ATHO

Atho — The Platinum Standard of the Quantum Age.

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

Generated: 2026-03-17
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

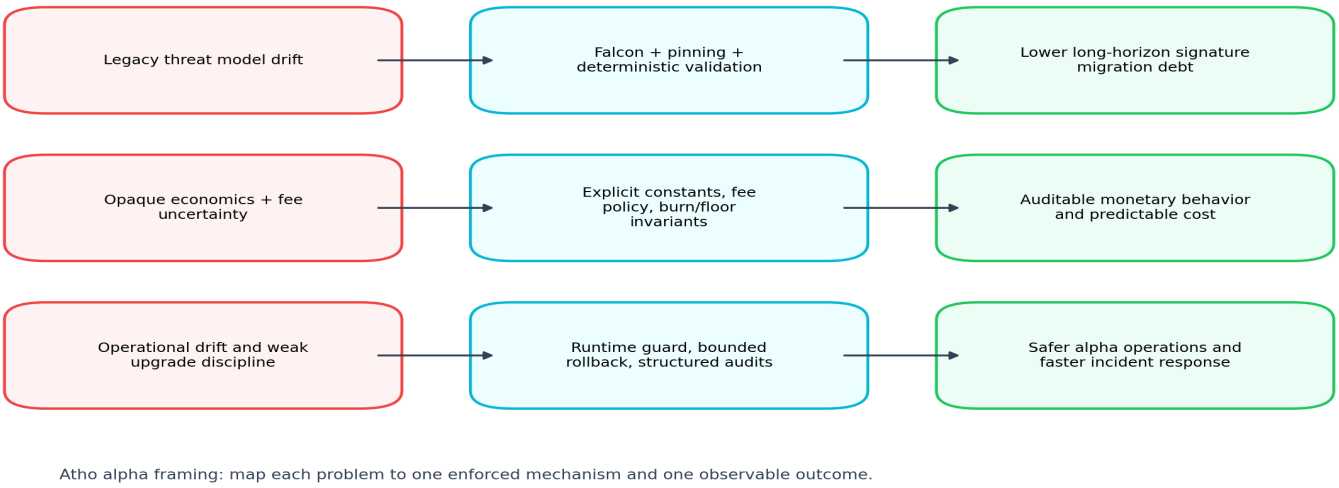


Figure 3B: Atho alpha problem-to-mechanism mapping with operator-visible outcomes.

Quantum Intuition Visuals

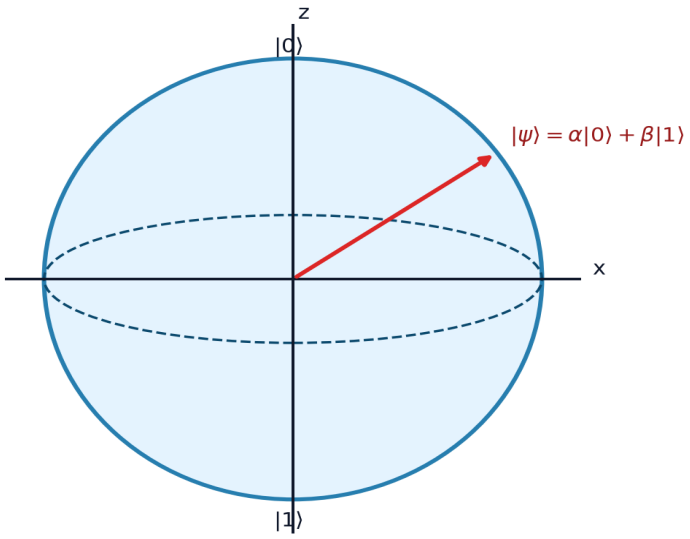


Figure 1: Stylized Bloch-sphere view of one qubit state. This is a conceptual guide, not hardware geometry.

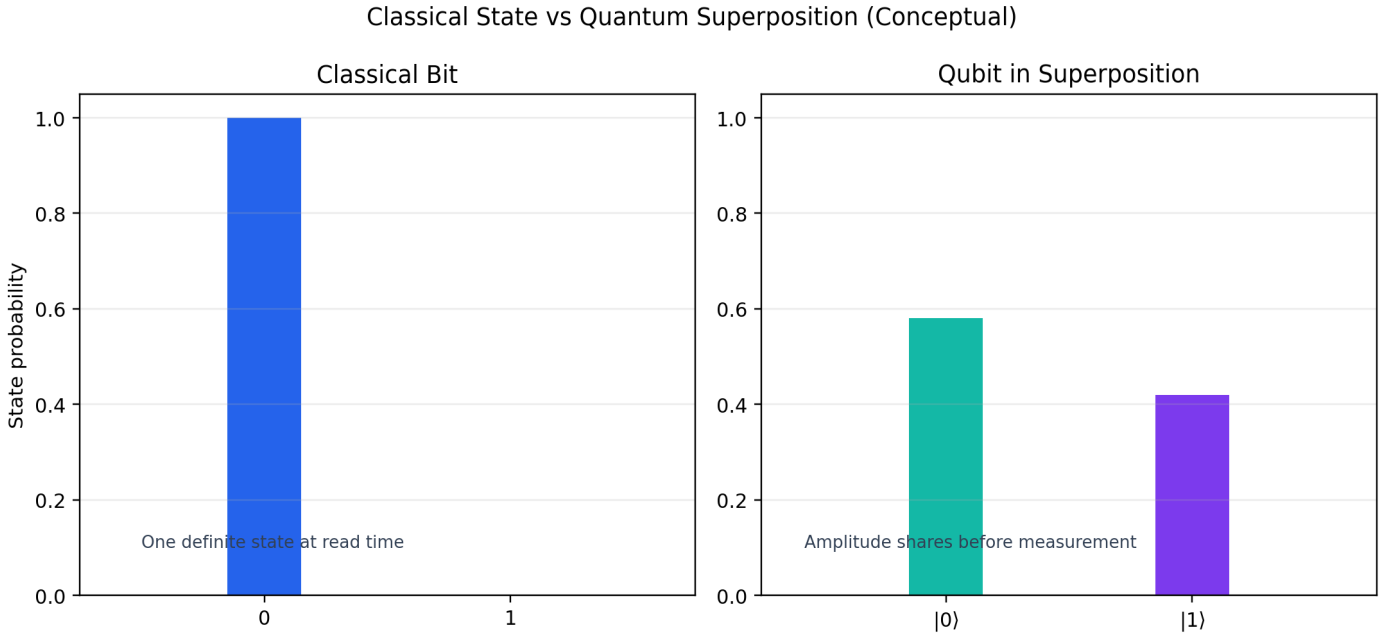


Figure 2: Classical bit states versus qubit superposition probabilities before measurement.

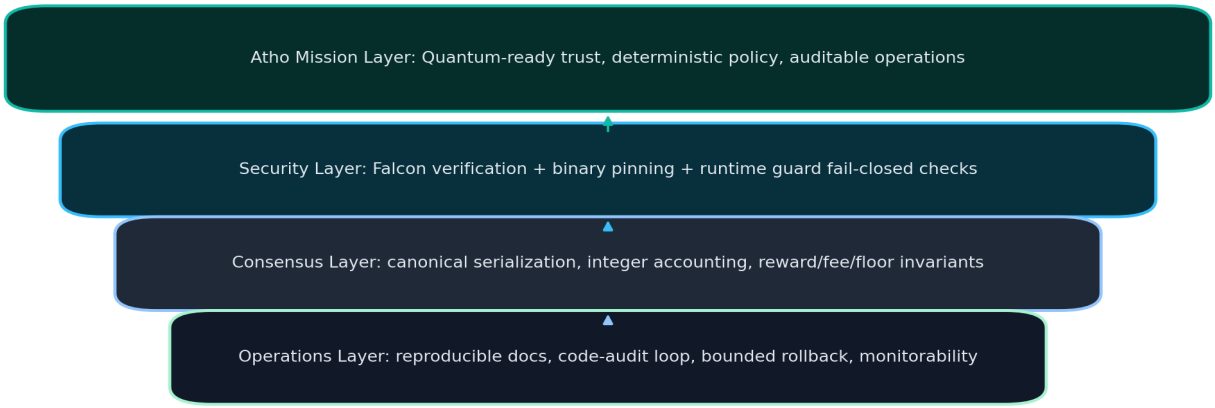


Figure 3: Atho-focused architecture priorities for the alpha pre-release whitepaper.

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.

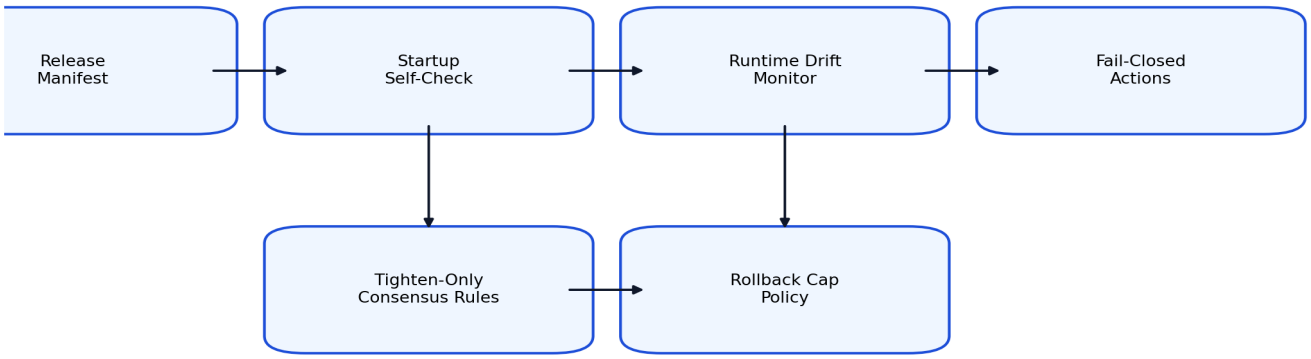
Research-grounded design means translating external findings into explicit controls. For signatures, this means selecting and operationalizing post-quantum-compatible pathways.

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
Blocks Per Era	1,314,000
Tail Start Height	21,102,000
Tail Reward	0.25000000 ATHO/block
Pre-Tail Supply	30,000,000.000 ATHO
Supply Floor	21,000,000 ATHO
Fee Floor	200 atoms/byte (0.00000020 ATHO/byte)
Minimum Tx Fee	187,500 atoms (0.000188 ATHO)
Max Block Size	2,500,000 bytes
Max Block Weight	10,000,000
Annual Tail Issuance	65,700.000 ATHO/year
Max Annual Fee Pool (100% Utilization)	131,400.000 ATHO/year
Deflation Threshold Utilization	50.000%

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	1,371	5,483	0.000246780	15.20
3	2	1,646	6,583	0.000296280	12.66
16	2	3,428	13,711	0.000617040	6.08
31	2	5,483	21,931	0.000986940	3.80

Protocol Mechanics Illustrations

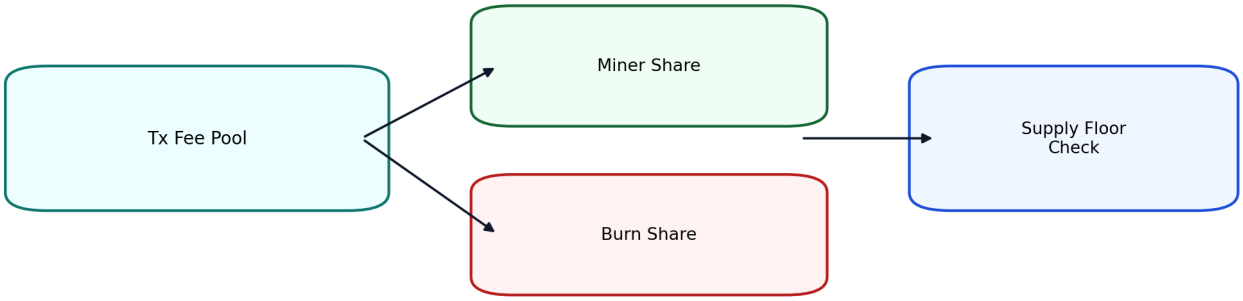
These diagrams show concrete mechanics that are easy to miss in text-only explanations: verification path, UTXO transformation, fee burn, relay, and persistence flow [FN-17][FN-19][FN-23].

Goal: one canonical verification path so mempool and block validation agree on exactly the same rules.

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

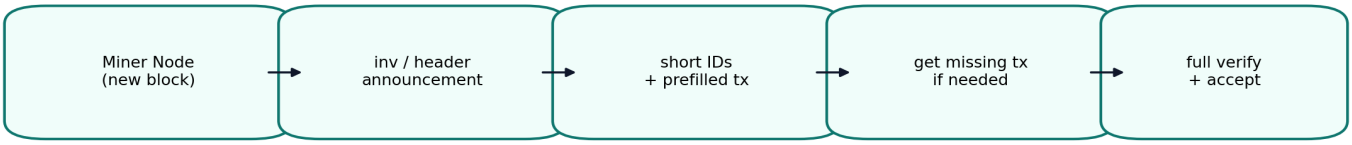
In plain terms: old coins are consumed once, then replaced by new coins.

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].

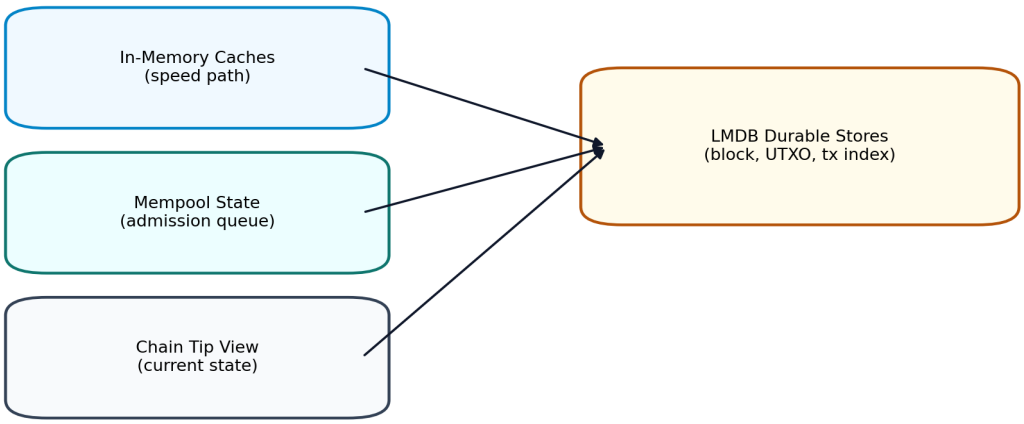


Figure 10: RAM acceleration layered on top of LMDB durability and restart safety [FN-22].

