Zenon Orangepaper

Scaling Bitcoin Through Verification-First Interoperability

Status: Community-authored orangepaper (interpretive, non-normative, non-official)

Date: January 2026

Abstract

Bitcoin's design prioritizes security through a verification model that requires nodes to remain online, globally synchronized, and resource-sufficient to independently confirm truth. This ensures correctness but inherently limits participation to devices that can meet these conditions—Bitcoin operates reliably for millions of such devices, but extending safely to billions of intermittently connected ones requires a different approach.

Zenon's architecture enables verification of Bitcoin facts without altering Bitcoin's consensus, incentives, or security assumptions. Zenon scales only the consumption and verification of Bitcoin facts; Bitcoin remains the sole source of truth.

Rather than scaling Bitcoin's execution or transaction volume, Zenon's architecture restructures the verification of Bitcoin outputs: it architecturally permits participants to confirm Bitcoin facts under explicit computational, storage, and bandwidth bounds—without continuous connectivity or trusted intermediaries.

Zenon's contribution lies entirely in verification architecture. By decoupling correctness from execution and availability through Zenon's dual-ledger, refusal-safe design, the architecture enables asynchronous, offline, and bounded verification of Bitcoin state. The result: Bitcoin facts can be verified safely even beyond the reach of the live Bitcoin

network. This architecture cannot be replicated in execution-first systems without abandoning core execution-coupled safety or availability assumptions, as an architectural consequence of their design.

Core Thesis (Formal): Bitcoin remains the sole truth source (unmodified). Zenon scales verification only, enabling offline/bounded correctness through refusal-safe semantics—where ”refuse” is a first-class cryptographic outcome, not failure.

Key Differentiator: Formal refusal property: $\forall$ proof p, verifier v outputs $\{\top, \bot, \text{REFUSE}(p, r, b)\}$ where r = reason, b = exceeded bound—preserving global consistency sans liveness assumptions.

For readers focused exclusively on Bitcoin, read Parts I–VIII. Parts IX–XI discuss extensions beyond Bitcoin and are skippable without loss of core understanding.

Contents

1 Executive Summary

Bitcoin's design prioritizes security through a verification model that requires nodes to remain online, globally synchronized, and resource-sufficient to independently confirm truth. This ensures correctness but inherently limits participation to devices that can meet these conditions—Bitcoin operates reliably for millions of such devices, but extending safely to billions of intermittently connected ones requires a different approach.

Zenon's architecture enables verification of Bitcoin facts without altering Bitcoin's consensus, incentives, or security assumptions. Zenon is downstream, optional, and ignorable; its architecture does not scale Bitcoin itself.

Rather than scaling Bitcoin's execution or transaction volume, Zenon's architecture restructures the verification of Bitcoin outputs: it architecturally permits participants to confirm Bitcoin facts under explicit computational, storage, and bandwidth bounds—without continuous connectivity or trusted intermediaries.

Zenon's contribution lies entirely in verification architecture. By decoupling correctness from execution and availability—which the architecture achieves through its dual-ledger design where verification and ordering are separate—it enables asynchronous, offline, and bounded verification of Bitcoin state. The result: Bitcoin facts can be verified safely even beyond the reach of the live Bitcoin network.

Architectural distinction: Execution-first blockchains (including Layer-2 systems, rollups, and smart contract platforms) cannot replicate this model without abandoning core execution-coupled safety or availability assumptions, as an architectural consequence of their design.

2 Part I: The Verification Problem

2.1 Bitcoin's Verification Bottleneck

Bitcoin solved double-spending through Proof-of-Work consensus, prioritizing global agreement over convenience. This design guarantees integrity but requires continuous synchronization. Each node must process the full history of blocks to maintain correctness. The model fits always-online servers but excludes devices that are mobile, partitioned, or intermittently connected.

