally via Docker to exercise the P2P network. All nodes speak P2P on port 56000 inside the network; host bindings differ so they can coexist on one machine.

### Prerequisites

- Docker Desktop running
- Repo root: /path/to/<repo-dir>

### Bring up the stack

### Observe logs

Run in another terminal to watch node activity:

### Interact with the CLI

### Hit APIs directly

- Primary full node: curl http://127.0.0.1:10100/chain/info
- Secondary full node: curl http://127.0.0.1:10300/chain/info
- Miner1: resolve host port first with docker compose port miner 10200, then curl http://127.0.0.1:<resolved_port>/chain/info
- Miner2: resolve host port first with docker compose port miner2 10250, then curl http://127.0.0.1:<resolved_port>/chain/info

### Ports and data

- P2P (in-container): 56000 for every node.

- P2P (host bindings): fullnode 56000→56000, fullnode2 56002→56000, miner1 56001→56000, miner2 56003→56000 (use docker compose ps to confirm).
- APIs (host): fullnode 10100, fullnode2 10300; miner1/miner2 use random host ports mapped to container ports 10200/10250 (check via docker compose port miner 10200 and docker compose port miner2 10250).
- Data/logs are volume-mounted per node:
- Primary: ./blockchain_storage, ./Keys, ./Logs
- Secondary: ./blockchainstoragenode2, ./Keysnode2, ./Logsnode2
- Miner1: ./blockchainstorageminer, ./Keysminer, ./Logsminer
- Miner2: ./blockchainstorageminer2, ./Keysminer2, ./Logsminer2

Bootstrap to an external/host node

- Set the seed to your host IP/P2P port (default P2P is usually 56000 unless you overrode it):
- Verify connectivity from inside a container:
- Check logs for handshakeok and syncheaders_ok.

Shutdown

Reset to clean runtime data

If you need to wipe all local data/peers and start fresh:

Then rerun the build/up steps above.

Ten-node add-on (for P2P stress)

- Assumes the base stack above is already up (fullnode is the bootstrap peer).
- Adds 9 more full nodes (nodes 2–10) on the same Docker network with unique host ports and isolated data.

Start them:

Check one of the extra nodes:

Tear them down (keeps volumes):

Per-terminal commands (copy/paste)

- Terminal 1: build + start stack
- Terminal 2: follow logs for built-in services
- Optional CLI shell
- API checks

Host + Docker sync test on P2P port 56000

If you run a seed/full node on the host and want Docker nodes to join it on P2P 56000:

Verify connectivity from inside a container:

Watch logs for handshakeok / syncheaders_ok.

### Build and Binaries

Source: Docs/Binaries.md

### Atho Binary Build and Pinning Guide

Date: 2026-04-10

This document defines the production binary layout, build flow, and pinning controls for Atho native components. The goal is deterministic runtime loading with explicit integrity checks, while keeping build and deployment workflows straightforward for operators.

### 1) Canonical Binary Layout

All runtime binaries are loaded from:

- Binaries/<platform-tag>/

Typical platform-tag examples include architecture and OS identity (for example, macOS arm64 vs Linux x86_64). Runtime should not depend on ad-hoc source-tree paths.

Common binary artifacts:

- falcon_cli(.exe)
- libathofalconverify.(dylib|so|dll)

- libtxsigningbody_bridge.(dylib|so|dll)
- libutxobatchcheck.(dylib|so|dll)
- libathopowcpu.(dylib|so|dll)
- gpumineropencl(.exe)
- gpuminercuda(.exe) when CUDA build is available

### 2) Pin Registry and Metadata

Integrity metadata is maintained in:

- Binaries/pin_registry.json
- Binaries/<platform-tag>/binary_meta.json

These files map expected SHA3-384 digests to binary names and are consumed by runtime pin validators. If a binary is rebuilt and metadata is not refreshed, strict mode will reject the load.

3) Strict Loading Mode

Enable strict canonical-path loading with:

Behavior in strict mode:

- runtime only loads from canonical Binaries/<platform-tag>/ paths,
- legacy fallback paths are disabled,
- hash mismatches fail closed.

This is the recommended production setting.

4) Build Commands

From repository root:

Run these in your release pipeline before packaging or smoke tests.

5) Script Output Variables

Build scripts emit expected environment values so downstream tooling can verify exact artifacts.