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.

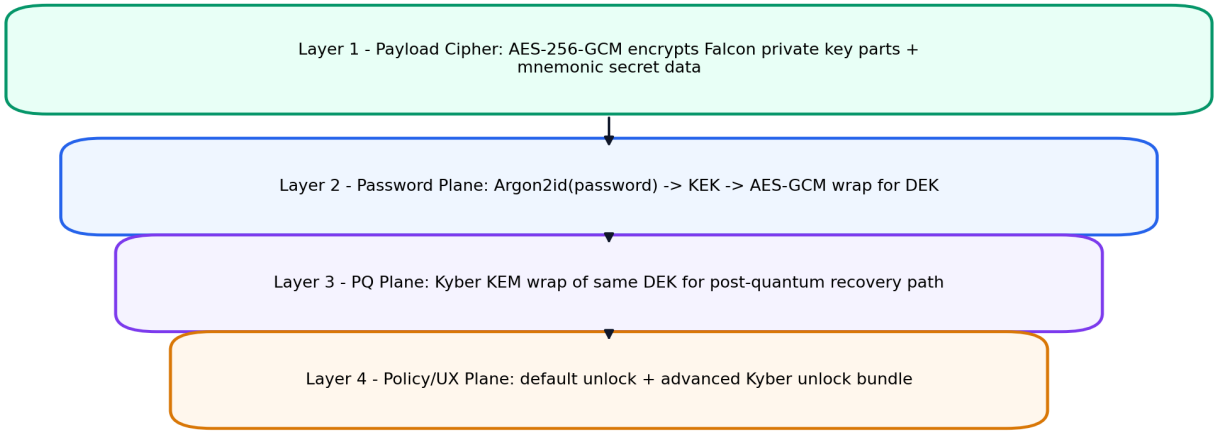
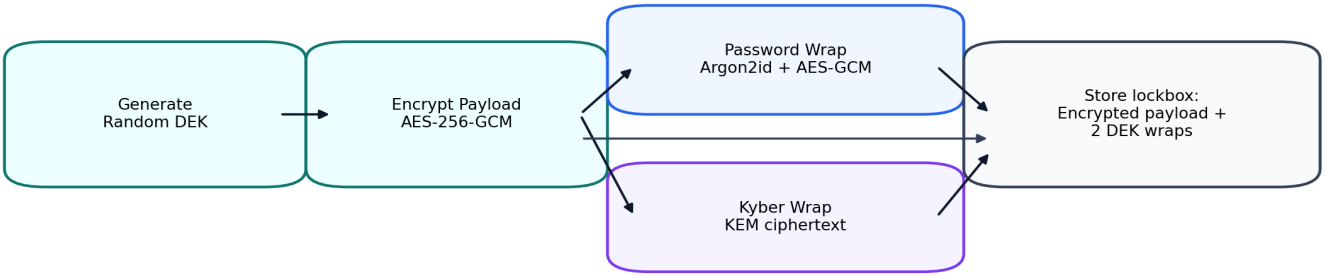
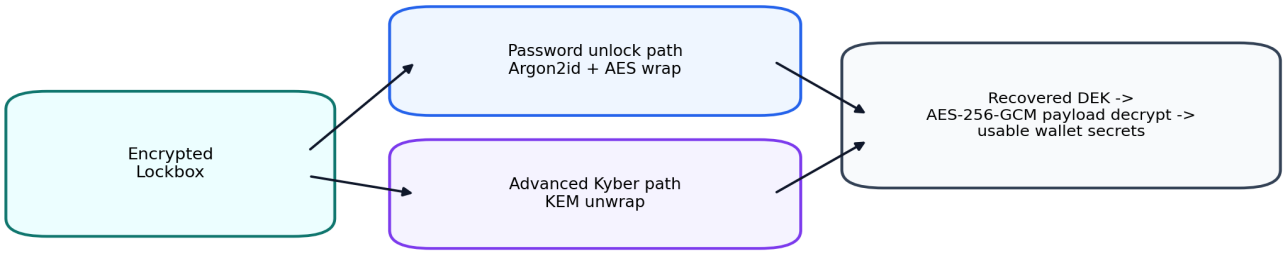


Figure 10A: Layered Atho key-at-rest stack showing AES payload encryption with password and Kyber DEK protection planes.



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

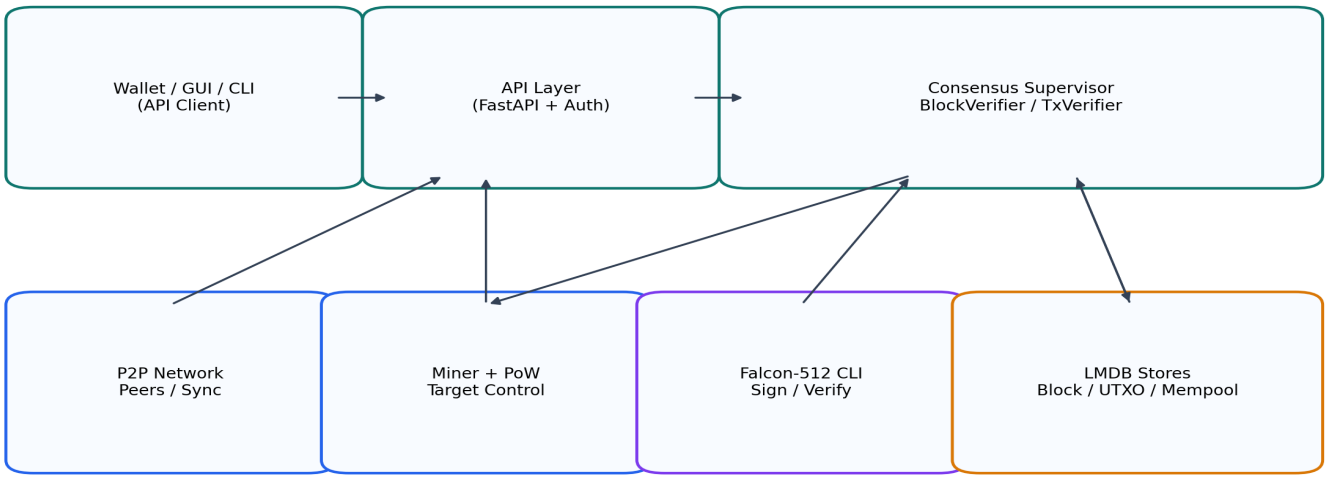


Figure: Atho runtime architecture from client entry points through consensus, cryptography, and storage.

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

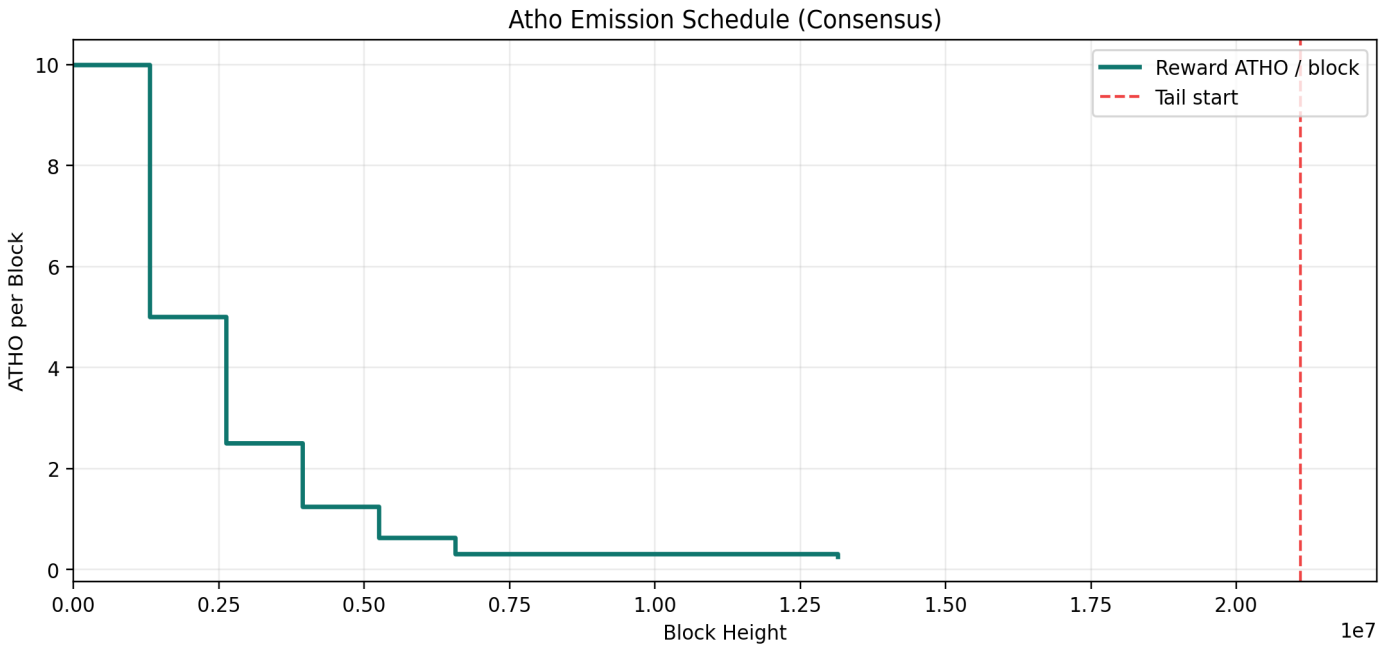


Figure 12: Consensus reward schedule with pre-tail eras and tail transition [FN-06][FN-25].

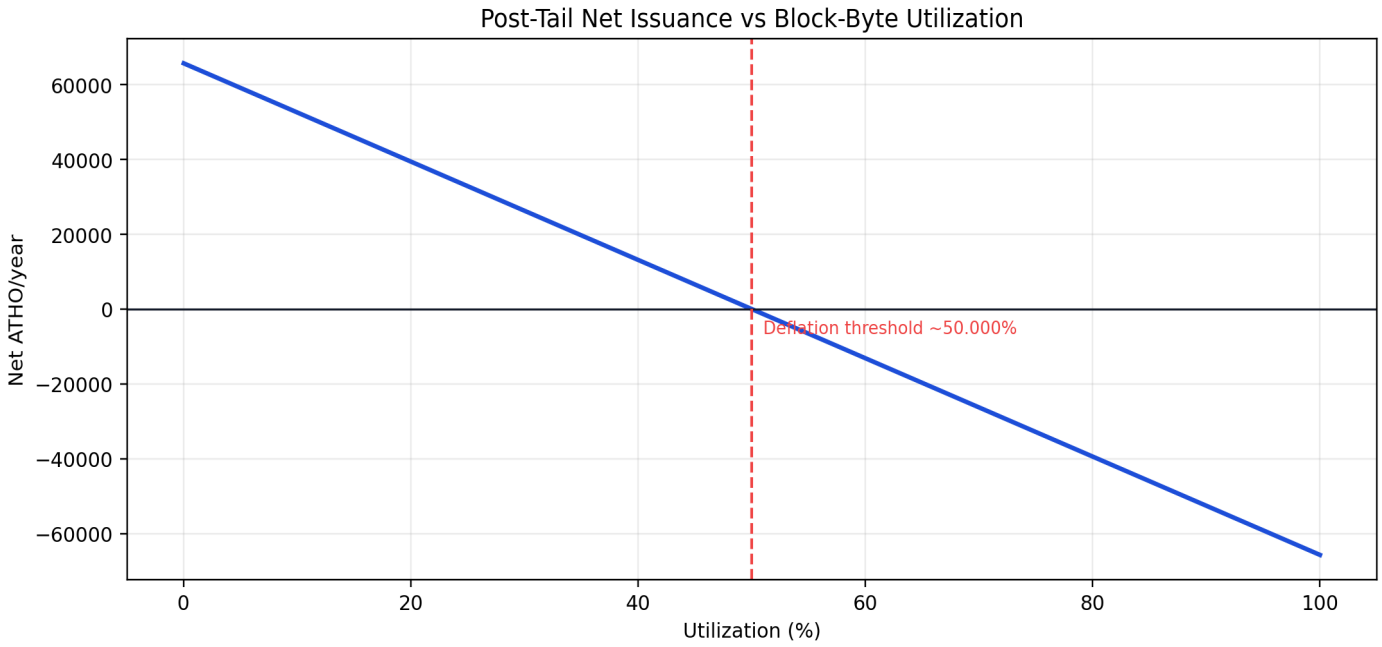


Figure 13: Net issuance response to block-byte utilization under active fee policy [FN-07].