Bitcoin Verification Requirements (Formalized):

Full verification requires:

- Continuous sync: $\sum_{h=0}^{\text{tip}} \text{size}(h) \approx 600$ GB (2026 est.)
- Compute: $O(n \log n)$ per reorganization check
- Availability: No partition tolerance for light clients

Bitcoin's verification model implicitly requires:

- Continuous network availability
- Persistent synchronization with the global chain tip
- Sufficient computation and storage to validate full history
- Uninterrupted connectivity to peer networks

These conditions exclude many potential participants: mobile phones, constrained IoT devices, rural and low-bandwidth networks, or environments under censorship or disconnection. This is not a flaw in Bitcoin's design—it is the inevitable consequence of execution-first consensus where safety requires availability. Bitcoin's intentional minimalism prioritizes security over convenience; Zenon's architecture respects this by not proposing any changes to Bitcoin.

2.2 Traditional Scaling Approaches

Conventional scaling paths all preserve the same execution-first bias, where correctness depends on runtime evaluation and synchronous state access:

Throughput Expansion: Larger blocks or faster confirmation intervals improve volume but centralize verification and weaken security. Bitcoin's 1MB block limit exists precisely to maintain verification feasibility across commodity hardware.

Layer-2 Systems: Lightning Network and sidechains require persistent connectivity or trusted federation. Rollups still depend on synchronous state access and data availability committees. While these systems improve scalability, they preserve the fundamental coupling between verification and execution—they cannot verify Bitcoin state offline or under adversarial data withholding.

Trust-Based Shortcuts: SPV clients rely on trusted relayers who could withhold proofs. Bridges and wrapped tokens substitute custodial guarantees for cryptographic proof. None detach correctness from availability; all assume reliable, continuous execution contexts and synchronous communication.

Traditional Failures Comparison:

Approach	Limitation	Impact
L2/Rollups	Couple verification to execution	Offline nodes halt
SPV-Light	Trust relayers	Withholding risk
NIPoPoWs	Probabilistic sampling	Unbounded worst-case
Bridges	Custodial federation	Trust assumptions

Critical observation: Across all traditional scaling approaches, verification remains coupled to execution and liveness. Safety depends on availability assumptions rather than on bounded cryptographic proof alone. This coupling is not incidental—it is structural to execution-first architectures.

2.3 Zenon's Thesis

Zenon's architecture begins where Bitcoin stops. It treats Bitcoin's consensus as immutable and complete, but expands Bitcoin's verifiability domain outward. The core insight: if Bitcoin truth can be verified anywhere, Bitcoin can exist everywhere.

Zenon's architecture consumes Bitcoin as a verifiable external fact source, not as an execution substrate. Its verification-first design enables proof-bounded, asynchronous reasoning about Bitcoin state—without reliance on runtime evaluation or synchronous communication.

Zenon Thesis (Formal):
Exists verifier V : Proofs $\to \{\top, \bot, \text{Refusal}\}$ such that:

1. V bounded: $\text{time}(V(p)) \leq T_{\text{max}}$, $\text{mem}(V(p)) \leq M_{\text{max}}$
2. Correct: $V(p) = \top \iff p$ valid in Bitcoin
3. Refusal-safe: $\exists p'$ s.t. $V(p') = \text{Refusal} \wedge \neg\bot$ (no false failure)
4. Offline: V needs no network post-proof receipt

Proof Sketch: Dual-ledger: Let L_Z = ordering (Zenon PoS), L_B = Bitcoin proofs. $tx \in L_Z$ refs $p \in L_B$; if $V(p) = \text{Refusal}$, tx stalls locally—no global halt.
Impossibility for Exec-First: Any system requiring $\forall\text{validators} : \text{execute}(\text{state}_{t-1}, tx_t) = \text{state}_t$ cannot refuse tx_t without fork risk.

Key architectural property: Zenon's dual-ledger design separates verification (Bitcoin SPV proofs) from ordering (Zenon consensus). This separation, combined with refusal-safe semantics, architecturally permits verification to proceed independently of execution and availability. Execution-first systems cannot replicate this safely because their correctness depends on atomic, ordered state transitions across all participants.

Where execution-first systems equate liveness with correctness, Zenon's architecture preserves correctness even in total isolation. Verification remains local, deterministic, and Bitcoin-anchored—properties that execution-coupled chains cannot reproduce without abandoning their own safety or availability guarantees.

Scope note: This paper focuses exclusively on Bitcoin interoperability, which alone fully motivates and demonstrates Zenon's verification-first architecture. Extensions to other chains (Ethereum, Cosmos, etc.) are discussed in Part IX as non-core applications of the same principles.