Examples:

- Falcon FFI verifier:
- ATHOFALCONFFI_ENABLE
- FALCONVERIFYSO
- FALCONVERIFYSOSHA3384
- TX signing bridge:
- ATHOTXSIGNING_SO
- ATHOTXSIGNINGSOSHA3_384
- UTXO checker:
- ATHOUTXOBATCHCHECKSO
- ATHOUTXOBATCHCHECKSOSHA3384
- CPU PoW bridge:
- ATHOPOWNATIVE_SO
- ATHOPOWNATIVESOSHA3_384

6) GPU Build Notes

GPU build behavior is backend-aware:

- OpenCL build is attempted broadly,
- CUDA build runs only where toolchain and platform support are present,
- on macOS, CUDA output may be intentionally absent.

Treat missing CUDA binary on unsupported hosts as expected, not a build failure.

7) Release Packaging

Release helper:

This produces timestamped bundles under:

- releases/binaries/<timestamp>/

Recommended release flow:

- build all required binaries,
- verify pin metadata updates,
- run runtime smoke tests in strict mode,
- package release payload,
- generate and sign checksums for distribution.

8) Common Failure Modes

Most frequent issues:

- wrong platform artifact loaded,
- missing LMDB headers during native build,
- hash mismatch after rebuild without metadata update,
- strict mode enabled while a required binary is absent.

First response:

- confirm active platform-tag path,
- inspect pin metadata for expected digest,
- rebuild artifact + refresh metadata,
- retest with strict mode still enabled.

9) Security and Policy Context

Binary integrity controls complement consensus policy but do not replace consensus validation. Active policy values remain independent:

- block target 120s, retarget 180,
- tx confirmations 10, private tx confirmations 10, coinbase maturity 150,
- fee floor 350 atoms/vB.

### Troubleshooting

Source: Docs/Troubleshooting.md

### Troubleshooting Guide (Atho)

Last refresh: 2026-04-04.

This guide collects common issues and fixes by area (network/P2P, APIs/auth, storage/LMDB, mining, Docker, build/tooling).

### Most Common Right Now (Fast Fixes)

- Nodes won’t start from GUI / start inconsistently
- Cause: stale background processes or PID conflicts.
- Fix:
- ./.venv/bin/python Src/Main/stop.py --active
- ./.venv/bin/python Src/Main/stop.py --all
- start again from GUI Node tab or Src/Main/runnode.py.
- GUI says auth/API key issue
- Cause: API key/user/password not set or mismatched between GUI and Api_Keys.json.
- Fix:
- regenerate key: ./.venv/bin/python Src/Api/auth.py
- re-enter values in GUI Settings.
- Falcon/Kyber binary or compile failures
- Cause: missing compiler/toolchain or wrong platform binary.
- Fix:
- follow compile sections in README.md and Docs/quickstart.md
- register Falcon binary:
- ./.venv/bin/python Src/Falcon/Falcon/installplatformbinary.py
- confirm compiler exists (clang/gcc/cc).
- Peer connects but later drops (Connection refused)

- Cause: peer process stopped, wrong target port, or service restart timing.
- Fix:
- verify target node process still running (stop.py --active)
- verify correct P2P port from Src/Config/NodePorts.json
- restart both peers and recheck logs/mainnet/network/network.log.
- Remote peers cannot connect to bootstrap IP
- Cause: bootstrap set to private LAN IP (192.168.x.x, 10.x.x.x, 172.16-31.x.x).
- Fix:
- use public endpoint (DDNS/public IP + port-forward) or VPN mesh endpoint (Tailscale/ZeroTier).
- keep LAN IP only for local-network testing.

Network / P2P

- Peers=0 / no headers: Check ATHOBOOTSTRAP and network (ATHONETWORK) match your seed; ensure P2P port (default 56000) is reachable. Clear peers: rm Src/Config/Peers.json and restart with ATHORESETPEERS=1.
- syncnoheaders or tip stuck at 0: Often a network mismatch or genesis mismatch. Align ATHO_NETWORK across nodes; verify bootstrap points to the right chain.
- Block rejects like 'Transaction' object has no attribute 'get': Incoming blocks weren't deserialized to the expected dicts. Ensure all nodes run the same code/version; wipe stale storage if formats changed.
- Port in use: Pick a free P2P/API port (ATHOP2PPORT, ATHOFULLNODEAPIPORT, ATHOMINERNODEAPIPORT) or let runnode.py auto-assign. On Docker, avoid binding API ports if they conflict; use random host ports.
- Different tips across nodes: Confirm all nodes point to the same bootstrap seed and network; if a node lags, clear its storage/peers and resync.
- Repeated syncheadersok but no progress: The serving peer may be on a short fork or serving invalid blocks. Point to a trusted seed with the desired tip; stop low-tip peers.