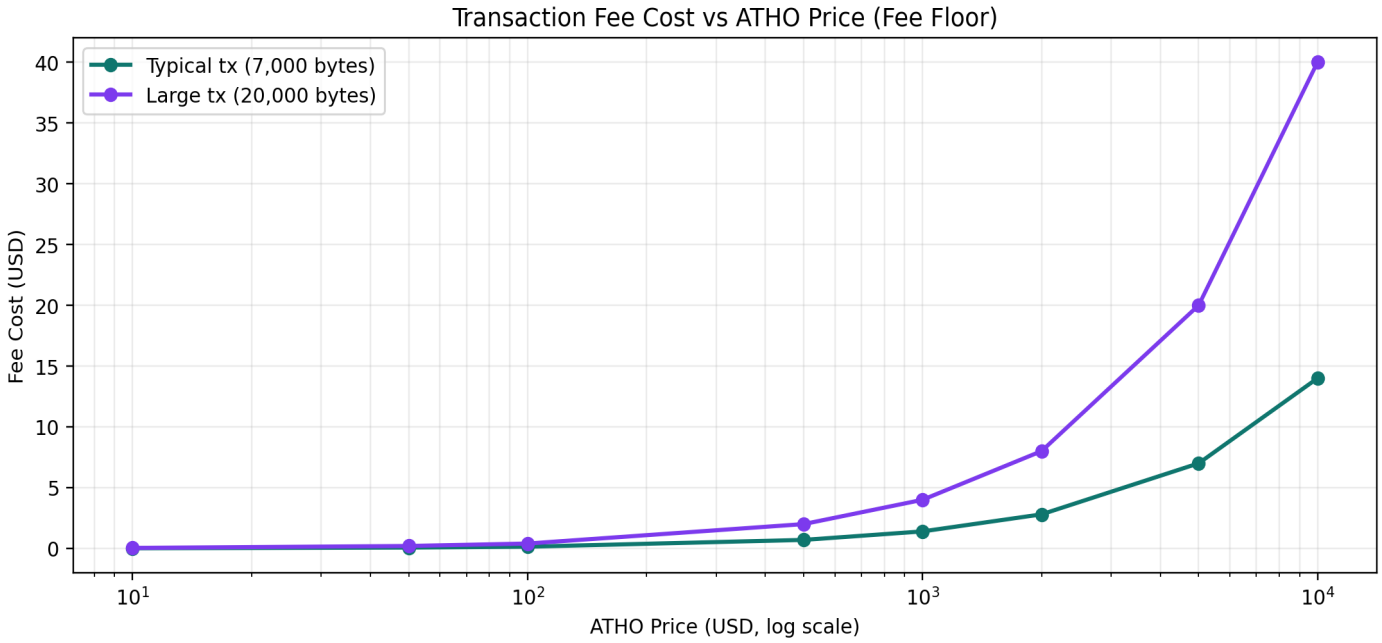


Figure 14: Fee cost profile for representative transaction sizes across ATHO price levels [FN-24].

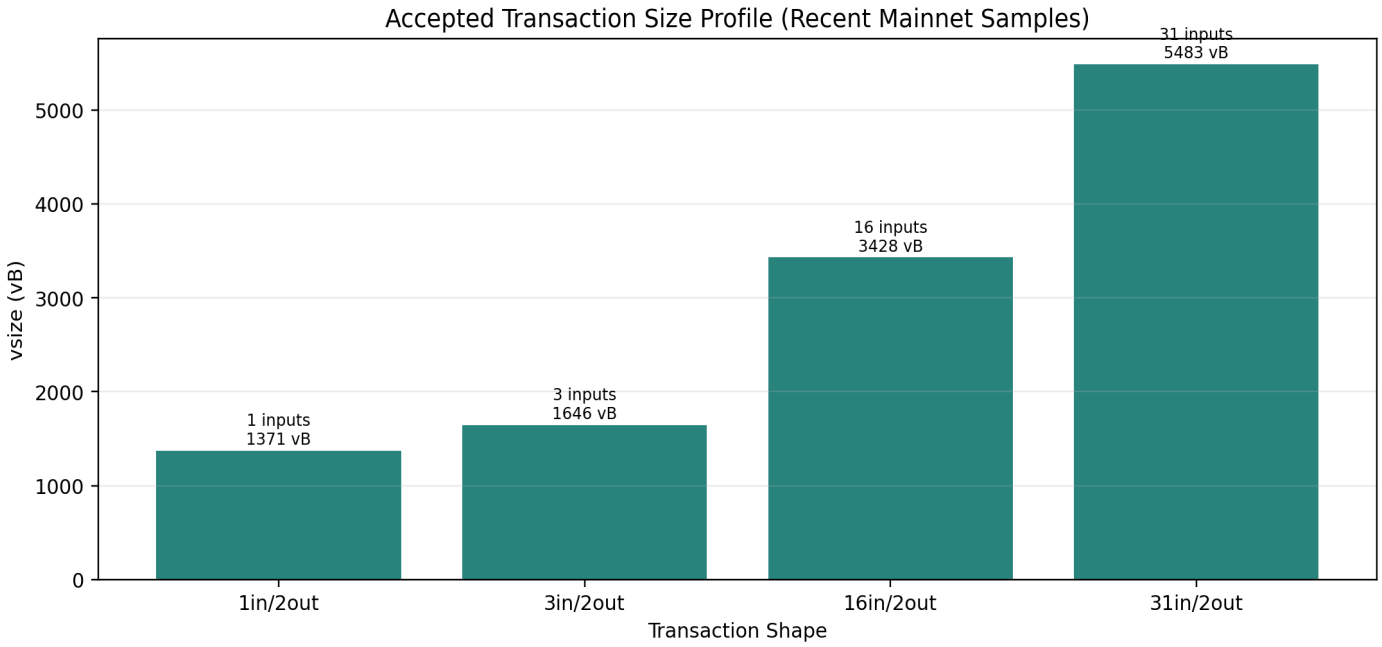


Figure 15: Accepted transaction vsize profile from recent mainnet samples (optimized wire format era) [FN-04].

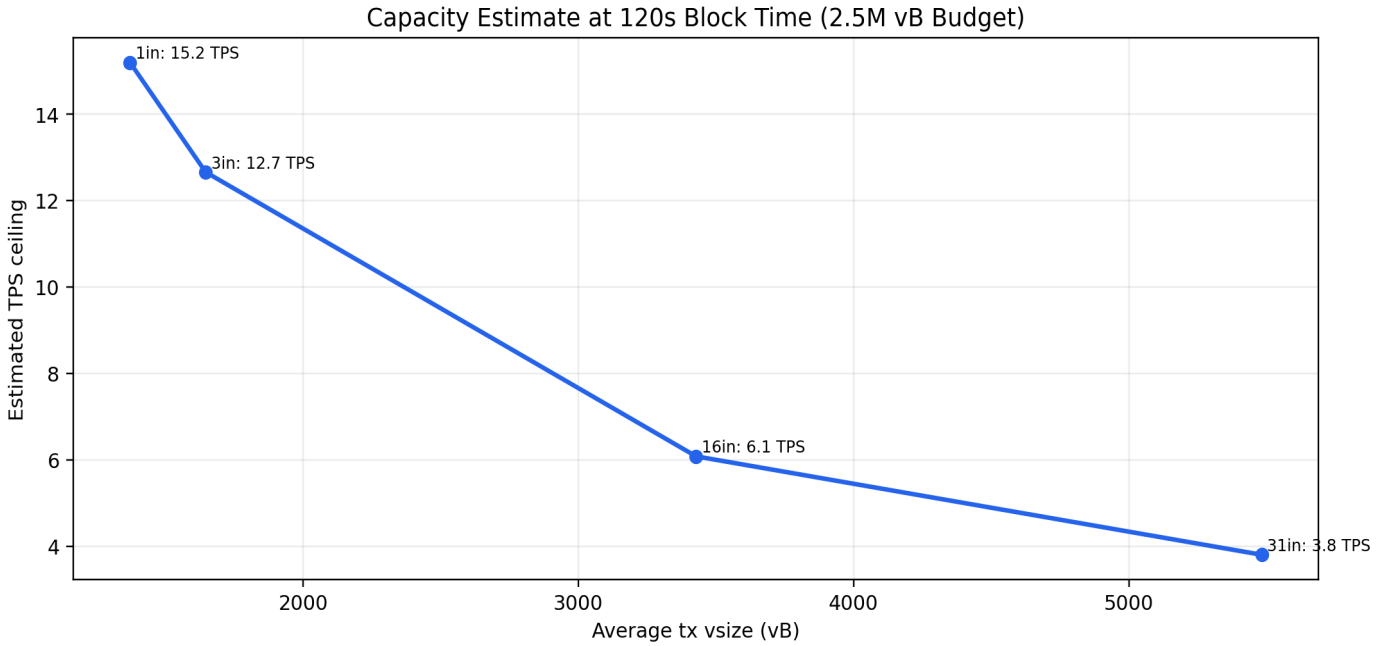


Figure 16: Estimated TPS envelope at 120-second blocks as a function of average transaction vsize.

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].

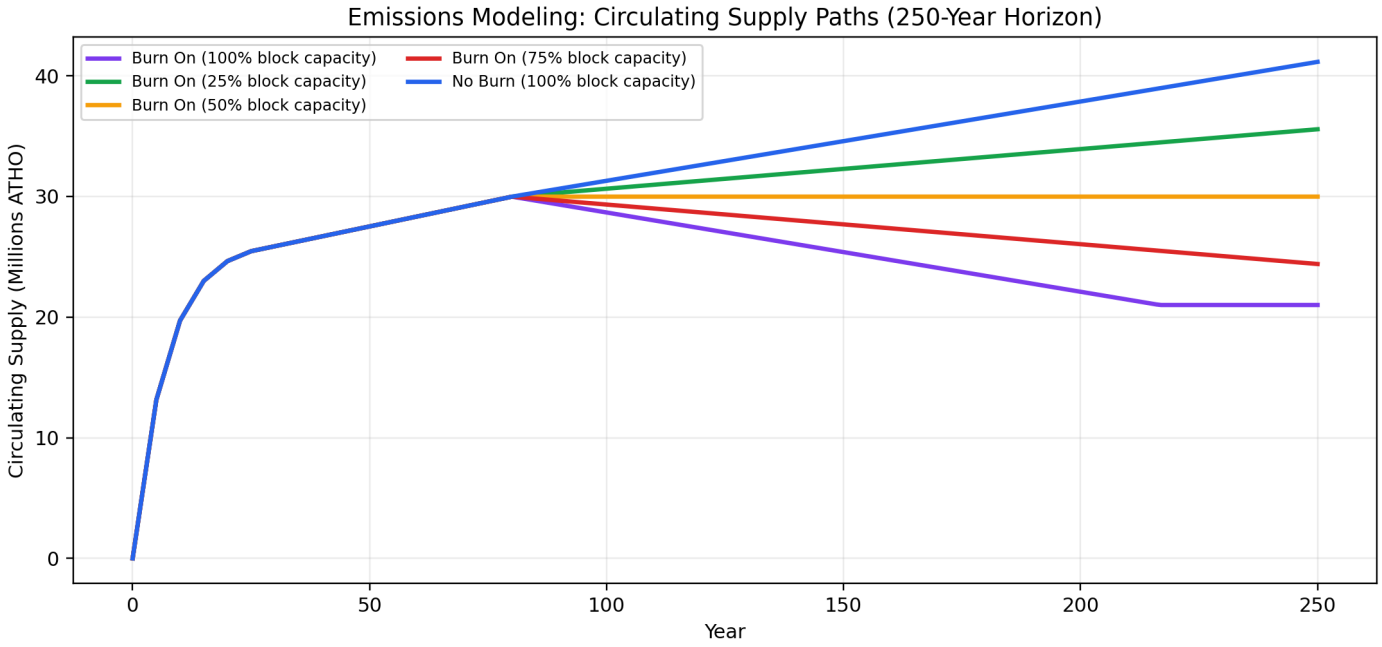


Figure 17: 250-year circulating supply trajectories by scenario (no burn and burn-at-capacity levels).

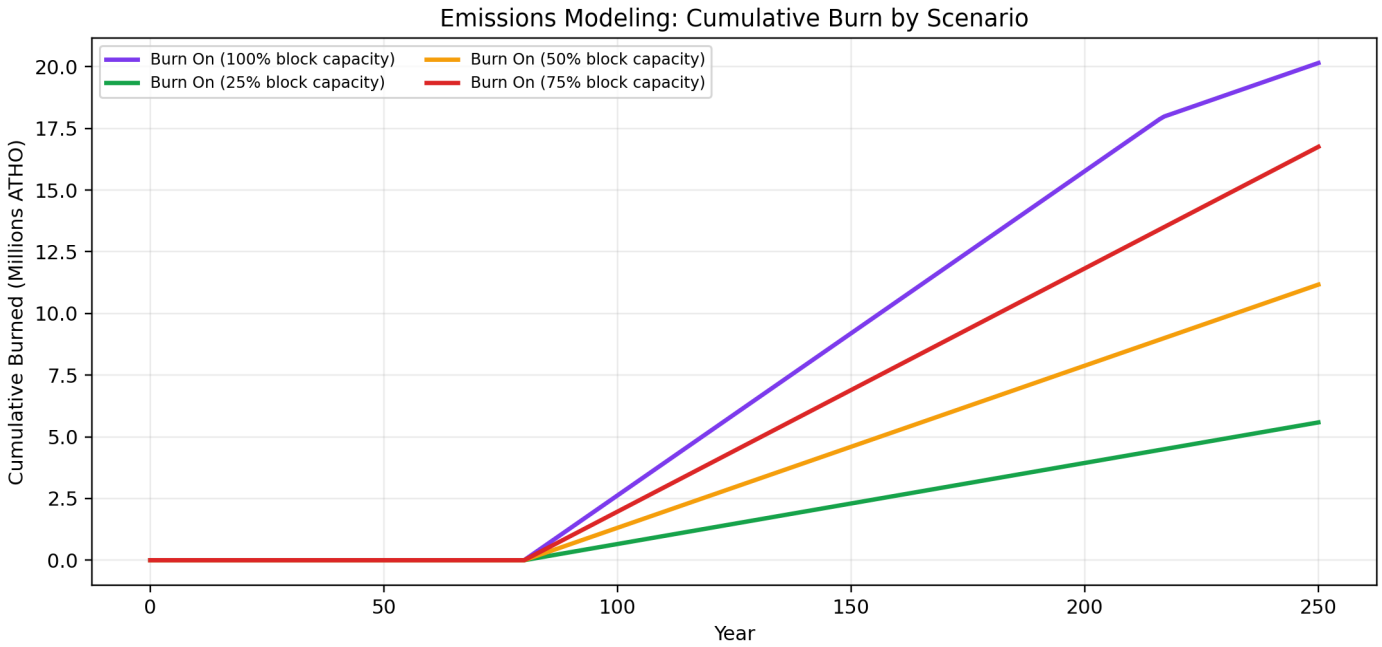


Figure 18: Cumulative burned ATHO over time for burn-enabled scenarios, showing monotonic divergence by utilization.