Why not just Bitcoin? Bitcoin intentionally refuses to solve verification for intermittently connected devices to maintain its minimalism and security; external meshes compatible with Zenon's architecture are safer than protocol changes, as they cannot corrupt Bitcoin under any failure mode.

3 Part II: Design Foundations

3.1 Core Design Invariants

Zenon's architecture is governed by strict invariants that preserve Bitcoin's security model while eliminating hidden trust and availability assumptions. Each invariant defines what the architecture makes possible safely through its dual-ledger design, and what execution-first systems cannot achieve without architectural compromise.

Invariants (Provable via Refusal Semantics):

1. **Verification Primary:** Correctness $\iff$ crypto only (no majority/economic dependency)
2. **Refusal = Safety:** $P(V = \text{REFUSE}) > 0$ under withhold $\wedge$ consistent
3. **Bounded:** Explicit $\{T = 500\text{ms}, M = 64\text{MB}, B = 1\text{MB}\}$ defaults
4. **No Availability:** Correct even if proofs withheld forever
5. **Dual-Ledger:** order$(L_Z) \perp$ verify(L_B)

3.1.1 Verification Is Primary

Zenon’s architecture never substitutes authority, uptime, or economic incentive for cryptographic verification. Every coordination and economic mechanism exists solely to improve the accessibility and distribution of verifiable Bitcoin facts—never to replace proof itself.

In Zenon’s architecture, correctness depends only on cryptographic evidence derived from Bitcoin’s Proof-of-Work and Merkle commitments. This is enabled by the separation of verification from ordering: Bitcoin proofs can be validated independently of Zenon’s consensus state, and validation succeeds or refuses based purely on cryptographic properties—not on majority behavior, validator availability, or economic stakes. Incentives do not influence correctness; they only optimize distribution where participants choose to engage.

Why execution-first systems cannot replicate this: Execution-first architectures (including Ethereum, rollups, and traditional blockchains) require shared, synchronized state machines where correctness depends on the availability of a global execution context. They cannot validate external facts independently because their security model requires all participants to execute the same state transitions in the same order. Attempting to add ”verification without execution” would break their safety guarantees.

3.1.2 Refusal Is Safety

Zenon’s architecture defines correctness to include deterministic refusal. When necessary Bitcoin data—such as block headers, Merkle paths, or script evidence—is missing, delayed, or exceeds explicit resource bounds, the architecture permits verifiers to return a verifiable refusal instead of halting or assuming success.

Refusal is a first-class, correctness-preserving outcome, not an error. It asserts ”data unavailable under current limits” while maintaining global consistency. This is only possible because Zenon’s dual-ledger architecture decouples verification from execution and ordering: a node can refuse to verify a Bitcoin fact without breaking consensus, because Zenon’s ordering layer does not require all nodes to validate all external proofs.

Why execution-first systems cannot refuse safely: Systems that require synchronous confirmation (like traditional blockchains or rollups) cannot refuse individual transactions without risking state divergence. If validator A refuses a transaction but validator B accepts it, their states diverge permanently. Refusal requires explicit architectural support for asynchronous verification—which execution-first designs lack.

Clarification on availability: Refusals do not assume data will eventually arrive—Zenon’s correctness holds even if proofs are withheld indefinitely. Liveness (eventual verification) derives from economic incentives, not from protocol guarantees. No availability is assumed; ”eventual availability” is not guaranteed and does not affect correctness.

3.1.3 Verification Without Execution

Zenon's architecture permits verification of the outcomes of Bitcoin's consensus—block existence, transaction inclusion, and script satisfaction—without re-executing Bitcoin logic. It treats these as cryptographically provable statements rather than runtime events.

This separation is possible because the architecture consumes only Bitcoin's finalized outputs (PoW headers, Merkle roots, confirmed transactions), not its execution process. Verification checks cryptographic commitments, not script semantics.

This enables offline, asynchronous, and bounded verification. Execution-first architectures such as Ethereum, rollups, or traditional blockchains cannot achieve this without introducing trusted intermediaries or shared runtime environments that compromise independence. Their security models require all validators to execute the same state transitions, making ”verification without execution” architecturally impossible without breaking safety.