### Node Stop Utility

Source: Docs/Node_Stop.md

### Node Stop Utility (Src/Main/stop.py)

Last refresh: 2026-04-10

This utility provides a deterministic way to stop Atho node processes, including nodes started from different shells, launch scripts, or background sessions. It is designed for operational safety: match only known Atho process patterns, stop gracefully first, then force-stop only when required.

### 1) Why This Utility Exists

runnode.py and related launch paths can start multiple long-lived processes. Without a centralized stop mechanism, operators may rely on manual process hunting, which is error-prone and risks killing unrelated Python workloads.

stop.py addresses this by providing:

- role-aware stop actions (full, miner, wallet, launcher),
- all-node stop behavior,
- explicit PID targeting,
- list and watch modes for live visibility,
- reusable Python-callable functions for API/GUI integration.

### 2) Targeted Process Scope

Default target process names:

- fullnode.py
- minernode.py
- walletnode.py
- runminer.py
- runnode.py (launcher)

Scope safety rule:

- the tool is intentionally narrow,
- it does not terminate arbitrary Python processes,

- operator intent is explicit through flags or menu selection.

3) CLI Usage

From repository root:

No flags opens interactive mode with role-specific options.

Non-interactive examples:

4) Programmatic API Functions

Importable utilities for integrations:

Main callable functions:

- discovernodeprocesses(include_launcher=True)
- listnodeprocesses(include_launcher=True)
- activenodessnapshot(include_launcher=True)
- stopnodes(target="all", includelauncher=True, timeoutseconds=5.0, forcekill=True, dry_run=False)
- stoppids(pids, timeoutseconds=5.0, forcekill=True, dryrun=False)

5) Signal Escalation Behavior

Stop sequence per process:

- send SIGTERM,
- wait up to timeout (default 5s),
- if still alive and force-kill enabled, send SIGKILL.

This preserves graceful shutdown opportunities for state flush and log closure while ensuring stuck processes cannot persist indefinitely.

6) API and GUI Integration

The same stop logic is used by API/GUI stop routes so user actions can terminate:

- processes started by API-managed lifecycle,
- processes started manually from other terminals.

Active monitoring endpoint:

- GET /nodes/active

Response includes:

- countsbyrole,
- per-process PID and role grouping,
- flattened process views for UI refresh logic.

7) Operational Recommendations

Production-like usage guidance:

- use --list or --active before stop actions when diagnosing incidents,
- use --dry-run in scripted environments before enabling destructive mode,
- keep force-kill enabled for unattended automation, but record stop outcomes,
- verify no pending maintenance operation depends on node liveness before issuing --all.

For service wrappers or orchestrators, call programmatic functions directly and capture JSON-style result payloads in logs.

8) Policy Context

The stop utility itself does not change consensus policy. It supports safe operations around the active production constants:

- block cadence 120s,
- tx confirmations 10, private tx confirmations 10,
- coinbase maturity 150,
- fee floor 350 atoms/vB.

Reliable process lifecycle control is part of maintaining operational consistency for these policy-governed runtimes.

### Emissions Modeling Overview

Source: Docs/Emissions Modeling/Entire_Overview.md

**ENTIRE OVERVIEW - ATHO EMISSIONS MODELING**

Date: 2026-04-10 (UTC)

This document is a complete observation-based report of the active Atho emissions and burn model. It explains what the model computes, why the outputs look the way they do, and where practical limits show up under different utilization assumptions.

## Scope And Method

Scope includes consensus math, issuance behavior, burn mechanics, miner incentive surface, user fee costs, inflation/deflation transitions, floor clipping behavior, and long-horizon supply trajectories. The model evaluates no-burn and burn-enabled scenarios side-by-side under identical throughput assumptions to avoid framing bias.

Core constants used by the active model are: 120-second blocks, 262,800 blocks/year, 8 initial pre-tail eras of 1,000,000 blocks each, one-time bootstrap allocation of 390,625 ATHO at block 1, total pre-tail base target 100,000,000 ATHO, and tail reward 0.1953125 ATHO/block, fee floor 0.00000035 ATHO per policy byte (350 atoms/byte), max block policy budget 3,500,000 vbytes, burn share 45.0%, miner fee share 0.0%, consensus-pool routing share 55.0%, supply floor 21,000,000 ATHO, and hard max supply cap 150,000,000

Atho.