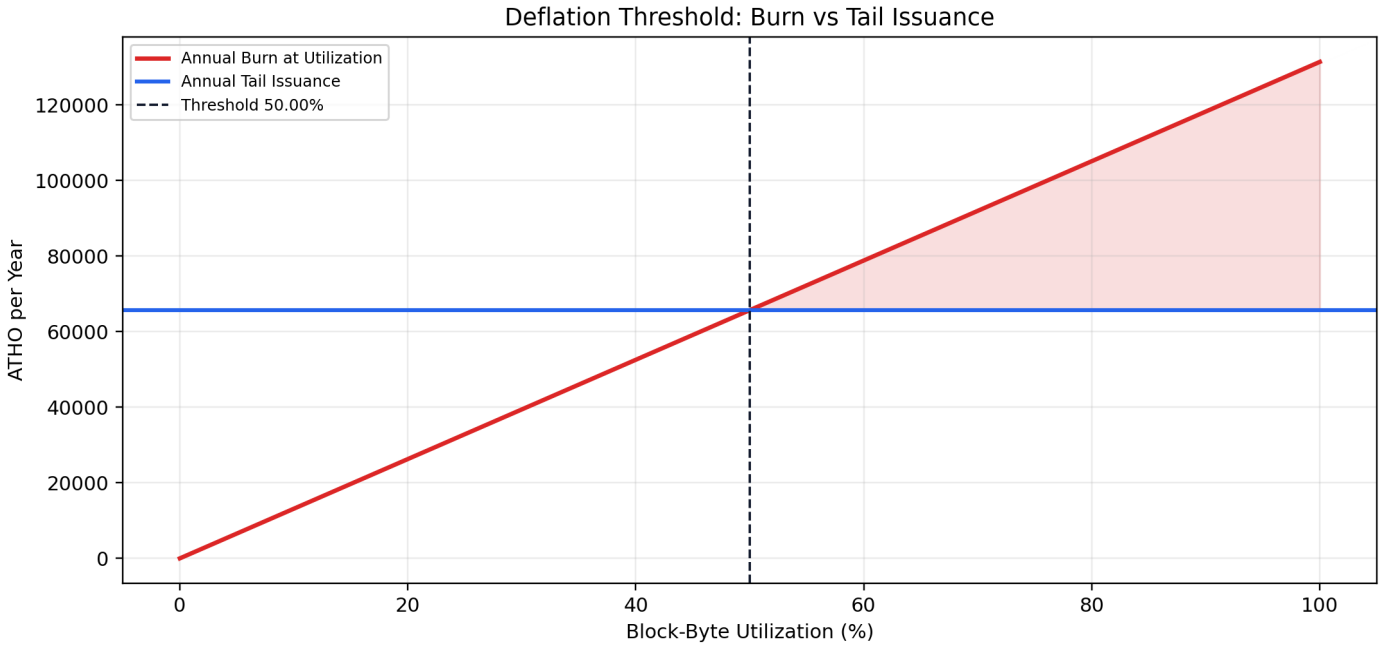


Figure 19: Deflation threshold as the intersection of annual burn and annual tail issuance.

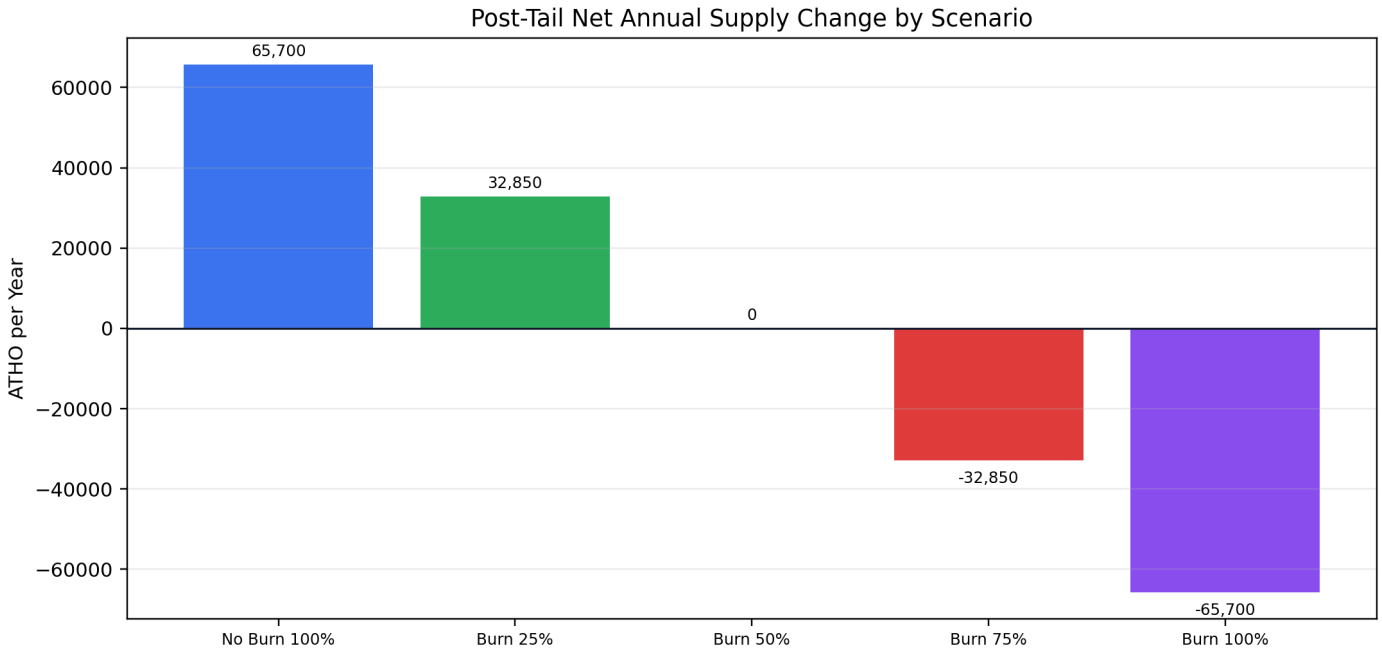


Figure 20: Post-tail net annual supply change by scenario (positive inflationary vs negative deflationary ranges).

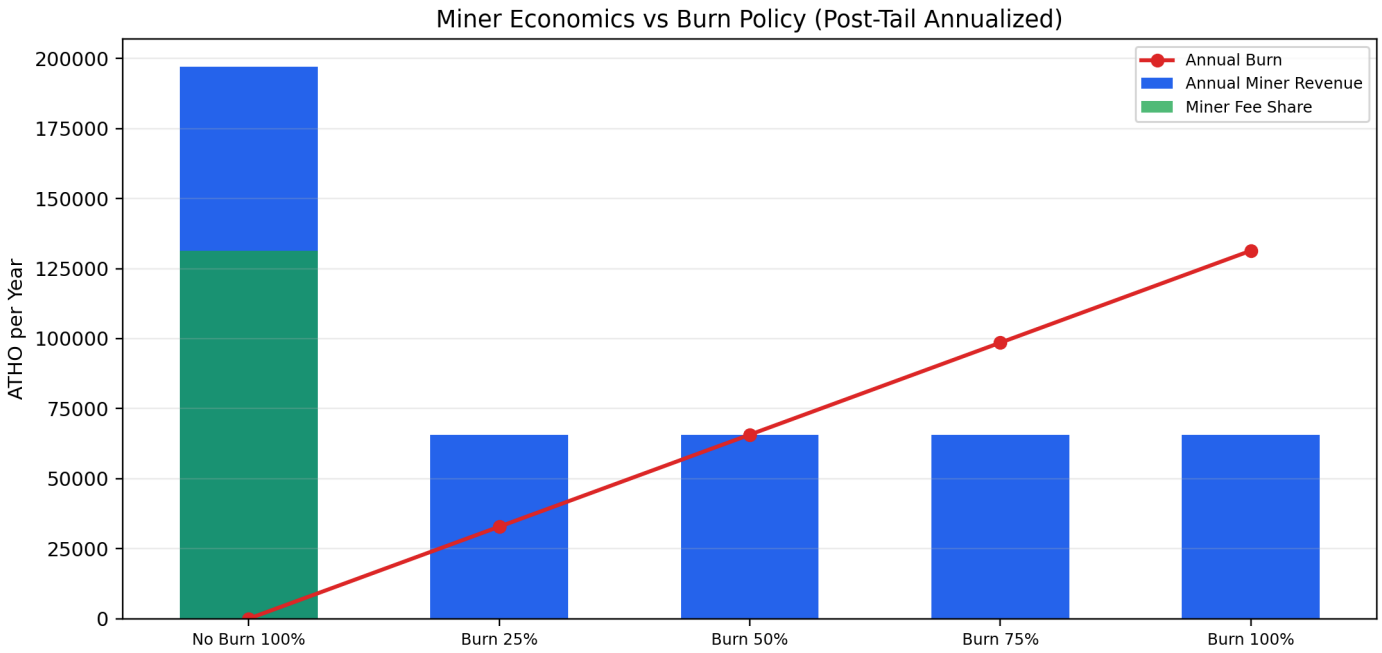


Figure 21: Miner economics comparison across scenarios, including annual miner totals, miner fee share, and annual burn.

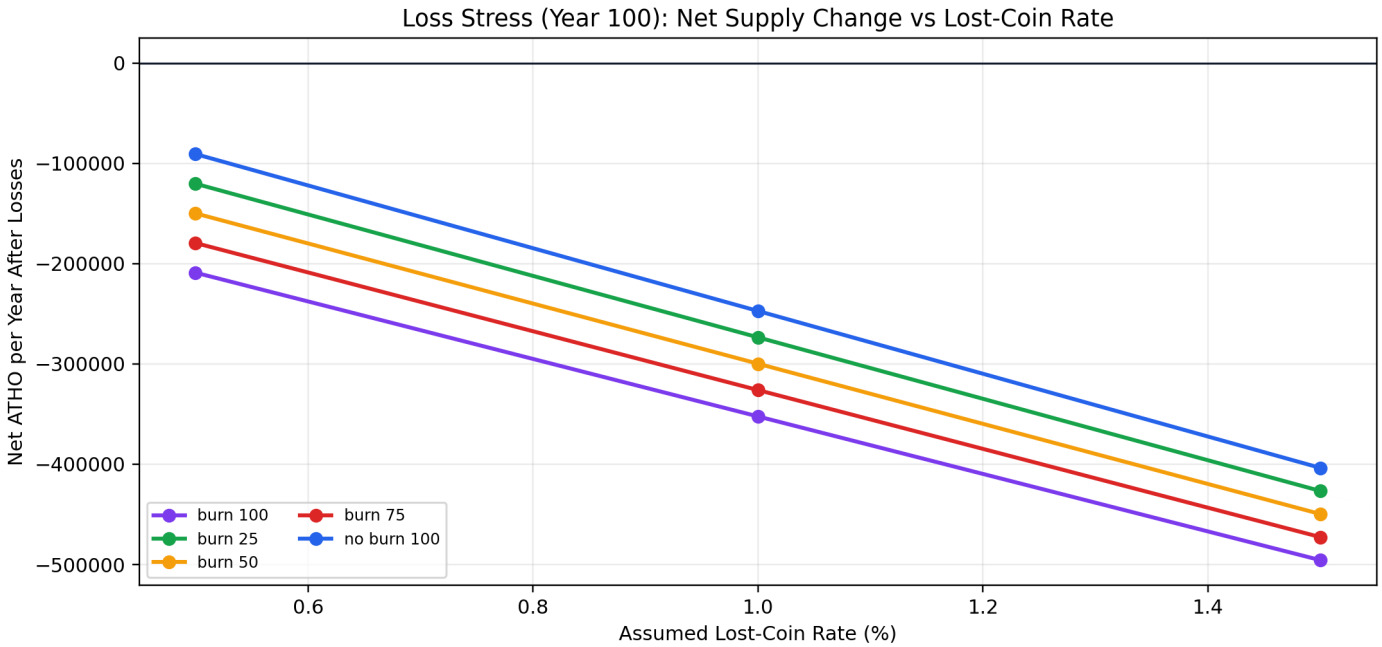


Figure 22: Lost-coin stress curves at year 100 showing when protocol net issuance is offset by loss assumptions.