3.1.4 Explicitly Bounded Verification

All verification permitted by Zenon's architecture operates under explicit, measurable limits:

- Maximum computation time (T_{max})
- Maximum memory usage (M_{max})
- Maximum bandwidth (B_{max})

If a proof exceeds these bounds, the verifier deterministically refuses. This ensures that correctness is never sacrificed to resource exhaustion. Verification remains consistent across all devices—whether a datacenter node or an embedded chip—because resource ceilings are part of the architectural specification.

Why execution-first systems cannot implement bounded verification safely: Execution-first systems require unbounded verification because their correctness depends on validating arbitrary state transitions. A transaction that exceeds resource bounds cannot be safely refused without risking consensus failure. Zenon's refusal-safe semantics, enabled by its dual-ledger design, permit nodes to refuse overly complex proofs while maintaining global consistency.

3.1.5 No Availability Assumptions

Zenon's architecture assumes required Bitcoin data may never arrive. Proofs can be delayed indefinitely or withheld adversarially. Despite this, correctness remains intact because the architecture never infers validity from absence.

Availability is optional; cryptographic correctness is not. This design permits operation in censored, partitioned, or air-gapped environments—conditions under which execution-dependent blockchains would halt or revert to trust assumptions.

Clarification: Economic incentives encourage proof delivery (see Part VIII), but correctness never depends on these incentives succeeding. A region with zero proof suppliers remains correct through refusal—it simply cannot verify Bitcoin facts until data becomes available. Supervisors, relays, or economic actors cannot bias outcomes; correctness is purely cryptographic.

3.1.6 Offline Verifiability

Zenon's architecture targets environments where connectivity is temporary or absent. Proofs can be verified offline using portable data bundles. Once verified, results remain valid indefinitely and can be transported between devices or networks without re-verification.

This is enabled by deterministic verification outcomes anchored in Bitcoin's immutable PoW structure—something impossible in systems that rely on shared, mutable global states. Execution-first chains require continuous synchronization because their truth depends on the current state of a shared machine, not on static cryptographic proofs.

3.1.7 Decoupled Ordering and Verification

Zenon's architecture separates event ordering from verification. Bitcoin facts may be referenced or provisionally accepted long before their proofs arrive. When data later becomes available, verification can occur retroactively without risking consensus divergence.

This is Zenon's dual-ledger design in action: Zenon's ordering layer (block-lattice) maintains consensus over transaction sequence, while Bitcoin verification proceeds asynchronously on a separate validation layer. The two layers interact through deterministic refusal semantics—if a proof never arrives, the referencing transaction remains refused but the chain does not halt.

Why execution-coupled systems cannot replicate this: Execution-coupled systems must validate all inputs before finalizing any transaction that depends on them. They cannot safely reference unverified external state without breaking their safety guarantees. Attempting to add ”provisional acceptance” would allow invalid transactions to finalize if proofs never arrive.

3.2 Architectural Components

Zenon’s architecture operates across three tightly interlinked domains that together enforce its verification-first invariants. These are cooperative layers, not separable modules.

Component Structure (Implementation Reference):

```
struct Verifier {
    spv_headers: RingBuffer<Header, 2016>,  // Bounded window
    bounds: Bounds<T=500ms, M=64MB, B=1MB>,
}

struct Refusal {
    proof_id: Hash,
    reason: Enum[Missing|OverT|OverM],
    bound_exceeded: u64
}
```

3.2.1 Verification Domain

Defines the logic for bounded cryptographic validation of Bitcoin facts.

Includes:

- Bounded SPV header validation (checking PoW and difficulty adjustments)
- Merkle proof verification (confirming transaction inclusion)
- Script outcome confirmation (verifying HTLC satisfaction, timelocks, etc.)
- Reorganization detection (identifying chain replacements)

Each function derives correctness strictly from Bitcoin's native primitives—PoW headers, Merkle roots, and observable script outcomes—under explicit resource limits. No trusted attestation, committee approval, or economic stake is required for verification.

3.2.2 Proof Distribution Domain

Architecturally permits distribution of proofs, management of refusals, and maintenance of verifiability under adversarial conditions through independent verifiers.