Consensus Alignment Status

PASS: Year-250 supply ordering NoBurn >= Burn25 >= Burn50 >= Burn75 >= Burn100.

PASS: noburn100 post-tail net change matches tailissuance - burnedfees.

PASS: burn25 post-tail net change matches tailissuance - burned_fees.

PASS: burn50 post-tail net change matches tailissuance - burned_fees.

PASS: burn75 post-tail net change matches tailissuance - burned_fees.

PASS: burn100 post-tail net change matches tailissuance - burned_fees.

PASS: Annual burn never exceeds annual fee pool.

PASS: Burn activation flips between year 30 (off) and 31 (on) in sampled yearly outputs.

Consensus alignment checks verify that the model schedule and policy math match live constants (tail start, fee floor, burn split, supply floor, and hard max cap). The same constants are enforced during block verification and persistence in the node, so model and chain policy remain coupled.

Emission Shape

Pre-tail emission is fixed by the consensus era sequence (50 -> 25 -> 12.5 -> 6.25 -> 3.125 -> 1.5625 -> 0.78125 -> 0.390625 -> tail 0.1953125 ATHO/block), producing an exact pre-tail issuance of 100,000,000.000 ATHO. Tail mode starts at height 8,000,000 (~year 30.44).

Tail annual issuance is 51,328.125 ATHO. At 100% policy-budget utilization, annual fee pool is 321,930.000 ATHO and max annual burn equals 144,868.500 ATHO.

Scenario Outcomes

Year-250 circulating supply ordering is strictly monotonic: NoBurn100=111,269,531.250 ATHO, Burn25=103,301,763.750, Burn50=95,333,996.250, Burn75=87,366,228.750, Burn100=79,398,461.250.

# References and Standards Context


- NIST Post-Quantum Cryptography Project (program overview, evaluation process, and standards-track context).
- NIST FIPS publications for post-quantum transition planning and cryptographic modernization guidance.
- Atho local Falcon/NIST companion package: Docs/falcon Docs.pdf.
- Falcon research paper family: Fast-Fourier lattice-based compact signatures over NTRU.
- CRYSTALS-Dilithium and SPHINCS+ literature (comparison context for post-quantum signature design tradeoffs).
- Shor, P. W. (1994): Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.
- Grover, L. K. (1996): Quantum mechanical algorithm for database search and implications for preimage security margins.
- Keccak / SHA-3 design rationale and sponge-function security analyses.
- Bitcoin whitepaper (Nakamoto): baseline PoW chain model and UTXO transaction framework.
- Bitcoin Core technical documentation (secp256k1, script, and network policy context).
- Litecoin protocol documentation (Bitcoin-derived consensus with different issuance/cadence parameters).
- Ethereum yellow-paper lineage and post-EIP fee-market behavior references.
- Academic and industry analyses on harvest-now, decrypt/forge-later strategic risk models.
- Operational security literature on software supply-chain integrity, binary provenance, and pinning controls.
- Database reliability references relevant to LMDB-style embedded-state operation and recovery practices.
- Distributed-systems references on deterministic serialization and consensus divergence failure modes.
- Atho repository documentation set: WhitePaper, Consensus, Falcon512, Sha3-384, Emissions, Threat, and modeling reports.
- Atho constants and code paths in Src/Utility/const.py, block/tx validators, and monitor modules.
- Atho emissions modeling reports and datasets for long-horizon utilization/supply scenario analysis.


# Explanatory Footnotes

These notes translate specialized terms into short plain-language definitions for faster reading and cross-team review.