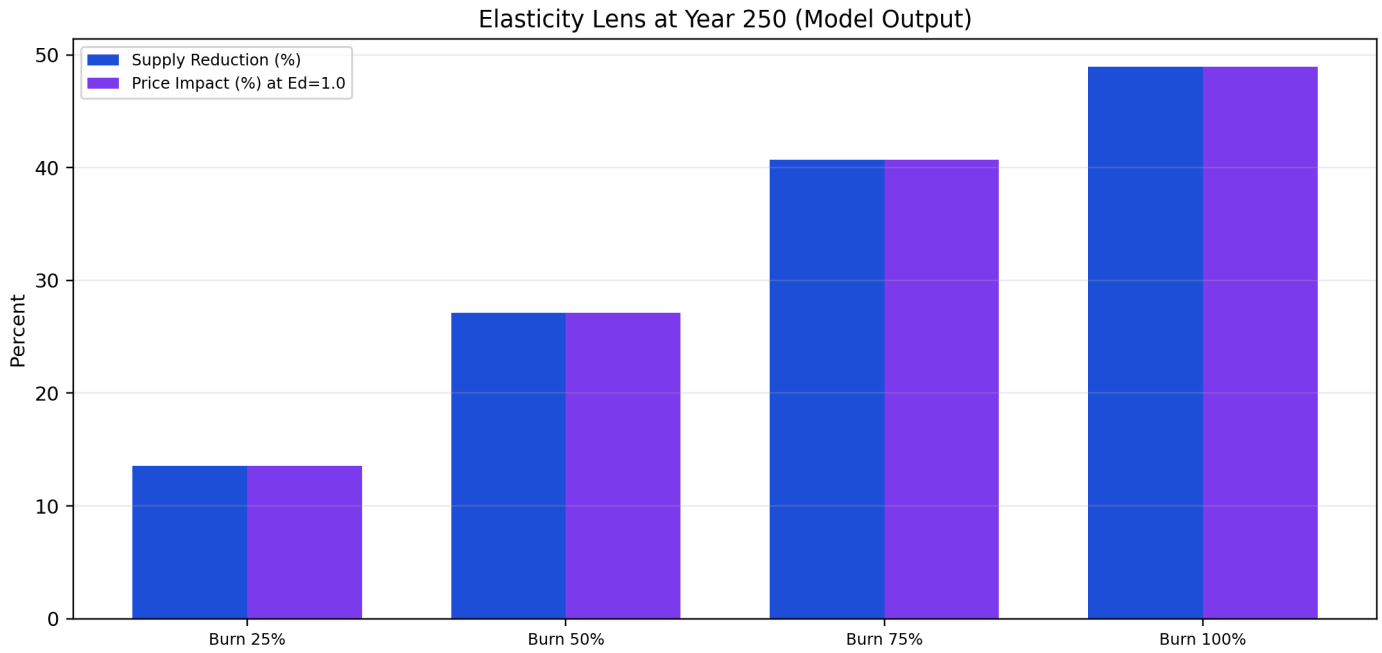


Figure 23: Year-250 elasticity lens comparing supply reduction with modeled price impact at elasticity 1.0.

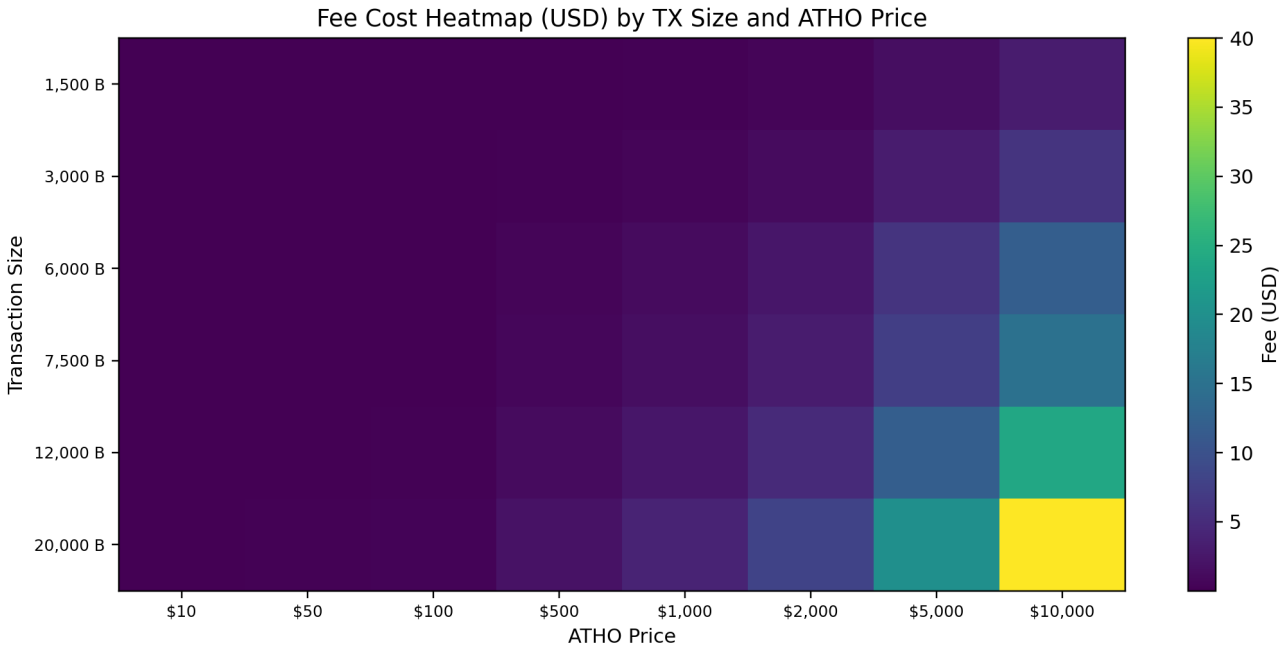


Figure 24: Fee-cost heatmap by transaction size and ATHO spot price under active byte-fee policy.

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	**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.

#	Question	Answer
3	**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.
4	**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.
5	**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.
6	**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.
7	**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.
8	**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.
9	**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.
10	**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.
11	**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.
12	**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.
13	**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.
14	**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.
15	**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.
16	**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.
17	**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.
18	**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.
19	**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.
20	**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.
21	**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.
22	**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.
23	**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.
24	**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.

#	Question	Answer
25	**What does 'confirmed' mean versus 'pending'?**	Pending means accepted by policy and waiting for block inclusion. Confirmed means included in an accepted chain block.
26	**Why can pending status sometimes last longer than expected?**	Block timing, fee competitiveness, input complexity, and mempool conditions all affect inclusion speed.
27	**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.
28	**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.
29	**Does making transactions smaller reduce security?**	Not if reductions remove redundant serialization only. Security-critical data and verification checks remain mandatory.
30	**Does compact relay mean nodes trust partial data?**	No. Compact relay improves transport efficiency, but full required data is fetched before final validation.
31	**Can a peer lie about telemetry like hashrate?**	Telemetry can be spoofed unless attested. Atho treats telemetry as operational signal, not consensus truth.
32	**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.
33	**How do upgrades avoid weakening security?**	The tighten-only approach prevents silent safety relaxation. Upgrades should add checks or clarity, not remove core protections.
34	**What is the practical purpose of signed release manifests?**	They let operators verify that release artifacts match approved metadata before trusting them in production.
35	**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.
36	**Why are rollback limits important in incidents?**	They bound recovery actions so crisis response stays controlled, auditable, and resistant to ad-hoc overreach.
37	**Can rollback policy be unlimited for flexibility?**	Unlimited rollback increases abuse and uncertainty risk. Bounded rollback is safer for operator coordination and user trust.
38	**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.
39	**What does the supply floor protect against?**	It prevents excessive deflation from driving spendable supply below a configured lower bound.
40	**Can fee policy change over time?**	Yes, through explicit governance and versioned updates. Changes should remain auditable and consistent across nodes.
41	**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.
42	**How should exchanges evaluate readiness?**	Check deterministic validation behavior, upgrade discipline, rollback policy, monitoring coverage, and incident-response clarity.
43	**How should institutional partners evaluate readiness?**	Use an evidence checklist: cryptographic posture, consensus invariants, artifact provenance, operational controls, and reproducible testing.
44	**Is Atho private by default?**	It is a transparent ledger model. Privacy expectations should match that transparency unless additional privacy layers are explicitly deployed.
45	**What data should operators monitor daily?**	Chain tip progress, mempool pressure, rejected block reasons, validation error rates, storage health, and security-log anomalies.
46	**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.

#	Question	Answer
47	**What is the value of tx indexing for performance?**	It avoids expensive historical scans during candidate building and verification lookups, reducing latency under load.
48	**What is the value of mempool revision markers?**	They let readers refresh state only when needed, reducing repeated rebuilds and lock contention.

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].

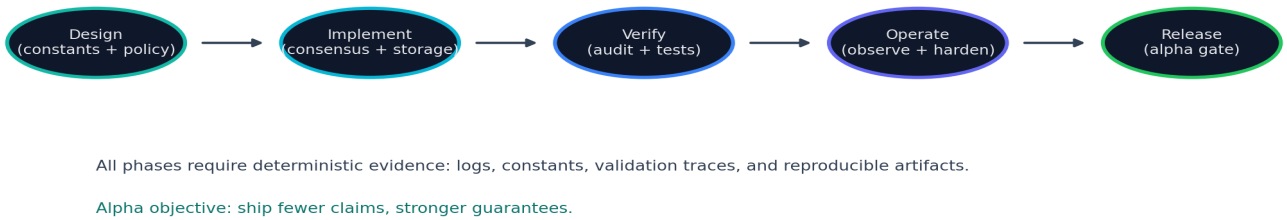


Figure 25: Alpha delivery flow from design to enforceable release controls.

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	2,707	Repository structure, component map, and operator-facing entry points.
Docs/README.md	182	Documentation index and authoritative navigation.
Docs/ONBOARDING.md	699	Developer Onboarding: protocol/operations reference.
Docs/quickstart.md	1,528	Operational startup, run commands, and baseline deployment steps.
Docs/WhitePaper.md	1,004	Technical overview and protocol summary.
Docs/Consensus.md	1,676	Block/tx acceptance, policy gates, and reorg behavior.
Docs/Falcon512.md	1,080	Falcon integration details and verification posture.
Docs/Sha3-384.md	791	Hashing rationale, canonicalization, and checksum rules.
Docs/Tx.md	582	Transaction model, fee policy, and wire/validation behavior.
Docs/Sigwit.md	408	Witness handling, weight/vsize accounting, and policy integration.
Docs/ApiAuth.md	938	API key scopes, auth controls, and security layering.
Docs/LMDB.md	1,183	Persistence model, map sizing, and recovery/operational constraints.
Docs/Base56.md	748	Address encoding, checksum behavior, and normalization rules.
Docs/KeyManager.md	1,397	Key lifecycle and signing/verification management.
Docs/Emissions.md	441	Issuance model, burn/floor policy, and emission audits.
Docs/Inflation_Deflationary.md	619	Long-horizon utilization and supply scenarios.
Docs/Threat.md	2,136	Security threat model and mitigation priorities.
Docs/Docker.md	954	Containerized deployment guidance and operational setup.
Docs/dockertest.md	935	Docker test harness and verification routines.
Docs/Binaries.md	776	Binary build/release provenance and constraints.
Docs/Troubleshooting.md	2,000	Failure-mode diagnostics and operator playbooks.
Docs/Node_Stop.md	422	Node process lifecycle control utility.
Docs/Emissions Modeling/Entire_Overview.md	2,839	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 (Alpha Release) !Version

Status: Alpha documentation snapshot (2026-03-12).

Simple Quick Start (Recommended Order)

If you want the fastest setup with the least troubleshooting, use macOS or Linux. Windows works, but typically needs more toolchain setup.

System requirements

- CPU: Intel i3 (or equivalent) or better
- RAM: 4-8 GB minimum
- Storage: 120 GB SSD recommended
- Graphics/runtime: OpenGL 2.0 support (for GUI/Kivy)

1) Download code + create virtual environment + install requirements

macOS / Linux:

Windows (PowerShell):

If requirements.txt install fails (lmdb, kivy, cryptography), see Docs/Troubleshooting.md -> Dependency install issues (venv + pip).

2) Build/install Falcon binary

Runtime uses platform-tagged Falcon binaries from Src/Falcon/Binaries/<platform_tag>/.

Register/install current host binary:

If Falcon is missing for your platform, build it first (compiler required), then rerun install:

- macOS: clang (Xcode Command Line Tools)
- Linux: gcc/clang (build-essential)
- Windows: MSYS2 MinGW-w64 (gcc) or Visual Studio Build Tools (cl)

Falcon compile commands (full flags):

macOS:

Linux:

Windows (MSYS2 MinGW-w64):

3) Build Kyber shared library (wallet PQ unlock path)

Kyber can auto-build when needed, but you can prebuild by running any wallet key action after deps are installed.

Manual Kyber build requirement:

- C compiler available (cc, clang, or gcc)
- Output path used by runtime: Src/Kyber/ref/lib/

Kyber compile commands (full flags, Kyber-1024 / KYBER_K=4):

macOS:

Linux:

Windows (MSYS2 MinGW-w64):

4) Open Key Manager, create keys, encrypt if desired

- You can do this in CLI (Src/GUI/cliui.py) or GUI.
- CLI is good for first-time guided flow (wallet new, wallet recover, wallet export, etc.).

5) Open GUI and set API auth

Run GUI:

- macOS: ./run_gui.command
- Linux/macOS shell: ./run_gui.sh
- Windows: run_gui.bat

In GUI -> Settings:

- set API key
- set username/password you created in Src/Api/auth.py

6) Start nodes (Node tab in GUI)

- Start default nodes first (full + wallet is enough to participate).
- Miner node is optional and needs extra compute.

If nodes do not start:

- Run ./.venv/bin/python Src/Main/stop.py --active (or --list) to inspect running PIDs.
- Stop all stuck nodes:
- ./.venv/bin/python Src/Main/stop.py --all
- Start again from GUI Node tab.

You can also start manually:

- ./.venv/bin/python Src/Main/runnode.py
- ./.venv/bin/python Src/Main/join.py

6.5) Request Inbox (GUI)

- Request inbox UI is currently disabled by default in the alpha desktop build for stability.
- If you enable it in code/runtime, keep the send safety flow:

- open request inbox,
- choose a request,
- copy requester address,
- paste the same address in the verify field,
- continue to prefilled send modal (address + amount).
- Legacy bell/notification-center flow is not part of the stable default path.

Docs Index

Source: Docs/README.md

Atho Documentation Index

Status: Alpha documentation snapshot (2026-03-12).

Use this as the entry point for all docs in Docs/.

Start here

- quickstart.md – fastest node + wallet + CLI startup flow.
- Troubleshooting.md – common install/start/sync failures.
- ONBOARDING.md – developer setup and codebase orientation.