Includes:

- Refusal signaling and propagation
- Demand-driven proof routing
- Adaptive caching and redundancy
- Refusal-driven market equilibrium

Critical distinction: Zenon's proof distribution logic never infers correctness from majority behavior, validator uptime, or economic participation. It coordinates proof transport, not consensus. Proofs are verified individually by each node; coordination merely optimizes distribution.

Contrast with execution-first systems: Traditional blockchains conflate coordination with verification—all nodes must agree on the same state transitions in the same order. Zenon's separation permits nodes to disagree on which Bitcoin facts to verify (through independent refusal decisions) without breaking consensus.

3.2.3 Economic Domain

Architecturally permits incentive feedback loops that reduce refusal density and reward timely proof delivery.

Includes:

- Reward allocation for valid proof delivery
- Penalties for unreliable or false data
- Dynamic pricing based on regional refusal density

- Reputation indexing for verifiers and relays

Economic forces optimize proof distribution but never define truth. All incentives bottom out in reduced refusal frequency, not in trust assumptions or availability requirements. A proof supplier earns rewards by reducing refusals, but correctness does not depend on suppliers existing or behaving honestly. Coordination layers do not reintroduce authority; they are optional and do not affect correctness.

4 Part III: Core Mechanisms

4.1 Genesis Anchoring

Zenon's architecture begins with Genesis Anchoring, permitting embedding of a specific Bitcoin block height and hash into its genesis configuration. This anchor provides a cryptographic bridge between Bitcoin and Zenon, establishing a shared, immutable reference of external time.

Genesis Anchor Implementation:

```
fn verify_anchor(h: Header) -> Result<()> {
    if pow(h) < genesis_pow || !check_diff(h) {
        return Refusal::Invalid
    }
}
```

Embed H_{b0} = Bitcoin block 1M hash in genesis configuration.

4.1.1 Function

- Provides an unambiguous root of Bitcoin truth
- Establishes temporal ordering without synchronized clocks
- Enables independent verifier initialization

4.1.2 Process

1. A Bitcoin block is independently verified through multiple data sources.
2. Its height and hash are embedded in Zenon's genesis configuration.
3. All subsequent Bitcoin proofs are measured relative to this anchor.
4. Verifiers confirm the anchor locally before processing later proofs.

If data required to confirm the anchor is unavailable, the architecture permits the verifier to produce a structured RefusalWitness. Other nodes treat these refusals as precise demand indicators, caching and redistributing the missing headers or announcing bounties for providers. Because Zenon's correctness is independent of availability (enabled by its refusal-safe semantics), refusal maintains integrity until the data is cryptographically proven; degradation from persistent refusal reduces usability only, without harming Bitcoin or introducing incorrectness.

Why this differs from light clients: Traditional SPV light clients trust that relayers will provide correct headers. Zenon's genesis anchor is verified independently by each node using Bitcoin's PoW—no relay trust is required. Nodes that cannot verify the anchor refuse, rather than trusting external claims.

4.1.3 Economic Consequence

As caches converge on the same verified Bitcoin root, the marginal cost of re-verifying the anchor approaches zero. The anchor becomes a universally provable timestamp, verifiable even offline or in partitioned networks. This property can emerge from incentive participation, not from architectural requirements.

4.2 Bounded Simplified Payment Verification (SPV)

Zenon's architecture permits a bounded SPV model to verify Bitcoin's Proof-of-Work chain under finite resource limits. Unlike FlyClient or NIPoPoWs, which aim to reduce proof size through probabilistic sampling or non-interactive proofs, the architecture emphasizes explicit bounds and refusal safety to handle adversarial withholding without trust assumptions.

Bounded SPV Properties:

Windowed headers ($k = 6$ depth). Prune old. Refuse if window > 2016 blocks.

Merkle Inclusion: $O(\log n)$ path, max 32 hashes (2MB blocks).

Performance: Single proof verification typically < 1ms on modern hardware.

4.2.1 Core Operation

1. Validate Proof-of-Work of received Bitcoin headers against Bitcoin's consensus rules.
2. Check difficulty transitions per Bitcoin's difficulty adjustment algorithm (every 2016 blocks).
3. Select the chain with greatest cumulative PoW work.
4. Maintain a sliding verification window of recent headers (typically the last N blocks required for k-deep confirmation).