[FN-01] Post-quantum means cryptography designed to remain secure even if practical quantum computers emerge.
[FN-02] Consensus-critical means nodes must agree on this rule; if not, chains can split.
[FN-03] UTXO means each spend references specific prior outputs rather than changing a global account balance field.
[FN-04] vsize is the virtual transaction size used for block capacity and fee policy after witness weighting.
[FN-05] Weight units count base bytes at 4x and witness bytes at 1x; vsize is roughly ceil(weight/4).
[FN-06] Tail issuance is a fixed long-run block reward after halving eras finish.
[FN-07] Fee burn means a configured portion of paid fees is permanently removed from spendable supply.
[FN-08] Runtime guard monitors critical files and policy state; in enforce mode it can stop unsafe operation.
[FN-09] Tighten-only policy means future updates can increase strictness but cannot silently relax core safety rules.
[FN-10] Rollback cap limits emergency reorg depth so recovery is bounded during incidents.
[FN-11] Binary pinning checks the exact digest of critical binaries before they are trusted.
[FN-12] Compact relay means peers send short IDs and request only missing transactions to reduce network payload.
[FN-13] Mempool is the set of valid unconfirmed transactions waiting for block inclusion.
[FN-14] Deterministic serialization means the same transaction always encodes to the same bytes everywhere.
[FN-15] Fail-closed means uncertain or unsafe states are rejected rather than accepted.
[FN-16] Canonical means only one valid encoding is accepted, preventing ambiguous interpretation.
[FN-17] Outpoint is a pointer to a previous tx output: tx_id plus output index.
[FN-18] Coin selection is the wallet strategy for choosing which UTXOs fund a transaction.
[FN-19] Propagation is how quickly blocks and txs travel between peers.
[FN-20] Orphan/stale block means a valid block that lost the race to another tip.
[FN-21] Tx index is a lookup map from tx_id to where it appears in chain storage.
[FN-22] Revision marker is a monotonic counter used to detect state changes cheaply.
[FN-23] Replay-safe means a transaction cannot be validly applied twice.
[FN-24] Byte-fee policy charges by serialized size rather than computational gas.
[FN-25] Tail phase is the era after halvings where reward becomes fixed.
[FN-26] Checkpoint interval is a bounded-history anchor used in rollback policy.
[FN-27] Supply floor is the lower bound under which net burning is clipped.
[FN-28] Regression tests are repeatable tests proving behavior stayed correct after changes.

# Keyword Index

Deduplicated index of key terms used in this whitepaper.

| Keyword | Meaning |
|---|---|
| API Authentication | Scope-based access controls for read/write/admin operations. |
| Base56 | Human-facing address encoding with checksum discipline. |
| Binary Pinning | Digest-based trust gate for cryptographic binaries. |
| Block Weight | Consensus capacity metric measured in weight units. |
| Burn Policy | Configured fee burn behavior and supply impact. |
| Canonical Encoding | Single accepted representation for wire/signature fields. |
| Consensus Version | Protocol rule-set identifier for node compatibility. |
| Deflation Threshold | Utilization level where annual burn exceeds tail issuance. |
| Deterministic Serialization | Byte-stable encoding used for consensus-critical hashing. |
| Emission Schedule | Block reward path from genesis eras into tail phase. |
| Falcon-512 | Post-quantum signature scheme used for transaction authenticity. |
| Fee Floor | Minimum fee policy required for transaction admission. |
| Floor Supply | Configured lower bound for circulating supply clipping logic. |
| Grover Model | Quantum search model affecting effective symmetric/hash margins. |
| Hashrate | Aggregate PoW compute rate observed across miners. |
| LMDB | Embedded storage engine for chain, UTXO, and mempool state. |
| Mempool | Set of valid pending transactions not yet mined. |
| Merkle Root | Block commitment over included transaction IDs. |
| Post-Quantum | Security posture designed for future quantum adversaries. |
| Runtime Guard | Startup/runtime integrity checks for critical artifacts. |
| SegWit | Witness separation and weighted sizing model for transactions. |
| Shor Model | Quantum algorithmic model threatening discrete-log systems. |
| Tail Issuance | Long-run fixed reward phase after halving periods. |
| Tighten-Only Rules | Upgrade policy that allows stricter but not weaker safety constraints. |
| TPS | Transactions-per-second capacity estimate under given block and tx sizing. |
| UTXO | Unspent transaction output model for ownership and spending. |
| vsize | Virtual size used for fees and block fit decisions. |
| Witness | Signature/public-key data validated with discounted weight rules. |

# Document Integrity Notes

This PDF is generated from live repository state. Regenerate after any consensus, security, or economic policy changes to maintain alignment [FN-08][FN-28].

# v3 Technical Release Addendum

This addendum page is included to preserve explicit release traceability for the v3 whitepaper artifact. It records that the document was regenerated from active repository state after synchronized updates to consensus constants, emissions model outputs, and technical/operational documentation sources.

For external review, validate this artifact by checking the active policy profile (120-second target cadence, 180-block retarget window, 10 standard/private confirmation policy, 150-block coinbase maturity, 350 atoms/vB fee floor, and post-tail 55% pool routing), then regenerate the linked emissions reports and confirm output coherence with this technical narrative.

This page intentionally exists as a stable release marker so future audits can distinguish major synchronized whitepaper revisions from minor incremental rebuilds.