Protocol and architecture

- WhitePaper.md
- Consensus.md
- Tx.md
- Sigwit.md
- Emissions.md
- Inflation_Deflationary.md
- Threat.md
- LMDB.md

Security and operations

- ApiAuth.md
- KeyManager.md
- Binaries.md
- Packaging.md
- Node_Stop.md
- Docker.md
- dockertest.md

Reference docs

- Falcon512.md
- Sha3-384.md
- Base56.md
- gui.md
- SRCFILEMAP.md
- MnemonicRecoveryAudit_2026-03-09.md

Whitepaper artifacts

- AthoCompleteWhitepaper.pdf
- AthoCoreWhitepaper_Compact.pdf
- falcon Docs.pdf

Modeling outputs

- Emissions Modeling/Entire_Overview.md
- Emissions Modeling/athoemissions250y_*.csv
- Emissions Modeling/athoemissions250y_*.txt

- Emissions Modeling/AthoEmissions250Y_*.pdf

Developer Onboarding

Source: Docs/ONBOARDING.md

Atho Network — Developer Onboarding

Status: Alpha documentation snapshot (2026-03-12).

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-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 Node Quickstart

Status: Alpha documentation snapshot (2026-03-12).

This walks you through running a full node, miner node, and wallet/API node, then using the CLI. All auth is secured via API key + user/pass, and ports are auto-assigned to avoid collisions.

This is the fastest setup path. For the full docs map, start at the documentation index.

Prereqs (why you need them)

- Python 3.11+ (or 3.9+) to run nodes and CLI.
- git and curl for fetching code and checking APIs.
- macOS/Linux/WSL recommended (Windows native works but WSL is smoother for toolchains).
- Prefer containers? See the Docker guide for building/running via docker compose (full/miner/wallet) as an alternative to local toolchains.

- Hardware guidance: OpenGL 2.0 compatible system, i3-class CPU or better, 4-8 GB RAM minimum, 120 GB SSD recommended.

1) Get the code + venv + deps

macOS / Linux / WSL:

Windows (PowerShell):

Why: venv keeps deps isolated; requirements pulls FastAPI/LMDB/requests, etc. If install fails (especially lmdb/kivy/cryptography), go to Troubleshooting.md -> Dependency install issues (venv + pip).

2) Create an API key (required)

- Pick a username/password (default user is atho). Password set = full permissions; blank password = read-only.
- Writes Src/Config/Api_Keys.json (keep private; chmod 600).

Why: All API calls (including CLI) require X-API-Key/X-User/X-Pass to prevent misuse.

2.5) Falcon + Kyber binary/runtime check (important)

Falcon:

- Runtime expects falcon_cli for your host platform.
- Install/register current host binary:
- If Falcon is missing, build with platform compiler first:
- macOS: clang (Xcode Command Line Tools)
- Linux: gcc/clang (build-essential)
- Windows: MSYS2 MinGW-w64 (gcc) or Visual Studio Build Tools (cl)

Falcon compile commands (full flags):

macOS (Apple Silicon/Intel):

Linux:

Windows (MSYS2 MinGW-w64):

Kyber:

- Wallet encryption/decryption paths use vendored Kyber and can auto-build shared libs.
- Ensure a C compiler is installed (cc/clang/gcc) so Kyber build can succeed when first needed.

Kyber compile commands (full flags, Kyber-1024 / KYBER_K=4):

macOS:

Linux:

Windows (MSYS2 MinGW-w64):

3) Join the network (start nodes)

- Ports auto-assign and are saved to Src/Config/NodePorts.json (P2P base is 56000).
- To point at an existing seed:
- Check sync: tail -f logs/<network>/network/network.log and look for handshakeok / syncprogress with growing tip_height.

4) Verify via API (optional but recommended)

- Full API port is in NodePorts.json (default ~10100). Success shows network, tip height/hash.
- Optional network hashrate telemetry check:

5) Use the CLI

macOS / Linux / WSL:

Windows (PowerShell):

- CLI loads Api_Keys.json and NodePorts.json; prompts for password if not set in env.

Technical Whitepaper (Project)

Source: Docs/WhitePaper.md

Atho Whitepaper (Core Brief)

Status: Alpha documentation snapshot (2026-03-15).

Table of Contents

1. Executive Summary

Atho is a UTXO blockchain built for deterministic validation, operational clarity, and post-quantum readiness. It combines Falcon-512 signatures, SHA3-based hashing, Base56 user addresses, and LMDB-backed state storage.

The objective is to keep consensus behavior explainable, node roles explicit, and state drift low across node, wallet, GUI, and explorer surfaces.

Project phase at this snapshot: Alpha hardening.

2. Origin and History

Atho started as an operator-first chain project. Early development exposed role confusion, stale UI/API state, weak incident traceability, and fragile restart behavior.

Development then shifted toward hardening:

- stronger validation and payout checks,
- cleaner role boundaries,
- more explicit node lifecycle control,
- and better structured logs for audit and recovery.

This paper preserves mission and context while removing stale implementation detail.

3. Vision and Principles

Atho follows five principles:

- Determinism: identical data should produce identical validation outcomes.
- Auditability: operators must be able to explain failures quickly.
- Role clarity: full/miner/wallet behavior must remain explicit.
- Cryptographic modernization: post-quantum and SHA3-first direction.
- Operational resilience: failures should be visible and recoverable.

4. Problems Atho Solves

Atho targets practical infrastructure issues:

- state drift across chain, mempool, wallet, and explorer,
- miner payout mismatches from stale defaults,
- limited visibility into emission, fee, and coinbase anomalies,
- and long-term risk from legacy cryptographic assumptions.

5. Architecture and Network Design

5.1 Identity and Addressing

- Falcon-512 for signatures.
- SHA3-384-derived key and transaction hashing.
- Base56 address format for user-facing workflows.
- Internal normalized HPK representation for deterministic processing.

5.2 Ledger and Validation

Atho uses a UTXO ledger with explicit spent/locked semantics, canonical transaction hashing, fee and dust policy enforcement, and coinbase limits tied to reward schedule plus fees.

5.3 Consensus and Chain Integrity

Block acceptance verifies linkage, structure, signatures, target bounds, and payout correctness. Reorg/orphan handling is treated as a normal consensus path.

Current PoW retargeting uses a deterministic interval-based weighted-median window on production networks: target block time is 120 seconds, retarget every 360 blocks, clamp per-epoch adjustment ratio to 0.60..1.85, clamp interval outliers, and weight recent intervals more heavily for responsiveness without excessive volatility. Timestamp safety uses median-time-past (MTPWINDOW=13) plus bounded future drift (MAXTIME_DRIFT=120 seconds) across verifier and direct chain-accept paths to reduce timestamp-manipulation attack surface.

5.4 Storage and APIs

Consensus

Source: Docs/Consensus.md

Consensus Overview

Status: Alpha documentation snapshot (2026-03-15).

This document explains Atho's consensus logic, how blocks/transactions are validated, how PoW/difficulty and rewards are enforced, and where the code lives. File references are relative to repo root.

Core constants and parameters (Src/Utility/const.py)

- Network: Constants.NETWORK (mainnet/testnet/regnet), address prefixes, block/tx size/weight limits, witness policy, time drift, checkpoint depth.
- Hard size limits: MAXBLOCKSIZEBYTES = 2,500,000 (2.5 MB base bytes), MAXBLOCKWEIGHT = 10,000,000, MAXTRANSACTIONSIZEBYTES = 250,000.
- Supply: rewardatoms schedule → Constants.getblockreward(height); current pre-tail issuance target is 30,000,000 ATHO, tail starts at height 21,102,000 (0.25 ATHO).
- Burn/floor: post-tail fee split target is 0% miner / 100% burn (floor-clipped), with a 21,000,000 ATHO supply floor enforced by burn clipping.
- Fees: FEEPERBYTEATOMS, MINTRANSACTIONFEEATOMS, DUSTLIMITATOMS, TXIDEXCLUDEFIELDS, TXIDINCLUDEPUBKEY.
- Current fee/dust constants:
- FEEPERBYTE_ATOMS = 200
- MINTRANSACTIONFEE_ATOMS = 187,500
- DUSTLIMITATOMS = 5
- Current production retarget constants:
- TARGETBLOCKTIME = 120
- DIFFICULTYADJUSTMENTINTERVAL = 360
- ratio clamps: 0.60 .. 1.85 (MINDIFFICULTYFACTOR, MAXDIFFICULTYFACTOR)
- timestamp guards: MTPWINDOW = 13, MAXTIME_DRIFT = 120
- confirmation policy: TXCONFIRMATIONSREQUIRED = 10, COINBASEMATURITYBLOCKS = 400
- Difficulty bounds: MINDIFFICULTY, MAXDIFFICULTY; target validation uses these in block verification.

How block size works (end-to-end)

- Canonical limits are defined in Src/Utility/const.py:
- MAXBLOCKSIZE_BYTES = 2,500,000
- MAXBLOCKWEIGHT = 10,000,000
- MAXTRANSACTIONSIZE_BYTES = 250,000
- During mining/template build (Src/Miner/miner.py), candidate transactions are selected with both per-tx and per-block byte caps in mind before PoW starts.
- Block byte size is computed from canonical compact JSON serialization in Block.todict() / calcsizefromstd() (Src/Blockchain/block.py) and stored as sizebytes.
- On validation (Src/Main/blockveri.py), a block is rejected if either:
- Constants.blockweightokfromobj(...) fails, or

- sizebytes > MAXBLOCKSIZEBYTES.
- ConsensusSupervisor.acceptblock (Src/Main/consensus.py) also enforces that each included transaction is at or below MAXTXSIZEBYTES before full acceptance.
- Practical result: both caps are active; whichever limit is reached first rejects the block.

Runtime optimization path (non-consensus semantics)

These changes improve liveness and throughput behavior without changing core monetary/cryptographic rules:

- Miner tx-in-chain checks use tx index fast path (blockstore.gettx_location) before fallback scans.
- Miner template assembly is revision-cached and incrementally sized (running payload accounting) instead of repeated full-candidate reserialization.
- Mempool readers are revision-gated with mempool:meta:revision, reducing unnecessary LMDB rebuild scans across full/miner/wallet processes.
- P2P block relay uses lean payloads (drops export/duplicate fields on wire) and response fitting to message limits.
- API history/recent paths use hard scan caps and cursor pagination to avoid unbounded request-time scans.

Proof of Work and targets (Src/Miner/pow.py)

Falcon-512 Integration

Source: Docs/Falcon512.md

Falcon-512 Integration Notes and Audit

Status: Alpha documentation snapshot (2026-03-12).

This explains how Falcon-512 is integrated into the Atho stack, where signing and verification happen, how keys are handled, and the security posture of the current implementation (excluding the C source itself under Src/Falcon/Falcon).

Related reference package: Docs/falcon Docs.pdf (NIST/Falcon context used by this repo).

Components (code paths)

- Binary: platform-tagged binaries live under Src/Falcon/Binaries/<platformtag>/falconcli(.exe) with legacy fallback to Src/Falcon/Falcon/falconcli. Build script: Src/Falcon/Falcon/compilefalcon.sh; registration script: Src/Falcon/Falcon/installplatformbinary.py.
- Wrapper: Src/Accounts/falconcli.py locates the binary, enforces strict version-bound pinning on required networks (FALCONCLIPINDOMAIN, FALCONCLIPINREQUIREDNETWORKS, FALCONCLIPINNEDVERSIONDIGESTS), validates it (runs keygen), and exposes keygen, sign, and verify helpers. Optional FFI verify fallback via FALCONVERIFYSO.
- Key manager: Src/Accounts/keymanager.py calls FalconCLI for keygen/sign/verify, enforces CLI-format keys, and stores them in JSON (default Keys/KeyManagerPublicPrivateKeys.json). It derives hashed public keys (HPK) and Base56 addresses from the Falcon public key.
- Transaction hashing: Src/Utility/txdhash.py defines canonicaltxid (no-witness hash) and signingmessagehexdigest (sha3384 of canonical fields) used for signing and verification.
- Transaction signing: Src/Main/send.py and Src/Transactions/tx.py build txs, then call KeyManager.sign_transaction, which in turn invokes FalconCLI.sign on the signing digest.
- Transaction verification: Src/Transactions/txvalidation.py::validatesignature strips witness data (SegWit.stripwitness), recomputes the signing digest, and calls KeyManager.verifytransaction, which uses FalconCLI.verify/verifytransactionhash (multiple call orders + optional FFI).
- Block validation: Src/Main/blockveri.py trusts TXValidation for per-tx signature checks; coinbase has no signature.

Key and signature formats

- Public key: Falcon-512, 1024 bytes = 2048 hex chars. Private key components f/g/F/G: 1793 bytes each = 3586 hex chars.
- Signatures: treated as 2048 hex chars (1024 bytes). Verification accepts 1024 or 2048 hex length.
- Wire format: transaction witness fields (pubkey, signature) are canonical base64 on-chain/wire; they are decoded back to hex/bytes before Falcon verification.
- Hashing/signing payload: sha3384 digest (96 hex chars) of the canonical no-witness transaction body from signingmessagehexdigest. Witness (tx.signature) is not included in the txid.
- Address binding: outputs carry hashedpublickey (HPK); HPK is derived from the Falcon public key and can be rendered as Base56 via Src/Blockchain/base56.py.

Binary discovery and hardening (Src/Accounts/falconcli.py)

- Path search order: provided clipath → ATHOFALCONCLIBIN env override → PATH (falconcli) → platform-tagged candidates in Src/Falcon/Binaries/ → legacy Src/Falcon/Falcon/falcon_cli.
- Hash pinning: preferred path is version-bound pinning (FALCONCLIPINNEDVERSIONDIGESTS[CONSENSUSVERSION]); legacy raw-binary pinning (FALCONCLIEXPECTEDHASH) is a fallback.
- Startup validation: supports mode-controlled CLI sanity checks:

SHA3-384 and Addressing

Source: Docs/Sha3-384.md

SHA3-384, HPK, and Base56 Addressing

Status: Alpha documentation snapshot (2026-03-12).

This document describes how Atho uses SHA3-384 for identity, transaction hashing, and addressing, and how Base56 encoding wraps hashed public keys (HPK). It also lists the code paths involved and audit notes.

Hash primitives (Src/Utility/hash.py)

- sha3(data): SHA3-384, returns 48-byte digest.
- sha3_hex(data): SHA3-384, returns 96-char lowercase hex.
- sha3256 / sha3256_hex: SHA3-256 helpers (used for checksums).
- Merkle helpers use SHA3-384 for parent/root computation.