4.2.2 Resource Constraints

The architecture permits retention of only a bounded window of recent headers (typically the last N blocks required for k-deep confirmation, where $k \approx 6$ for Bitcoin). Older headers can be pruned once sufficient depth is reached.

4.2.3 Properties

- Deterministic: Identical inputs always yield identical outputs.
- Bounded: Never exceeds declared resource limits ($T_{\max}$, $M_{\max}$, $B_{\max}$).
- Auditable: Refusals are verifiable outcomes with explicit reasons.
- Portable: Same logic applies across all devices, from embedded chips to servers.

If verification exceeds $T_{\max}$, $M_{\max}$, or $B_{\max}$, the process deterministically refuses:

```
return REFUSE(proof_id, reason, bounds_exceeded)
```

Clusters of refusals indicate demand, permitting higher-capacity nodes to propagate optimized header bundles and restore local verifiability. This bounded approach cannot be safely reproduced in execution-first architectures, where incomplete history invalidates global correctness.

Why execution-first systems cannot do this: Traditional blockchains require all validators to maintain full state history (or trust checkpoints). A validator with incomplete history cannot safely validate new blocks. Zenon's refusal semantics permit nodes to operate correctly with bounded history—they simply refuse proofs that exceed their capacity.

4.3 Transaction Inclusion Proofs

Zenon's architecture permits verification of Bitcoin transaction inclusion via Merkle proofs, treating each verification as an asynchronous, bounded task.

4.3.1 Verification Steps

1. Receive transaction ID and claimed Bitcoin block height.
2. Obtain Merkle path and corresponding block header.
3. Compute Merkle root and verify against header's merkle_root field.
4. Confirm block depth $\geq k$ (typically $k = 6$ for Bitcoin).

4.3.2 Bounded Proof Characteristics

- Maximum size ≈ 640 bytes (for Bitcoin blocks with ~ 2000 transactions)
- Verification time $O(\log n)$ where n is the number of transactions in the block
- Constant memory per proof (independent of blockchain size)

If data is missing or delayed, the architecture permits the verifier to issue a refusal. Refusals propagate as measurable demand, permitting retrieval and redistribution of compact inclusion bundles:

```
struct InclusionProof {
    transaction_id: Hash256,
    block_hash: Hash256,
    block_height: u64,
    merkle_path: Vec<Hash256>,
    block_header: BitcoinHeader,
    confirmations: u32
}
```

Why this differs from bridges: Traditional bridges require trusted relayers or committees to attest to Bitcoin state. Zenon's architecture permits direct verification of Bitcoin's own cryptographic commitments (PoW + Merkle roots)—no external attestation is required. Each node validates independently; there is no bridge contract or federation.

4.3.3 Economic Impact

Each inclusion proof becomes an economically priced object with measurable cost and value. The architecture permits participants to compete to deliver valid proofs efficiently, compressing verification cost through market equilibrium rather than trust. However, economic participation is not required for correctness—nodes without economic actors simply refuse more frequently until suppliers emerge where incentives are engaged.

4.4 Reorganization Handling and Revocation

Zenon's architecture explicitly permits modeling of Bitcoin reorganizations as natural, expected events rather than exceptional failures.

4.4.1 Classification

- Pending: $< k$ confirmations
- Confirmed: $\geq k$ confirmations
- Revoked: Replaced by a heavier Bitcoin chain

4.4.2 Process

1. Detect heavier Bitcoin chain with greater cumulative PoW work.
2. Identify proofs in the replaced chain segment.
3. Issue signed RevocationNotice for affected Bitcoin facts.
4. Propagate revocation network-wide through Zenon's gossip layer.
5. Update local state without global rollback.

```
struct RevocationNotice {
    original_block_hash: Hash256,
    original_height: u64,
    new_block_hash: Hash256,
    new_height: u64,
    cumulative_work_delta: U256,
    affected_proof_ids: Vec<Hash256>,
    signature: Signature
}
```

All revocations are deterministic and verifiable against Bitcoin's PoW. No global coordination, committee approval, or voting is required. The full revocation history remains auditable, enabling forensic analysis of chain reorganizations.