HPK (hashed public key) generation and storage

- Public keys are Falcon-512 (2048 hex chars). HPK is sha3384(pubkeybytes).hex() (96 hex chars), typically prefixed with the internal network tag (ATHO mainnet, ATHT testnet).
- Key import/generation:
- Src/Accounts/keymanager.py: derives HPK when adding keys; enforceclionly() ensures CLI-format keys with HPK present; also uses sha3384 in address export helpers.
- Src/Accounts/changedefaultkey.py: import helper routes materials through KeyManager validation/HPK recompute before persistence.
- HPK is the canonical on-chain address binding (hashedpublickey fields in outputs/UTXO).

Base56 address encoding (Src/Blockchain/base56.py)

- Alphabet: 23456789ABCDEFGHJKMNPQRSTUVWXYZabcdefghjkmnpqrstuvwxyz (56 chars, no visually ambiguous symbols).
- Input: HPK hex (96 chars), with or without internal prefix (ATHO/ATHT).
- Encoding:
- Convert HPK hex → 48 bytes.
- Prefix: A (mainnet) or T (testnet/regnet).
- Body: prefix + base56(rawhpkbytes).
- Checksum: first 4 bytes of sha3_256(body) encoded base56, padded to 6 chars.
- Final address: <prefix><body><checksum>.
- Decoding:
- Verifies prefix matches expected network (or Constants.NETWORK when enforced).
- Recomputes checksum with sha3_256 and compares.
- Returns HPK hex (padded to 96 chars) and network; can re-add internal prefix.
- Normalization helper: normalize_hpk accepts either Base56 address or prefixed HPK and returns the internal HPK string.
- Consumers:
- Wallet/CLI: Src/GUI/cliui.py and wallet APIs render Base56 for users, normalize back to HPK internally.
- Tx construction: Src/Main/send.py resolves recipient/change via normalize_hpk.
- UTXO lookup: Src/Transactions/txmanager.py normalizes HPK for spend selection.

Transaction hashing and signing (SHA3-384)

- Canonical txid (no witness):
- Src/Utility/txdhash.py::canonicaltxid hashes a normalized transaction dict (without witness fields, with hashedpublickey normalized) using SHA3-384 hex.
- Used for tx identifiers, merkle inclusion, and signing payloads.

- Signing payload:
- signingmessagehexdigest(tx) builds a deterministic string from version, locktime, sorted inputs/outputs, metadata, and txid, then sha3384(...).hexdigest().lower().
- Excludes witness/signature/pubkey; SegWit-style.
- Signing path:
- Src/Main/send.py builds the tx, then calls KeyManager.sign_transaction.
- KeyManager.sign_transaction normalizes the signing message to hex (detects 48-byte or 96-char SHA3-384 digests), then calls FalconCLI.sign on that hex string.
- Signature is stored once at tx.signature (witness); inputs do not carry the full sig.

Transactions

Source: Docs/Tx.md

Atho Transactions (Current Canonical Behavior)

Status: Alpha documentation snapshot (2026-03-12).

This is the current tx reference for structure, hashing, witness handling, sizing, fee policy, and validation flow.

Wire format (compact, current)

Top-level fields:

- tx_id
- fee (fixed 9-decimal string; atom-aligned)
- timestamp
- inputs
- outputs
- signature (witness, canonical base64 on wire)
- pubkey (witness, canonical base64 on wire)
- network
- coinbase
- metadata (only when non-empty)
- block_height (optional metadata field; not consensus payload for fee sizing)

Input fields:

- txoutid (<prev_txid>:<vout>)
- scriptsig (canonical SIGREF:<16hex>)
- sequence (only included when non-zero)

Output fields:

- amount (fixed 9-decimal string)
- hashedpublickey
- is_locked (only included when true)

Removed redundant wire fields

The wire format no longer depends on older redundant fields:

- input tx_id, index, address duplicates
- output txoutid payload duplication
- tx pubkeysize, signinghash, signinghashmode
- always-on default flags/fields that can be derived

This materially reduced high-input tx size.

High-input impact:

- each extra input now carries only the spend reference (txoutid) + compact script reference (SIG_REF) plus optional non-zero sequence,
- no duplicated per-input alias fields are emitted on wire.

Witness encoding

- Internal signing/verification still uses Falcon hex values.

- Wire format uses canonical base64 for signature and pubkey.
- Decoder is strict canonical base64 (wiretohex(...)), with legacy hex disabled by default:
- WITNESSWIREALLOWHEXLEGACY = False

Hashing model

- txid: SHA3-384 over no-witness tx (SegWit-style).
- Signing digest: SHA3-384 over deterministic no-witness signing payload.
- wtxid: SHA3-384 over full tx (with witness).
- Witness never changes txid or signing digest.

Size and weight model

Canonical metrics are:

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

Helper: Constants.txpolicymetrics(tx) returns (basebytes, totalbytes, weight, vsize).

Policy and consensus sizing both use vsize as the canonical fee/limit unit.

Important:

- size_bytes in tx logs is the canonical vsize used for policy.
- Raw serialized full bytes are reported separately as total_bytes in diagnostics when needed.
- For block packing and fee checks, vsize is the authoritative metric.

Fee policy (current constants)

- FEEPERBYTE_ATOMS = 200
- MINTRANSACTIONFEE_ATOMS = 187,500 (0.000187500 ATHO)
- DUSTLIMITATOMS = 5 (0.000000005 ATHO)

Required fee formula:

- requiredfeeatoms = max(vsize * 200, 187500)

SIG_REF rule (consistency fix)

script_sig is a compact reference only, never a full embedded signature:

- required canonical pattern: SIG_REF:<16hex>
- same rule is enforced in send path, mempool validation, tx validation, and block validation

This removed the prior class of “accepted in mempool, rejected in block” SIG_REF mismatches.

Bookkeeping field exclusion in policy math

Constants.txpolicymetrics(...) strips internal bookkeeping keys before sizing:

- block_height
- size
- size_bytes
- sizebasebytes
- weight
- vsize

This prevents fee drift when non-consensus metadata is attached during pipeline stages.

Signing flow

- Build base tx (empty witness/signature refs).
- Compute no-witness signing digest.
- Sign via Falcon CLI.

SegWit and Witness Rules

Source: Docs/Sigwit.md

Atho SegWit + Throughput Reference (Updated)

Status: Alpha documentation snapshot (2026-03-12).

This document explains how witness data is handled, how block/tx size is measured, and how to estimate TPS from real chain data.

Canonical limits

From Src/Utility/const.py:

- MAXBLOCKSIZE_BYTES = 2,500,000 (base-byte ceiling)
- MAXBLOCKWEIGHT = 10,000,000
- MAXTRANSACTIONSIZE_BYTES = 250,000 (applied as max tx vsize)
- WITNESSSCALEFACTOR = 4

Witness model

- Witness fields:
- top-level signature, pubkey, witness
- per-input script_sig
- Base transaction:
- witness removed at top-level
- inputs[].script_sig blanked
- Hashing:
- txid = no-witness hash
- wtxid = full hash
- Block commitment:
- block header carries witness_commitment when witnesses are present

Wire encoding

- Witness signature and pubkey are canonical base64 on wire.
- Internal verification uses decoded hex/bytes.
- Legacy hex witness wire is disabled by default (WITNESSWIREALLOWHEXLEGACY = False).

Exact size formulas

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

vsize is the canonical policy metric for:

- tx fee floor,
- tx size acceptance,
- mempool admission,
- miner selection.

Why logs show multiple size fields

Transaction logs include:

- size_bytes (canonical vsize used by fee policy),
- sizebasebytes,
- weight,
- vsize (same policy value as size_bytes).

So for policy/TPS planning, use vsize.

Consensus size vs relay payload size

- vsize/weight drives fee policy and block admission.
- Separate relay optimizations (for example lean block wire payloads that omit export-only fields like txlistfull) improve propagation latency but do not change consensus tx vsize.
- For TPS/capacity math, always use consensus tx vsize, not transport JSON byte counts from relay wrappers.

TPS formula

Let:

- B = 2,500,000 (effective vbytes per block ceiling from weight/base caps)

- T = block interval in seconds
- F = packing factor (use 0.95 for practical estimate)
- S = average non-coinbase tx vsize

Then:

- TPS ~= (B F) / (T S)

Production vs dev timing

- Production planning target: T = 120s.
- Some dev runs use shorter block intervals for testing. TPS scales linearly with 1/T.

Observed tx shapes (2026-03-07 logs)

- 1 in / 2 out: vsize=1371
- 3 in / 2 out: vsize=1646
- 16 in / 2 out: vsize=3428
- 31 in / 2 out: vsize=5483 (older pre-cleanup sample was 10012)

Estimated practical TPS at T=120, F=0.95:

- 1371 vB -> ~14.44 TPS
- 1646 vB -> ~12.02 TPS
- 3428 vB -> ~5.77 TPS
- 5483 vB -> ~3.61 TPS

Verify from your own chain

Use your transaction log or exported blocks and compute:

- average non-coinbase vsize,
- observed average block interval,
- observed TPS over a recent window.

Always publish all three together (size, interval, TPS) to avoid misleading comparisons.

API Authentication

Source: Docs/ApiAuth.md

API Authentication & Permissions

Status: Alpha documentation snapshot (2026-03-12).

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-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

LMDB Storage Overview

Status: Alpha documentation snapshot (2026-03-12).

This describes how Atho uses LMDB for chain, UTXO, mempool, orphan handling, and state checkpoints. Paths and knobs are driven by Constants (see Src/Utility/const.py).

Database layout and paths

- Root: blockchainstorage/<NETWORKFOLDER>/ (network-specific).
- Main environments (Constants.DATABASES):
- fullblockchain.lmdb — BlockStore (blocks by height/hash).
- utxo.lmdb — UTXOStore.
- mempool.lmdb — MempoolStore.
- orphan_blocks.lmdb — orphan/side-branch staging (OrphanReorgManager).
- Additional LMDB:
- blockchainstorage/BlockData/statehashes.lmdb — StateTracker Merkle accumulator.
- Map sizes/bounds:
- Initial sizes: BLOCKSTOREMAPSIZE, UTXOMAPSIZE, MEMPOOLMAPSIZE, STATEMAPSIZE, ORPHANMAPSIZE.
- Per-store rollover ceilings: BLOCKSTOREMAXMAPSIZE, UTXOMAXMAPSIZE, MEMPOOLMAXMAPSIZE, STATEMAXMAPSIZE, ORPHANMAXMAP_SIZE.
- Global controls: LMDBMAXREADERS, AUTORESIZELMDB, LMDBGROWTHFACTOR, LMDBMINGROWTHSTEPBYTES, LMDBROLLOVERALERTTHRESHOLD, LMDBRESIZERETRYATTEMPTS.

- Runtime logging: LMDBSTATUSLOGINTERVALSECONDS rate-limits memory-pressure logs while a store is near its rollover threshold.

Rollover model

- Atho does not shard a single logical store across multiple LMDB environments yet.
- In current code, "rollover" means bounded LMDB map growth: when a write hits lmdb.MapFullError, the store asks Src/Storage/lmdbutils.py for the next map size and grows the existing environment.
- Growth is store-specific and stops at that store's configured *MAXMAP_SIZE.
- Once a store reaches its max map size, writes stop auto-growing and the node logs lmdbmaplimit_reached. At that point you must raise the configured ceiling or archive/prune data.
- While a store is near the rollover threshold, the node emits lmdbmemorystatus log entries with usedbytes, freebytes, utilization_percent, and the next planned map size.
- Those LMDB memory events are also written to logs/<network>/storage/lmdb_memory.log so storage pressure is preserved even if terminal log filtering is enabled.
- Stores expose lmdbstatus() and the aggregate checker in Src/Storage/dbcheck.py reports usedbytes, freebytes, utilizationpercent, nearrollover, and nextmap_size.

BlockStore (Src/Storage/blockstorage.py)

- Keys:
- blk:h:<height> → compact JSON block.
- blk:x:<hash> → compact JSON block.
- blk:t:<txid> → tx location index payload (blockheight, blockhash, tx_index).
- meta:blk:txindex_ready → tx-index readiness marker.
- Serialization: tocompact/fromcompact("Block", ...); Decimal/bytes serialized via custom JSON hook.
- Writes: putblock enforces network, normalizes index/height, merges witness info, prunes witnesses for old blocks based on retention, and writes both height/hash keys in one LMDB txn. Optional guard ENFORCECONSENSUSWRITE requires consensuswrite_context.
- Tx index maintenance is in the same LMDB write transaction as block writes and deletes so tx lookup stays consistent with chain state.
- Persistent witness pruning: compactprunedwitnesses() now rewrites eligible old block records so witness maps and txlistfull payloads are actually removed from disk as the tip advances.
- Reads: getbyheight/getbyhash expand compact, prune witnesses in-memory if past retention window, and attach full txs/witnesses for display.

Base56 Addressing

Source: Docs/Base56.md

Base56 Address Encoding

Status: Alpha documentation snapshot (2026-03-12).

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)

Status: Alpha documentation snapshot (2026-03-12).

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 2048-char hex in key files).
- 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,
- PQC-capable DEK recovery path (Kyber),

Emissions and Monetary Policy

Source: Docs/Emissions.md

Atho Emissions, Burn, and Supply Floor (Consensus)

Status: Alpha documentation snapshot (2026-03-17).

Scope

This document defines the active consensus monetary policy.

- All consensus-critical accounting is integer atoms.
- 1 ATHO = 1,000,000,000 atoms.
- This is policy/engineering documentation, not a market forecast.

30M Standard (Active)

Core constants

- BLOCKTIMESECONDS = 120
- BLOCKSPERYEAR = 262,800
- BLOCKSPERERA = 1,314,000 (5 years)
- SUPPLY_FLOOR = 21,000,000 ATHO

Pre-tail schedule

First five eras (fixed):

- Era 1: 10.0 ATHO/block (0 .. 1,313,999)
- Era 2: 5.0 ATHO/block (1,314,000 .. 2,627,999)
- Era 3: 2.5 ATHO/block (2,628,000 .. 3,941,999)
- Era 4: 1.25 ATHO/block (3,942,000 .. 5,255,999)
- Era 5: 0.625 ATHO/block (5,256,000 .. 6,569,999)

Transition phase:

- Reward is fixed at 0.3125 ATHO/block.
- Transition runs until exact pre-tail supply reaches 30,000,000 ATHO.
- Transition block span: 6,570,000 .. 21,101,999.
- Transition block count: 14,532,000.

Pre-tail totals:

- First five eras: 25,458,750 ATHO
- Transition issuance: 4,541,250 ATHO
- Exact pre-tail total: 30,000,000 ATHO

Tail (perpetual)

- Tail reward: 0.25 ATHO/block
- Tail start height: 21,102,000
- Tail start year (120s blocks): ~80.3
- Tail annual issuance: 65,700 ATHO/year

Fee policy and burn

Defined in Src/Utility/const.py:

- FEEPERBYTE_ATOMS = 200 (2.0e-7 ATHO/vB)
- MINTRANSACTIONFEE_ATOMS = 187,500 (0.000187500 ATHO)
- DUSTLIMITATOMS = 5 (0.000000005 ATHO)
- Max block base size: 2,500,000 bytes

Burn policy:

- Before tail start: miner receives 100% of fees.
- At and after tail start: target split is 100% burn / 0% miner fee share.
- Burn is floor-clipped so effective circulating supply cannot go below 21,000,000 ATHO.

Derived post-tail economics

At 100% block-byte utilization:

- Max annual fee pool: 131,400 ATHO/year
- 2,500,000 262,800 2.0e-7
- Tail annual issuance: 65,700 ATHO/year
- Deflation threshold utilization: 50%
- 65,700 / 131,400 = 0.50

Net annual change formula (post-tail):

- Delta_supply = 65,700 - (131,400 * utilization)

Interpretation:

- <50% utilization: inflationary net.
- 50% utilization: neutral net.

- >50% utilization: deflationary net (until floor clipping binds).

Coinbase and accounting invariants

Consensus enforces:

- coinbaseoutputssumatoms == blockrewardatoms + feesminer_atoms

Tail-era blocks must carry auditable fee/burn fields:

- feestotalatoms
- feesmineratoms
- feesburnedatoms
- cumulativeburnedatoms

Monitoring and outputs

- Src/Main/emission.py tracks expected coinbase payout and floor-aware supply accounting.
- Src/Main/burn.py tracks fee split and cumulative burned totals.
- Src/Main/esim.py generates long-horizon modeling outputs in Docs/Emissions Modeling/.

Hard-fork note

Changing any of these is consensus-breaking:

- reward schedule,
- pre-tail target supply,
- tail reward,
- tail start behavior,
- fee floor/min fee,

Inflation/Deflation Model

Source: Docs/Inflation_Deflationary.md

Atho Inflationary/Deflationary Network Model (30M Standard)

Status: Alpha documentation snapshot (2026-03-17).

Scope

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

Canonical implementation references:

- Src/Utility/const.py
- Src/Main/blockveri.py
- Src/Storage/blockstorage.py
- Src/Main/emission.py
- Src/Main/esim.py

Executive summary

- Block time: 120 seconds (262,800 blocks/year).
- Era size: 1,314,000 blocks (5 years).
- First 5 eras follow 10 -> 5 -> 2.5 -> 1.25 -> 0.625 ATHO/block.
- Then reward is fixed at 0.3125 ATHO/block until exact supply reaches 30,000,000 ATHO.
- Tail starts at height 21,102,000 (year ~80.3) with 0.25 ATHO/block forever.
- Tail annual issuance is fixed at 65,700 ATHO/year.
- Fee floor is 200 atoms/vB (2.0e-7 ATHO/vB).
- Post-tail split target is 100% burn / 0% miner fee share, clipped by floor headroom.
- Supply floor is 21,000,000 ATHO.
- Protocol deflation threshold is exactly 50% sustained block-byte utilization.

Emission schedule

| Phase | Height range | Reward (ATHO/block) | Issuance (ATHO) | Cumulative (ATHO) | |---|---|---:|---:|---:| | Era 1 | 0 .. 1,313,999 | 10.0 | 13,140,000 | 13,140,000 | | Era 2 | 1,314,000 ..

Fee and burn math

Per block at full utilization

- Max fee pool per full block: 2,500,000 * 200 atoms = 500,000,000 atoms = 0.5 ATHO
- Tail issuance per block: 0.25 ATHO
- Net per-block change formula: Delta_block = 0.25 - (0.5 * utilization)

Per year at full utilization

- Tail issuance: 0.25 * 262,800 = 65,700 ATHO/year
- Max annual fee pool: 2,500,000 262,800 2.0e-7 = 131,400 ATHO/year
- Annual net formula: Delta_year = 65,700 - (131,400 * utilization)

Deflation threshold

Set annual net to zero:

65,700 - 131,400u = 0 u = 65,700 / 131,400 = 0.50

So:

- <50%: inflationary protocol net.
- =50%: neutral.
- >50%: deflationary protocol net (until floor clipping binds).

Post-tail net annual change by utilization

| Utilization | Net annual change | |---:|---:| | 0% | +65,700 ATHO/year | | 25% | +32,850 ATHO/year | | 40% | +13,140 ATHO/year | | 50% | 0 ATHO/year | | 60% | -13,140 ATHO/year | | 75% | -32,850 ATHO/year | | 100% | -65,700 ATHO/year |

Reference long-horizon outcomes (model-generated scenarios)

| Scenario | Supply at year 250 | |---:|---:| | No burn (100% capacity) | 41,149,500 ATHO | | Burn on (25% capacity) | 35,565,000 ATHO | | Burn on (50% capacity) | 29,980,500 ATHO | | Burn on (75% capacity) | 24,396,000 ATHO | | Burn on (100% capacity) | 21,000,000 ATHO (floor reached around year 217) |

Floor-clipped burn behavior

Burn is constrained by available headroom:

- burnheadroom = emitteduptо - cumulativeburnedbefore - supplyfloor

Threat Model

Source: Docs/Threat.md

Threat Model and Security Posture

Status: Alpha documentation snapshot (2026-03-15).

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

- Private keys stored unencrypted on disk.
- 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 verification accepts 1024-hex sigs in addition to Falcon-512 (2048-hex).
- Verbose logging leaks metadata (key lengths/previews, tx details).
- Tail issuance is active, with post-tail fee burn target (100%, 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 stored in plaintext (no encryption/at-rest protection). Impact: 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: KeyManager.verify_transaction accepts 1024- or 2048-hex sigs. Impact: ambiguity or acceptance of non-Falcon-512 artifacts.
- 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:
- release manifest can be verified at runtime (security_manifest.json),
- optional Falcon signature verification over canonical manifest payload.
- Consensus ingress coupling:
- accepttx and acceptblock reject with runtimeguardviolation in enforce mode if guard is violated.
- Reorg rollback cap:
- reorg depth is capped by policy (maxreorgdepthblocks), returning reorgdepth_exceeded when exceeded.

Docker Operations

Source: Docs/Docker.md

Docker Guide (Atho)

Status: Alpha documentation snapshot (2026-03-12).

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

- Dockerfile: builds a Python 3.11 image, installs deps, builds falcon_cli, and runs Src/Node/fullnode.py by default.
- docker-entrypoint.sh: simple entrypoint wrapper.
- 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 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 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).
- Host-mapped ports: set ATHOAPIURL=http://localhost:<hostapiport> when running cliui.py.

Docker Test Harness

Source: Docs/dockertest.md

Docker P2P Test Harness

Status: Alpha documentation snapshot (2026-03-12).

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 a clean state

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

Building Binaries (Advanced)

Status: Alpha documentation snapshot (2026-03-12).

This guide is for advanced users who want to build Atho binaries from source, including the falconcli helper, on Linux, macOS, and Windows. If you want the fastest path, use Docker (it builds falconcli inside a container with all dependencies preinstalled).

Cross-OS Binary Layout (required)

Runtime now checks platform-tagged Falcon binary folders first:

- Src/Falcon/Binaries/darwin-arm64/falcon_cli
- Src/Falcon/Binaries/darwin-x8664/falconcli
- Src/Falcon/Binaries/linux-x8664/falconcli
- Src/Falcon/Binaries/linux-arm64/falcon_cli
- Src/Falcon/Binaries/windows-x8664/falconcli.exe

App Binary Build (GUI launcher binary)

You can now generate release binary artifacts with:

Optional modes:

Output goes to:

- releases/binaries/<timestamp>/...

Important:

- Native GUI binaries should be built on their target OS for best compatibility.
- Windows .exe should be built on Windows host (or dedicated CI runner).

- Script always creates a source snapshot fallback package via makeprealphapackage.sh.

You can install the current host build into the correct folder with:

Windows (PowerShell):

Optional explicit source path:

The script prints:

- binarysha3384
- version-bound digest
- exact const.py pin values to update (FALCONCLIPINNEDVERSIONDIGESTSBYPLATFORM and FALCONCLIEXPECTEDHASHBY_PLATFORM)

Prerequisites by OS

Linux (Debian/Ubuntu)

- build-essential (gcc/g++/make)
- cmake (if you rebuild beyond the provided script)
- libssl-dev (for hashing/crypto)
- Python 3.11+ and pip install -r requirements.txt

macOS

- Xcode Command Line Tools: xcode-select --install
- Homebrew packages (if needed): brew install openssl cmake (adjust PATH/CPPFLAGS/LDFLAGS for OpenSSL if non-system)
- Python 3.11+ and pip install -r requirements.txt

Windows

- Recommended: WSL (Ubuntu) with the Linux prerequisites above.
- Native (more work): install Visual Studio Build Tools (C++ workload) or MSYS2/MinGW-w64; ensure gcc/clang and make are on PATH. Python 3.11+ with pip, and pip install -r requirements.txt.
- PowerShell/WSL tip: use WSL to avoid toolchain headaches; Docker Desktop can build for you (see below).

Building falcon_cli from source

Falcon sources live in Src/Falcon/Falcon/. The Dockerfile builds with gcc by default.

Linux / macOS

Notes:

- The script builds with -O3 -std=c99 -Wall -Wextra -DFALCON_FPNATIVE.
- Set FALCONUSEAVX2=true on x86_64 with AVX2 support; keep false on ARM/older CPUs.
- After build, ensure falconcli is executable: chmod +x falconcli.

Windows (WSL recommended)

- WSL path: follow Linux steps above.
- Native: open MSYS2/MinGW shell or Developer Command Prompt, and compile with your toolchain:
- MinGW example: gcc -O3 -std=c99 -Wall -Wextra -DFALCONFPNATIVE -o falconcli cli.c falcon.c fft.c fpr.c keygen.c rng.c shake.c sign.c vrfy.c common.c codec.c
- Adjust flags for AVX2: add -mavx2 -DFALCON_AVX2 if supported.
- Ensure falconcli.exe ends up in Src/Falcon/Binaries/windows-x8664/ (or set ATHOFALCONCLIBIN).

Troubleshooting

Source: Docs/Troubleshooting.md

Troubleshooting Guide (Atho)

Status: Alpha documentation snapshot (2026-03-12).

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)

Status: Alpha documentation snapshot (2026-03-12).

This utility stops Atho node processes even when they were started from another terminal/background session.

It targets:

- fullnode.py
- minernode.py
- walletnode.py

- runminer.py
- runnode.py (launcher, optional/available)

Why this exists

runnode.py launches background processes but does not include a dedicated stop command. stop.py gives you one place to:

- stop one node role (full, miner, wallet)
- stop everything
- stop specific PIDs
- integrate the same logic from API/GUI code

CLI Usage

From project root:

No flags opens an interactive menu where you can choose:

- stop all
- stop full
- stop miner
- stop wallet
- stop launcher
- stop by PID

Non-interactive examples:

API-Friendly Functions

You can import and reuse these directly:

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)

Signal Behavior

For each matched process:

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

This is designed to cleanly stop nodes first, then force-stop only when needed.

GUI/API Integration

The API node stop routes now call this stop logic, so GUI stop actions can terminate:

- processes started by API controller
- processes started from other terminals/background launchers

The API also exposes active node monitoring:

- GET /nodes/active -> returns active node list, roles, and PIDs (countsbyrole, processes, by_role).

Safety Scope

stop.py only matches Atho node/launcher command paths listed above. It does not blanket-kill all Python processes.

Emissions Modeling Overview

Source: Docs/Emissions Modeling/Entire_Overview.md

ENTIRE OVERVIEW - ATHO EMISSIONS MODELING AUDIT

This document is a complete observation-based audit 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

Audit 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, 5 initial pre-tail eras of 1,314,000 blocks each, followed by a transition reward of 0.3125 ATHO/block until 30,000,000 ATHO, and a tail reward of 0.2500 ATHO/block, fee floor 0.00000020 ATHO/byte (200 atoms/byte), max block size 2,500,000 bytes, burn share 100.0%, miner fee share 0.0%, and supply floor 21,000,000 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 80 (off) and 81 (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, and supply floor). 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 5-era + transition reward sequence (10 -> 5 -> 2.5 -> 1.25 -> 0.625 -> 0.3125 (until 30M) ATHO/block), producing an exact pre-tail issuance of 30,000,000.000 ATHO. Tail mode starts at height 21,102,000 (~year 80.30).

Tail annual issuance is 65,700.000 ATHO. At 100% block-byte utilization, annual fee pool is 131,400.000 ATHO and max annual burn equals 131,400.000 ATHO.

Scenario Outcomes

Year-250 circulating supply ordering is strictly monotonic: NoBurn100=41,149,500.000 ATHO, Burn25=35,565,000.000, Burn50=29,980,500.000, Burn75=24,396,000.000, Burn100=21,000,000.000. This confirms expected policy behavior: greater sustained utilization under burn policy removes more aggregate fees from circulating supply.

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].