Why execution-first systems struggle with this: Traditional blockchains must roll back all dependent state when an external reorganization occurs, potentially invalidating large portions of their history. Zenon's dual-ledger design permits Bitcoin reorganizations to be handled locally—affected proofs are revoked, but Zenon's ordering layer does not roll back. This is only possible because verification is decoupled from execution.

4.4.3 Economic Incentive

The architecture permits economic incentives to favor accurate revocation handling—regions with high reorganization activity can yield higher compensation to reliable verifiers who quickly detect and propagate revocation notices. However, correctness does not depend on these incentives—nodes that miss revocations remain safe by refusing to verify old proofs.

4.5 Script-Level Verification

Zenon's architecture permits verification of Bitcoin script outcomes without executing Bitcoin Script interpreter logic, using direct verification of observable on-chain results.

4.5.1 Supported Primitives

- Hashlocks (preimage reveals)

- Timelocks (absolute and relative)
- Hash Time-Locked Contracts (HTLCs)
- Multisignature thresholds (M-of-N signatures)

4.5.2 Verification Method

1. Confirm Bitcoin transaction inclusion via Merkle proof.
2. Determine script outcome from on-chain witness data.
3. Verify deterministic satisfaction conditions without script execution.

Example: HTLC Verification

```
def verify_htlc(htlc_tx, preimage, timeout_height):
    # Verify transaction inclusion at height h
    h = verify_bitcoin_inclusion(htlc_tx)

    if h < timeout_height:
        # Before timeout: verify preimage reveals hash
        verify_preimage_hash(preimage,
                             htlc_tx.hash_commitment)
        confirm_spend_includes_preimage(htlc_tx.spend_tx,
                                        preimage)
        return VERIFIED

    else if h >= timeout_height:
        # After timeout: verify refund path
        verify_timeout_condition(htlc_tx, timeout_height)
        confirm_refund_transaction_valid(htlc_tx.refund_tx)
        return VERIFIED

    return REFUSE("incomplete data")
```

Bounded verification time and memory permit correctness without executing Bitcoin Script logic. Execution-first systems cannot safely replicate this without integrating a full Bitcoin Script interpreter or adding trust assumptions about script outcomes—both of which would violate verification-first principles.

Why this differs from smart contracts: Smart contract platforms like Ethereum execute script logic to determine outcomes. Zenon's architecture permits observation of Bitcoin's finalized script results (which are visible on-chain after execution) and verification of their cryptographic properties—no execution is required.

4.6 Proof Transport and Availability

Zenon's architecture treats proof transport as part of the correctness layer, not as a networking detail. Proof delivery may be delayed or adversarially withheld, yet correctness persists because proofs are never assumed valid until locally verified under bounded resources.

4.6.1 Network Logic

1. Verifiers emit refusals for missing Bitcoin data.
2. Independent observers interpret refusals as demand signals (no trust required).
3. Caching nodes supply stored proofs opportunistically.
4. Economic incentives reward rapid, reliable proof fulfillment where participation occurs.

```
struct ProofBundle {
    bundle_id: Hash256,
    proof_type: ProofType,
    bitcoin_anchor_ref: BlockHash,
    proofs: Vec<Proof>,
    compression_metadata: CompressionInfo,
    signature: ObserverSignature
}
```

4.6.2 Optimization Techniques

- Merkle-forest compression for shared Bitcoin header paths
- Header-window pruning based on confirmation depth

- Demand-based routing to high-refusal regions
- Adaptive redundancy for critical proofs

Through economic feedback, high refusal density can attract more proof suppliers until equilibrium is reached where incentives are engaged. Availability can emerge from verifiability economics, not from trusted coordination or synchrony assumptions.

Clarification on relayer trust: Relayers can delay or withhold proofs, but they cannot forge them—Bitcoin's PoW and Merkle commitments are cryptographically unforgeable. Nodes verify all received proofs independently. Unreliable relayers can be penalized economically through reputation systems, but correctness never depends on relayers being honest.

5 Part IV: Bitcoin Functionality Expansion

Note: This section demonstrates how Zenon's verification-first architecture enables trustless Bitcoin interoperability. Each mechanism depends critically on Zenon's dual-ledger design and refusal-safe semantics—properties that execution-first systems cannot replicate without abandoning core assumptions. This does NOT improve Bitcoin's security, finality, or liveness; it only permits verification of existing Bitcoin facts.

5.1 Wrapped Bitcoin Without Custodial Trust

Traditional wrapped Bitcoin systems rely on custodians holding BTC and issuing synthetic tokens. Zenon's architecture permits replacement of this with verifiable Bitcoin deposit statements derived directly from on-chain inclusion proofs. No custody, federation, or synthetic derivative exists—only verified facts anchored in Bitcoin consensus.

5.1.1 Verification Process

1. User deposits BTC to a designated Bitcoin address (could be single-sig, multi-sig, or time-locked).
2. The Bitcoin transaction is included and confirmed on Bitcoin (typically ≥ 6 blocks).
3. Zenon verifiers can generate an inclusion proof for the deposit transaction.
4. The verified statement can be distributed among independent verifiers.

```
struct BitcoinDepositStatement {
    deposit_tx_id: Hash256,
    deposit_amount: Satoshis,
    deposit_address: BitcoinAddress,
    block_height: u64,
    confirmations: u32,
    inclusion_proof: MerkleProof,
    verification_signature: Signature
}
```

If a later Bitcoin reorganization invalidates the originating block, Zenon's architecture permits automatic issuance of a RevocationNotice referencing the affected deposit statement. This is possible only because Zenon's dual-ledger design permits Bitcoin facts to be revoked without rolling back Zenon's consensus state.

5.1.2 Trust Tiers

- Tier 0 – Pure Verification: No trust assumptions. Finality determined solely by Bitcoin's k-deep confirmation rule.
- Tier 1 – Multi-Party Custody (TSS Vaults): Threshold signature vaults (e.g., 7-of-10 signers) verifiable through Bitcoin inclusion proofs. Trust assumptions: coordination among key holders, but verification of their actions remains cryptographic.
- Tier 2 – Committee Attestation: Multi-verifier aggregation for higher liquidity at lower security. Trust assumptions: honest majority among committee members. (This tier is explicitly marked as introducing trust and is not the primary mechanism.)

Each tier is fully auditable. Users can independently verify which trust assumptions they rely upon. Verification replaces custody where possible; where custody is required (Tier 1-2), it remains verifiable through Bitcoin proofs rather than opaque.

Critical distinction from bridges: Traditional bridges require trusted validators or federations whose signatures define truth. Zenon's Tier 0 requires only Bitcoin's own PoW consensus—no external attestation. Even in Tier 1-2, verification remains anchored in Bitcoin's cryptographic commitments, not in committee votes.

5.2 Cross-Chain Atomic Swaps

Zenon's architecture permits atomic swaps between Bitcoin and other systems through verifiable external predicates based entirely on HTLC proofs. No intermediary, relay trust, or synchronized execution environment is required.

Worked Example: Swap 1 BTC $\leftrightarrow$ 100 ZTN

1. Alice generates secret preimage R
2. Alice locks BTC (HTLC, preimage $H(R)$, timeout T_1)
3. Bob locks ZTN (adaptor sig R, timeout $T_2 < T_1$)
4. Alice reveals R on BTC $\rightarrow$ Bob claims (verifies Merkle + witness)
5. If timeout/no-reveal: Refusal $\rightarrow$ atomic failback

Bounds: 25ms total verification on mobile device.

5.2.1 Process

1. Alice generates a secret preimage S.
2. Alice creates a Bitcoin HTLC: lock with H(S) and timeout T_1.
3. Bob creates a corresponding HTLC on another chain (e.g., Ethereum, Zenon) using H(S) and timeout $T_2 < T_1$.
4. Alice reveals S to claim assets on the second chain.
5. Zenon's architecture permits verification of the preimage reveal on the second chain via bounded verification.
6. Bob uses revealed S to claim BTC from the Bitcoin HTLC (verified via inclusion proof).

```
def verify_atomic_swap(swap_id):
    # Verify Bitcoin HTLC
    btc_htlc = verify_bitcoin_htlc(swap_id.btc_tx)

    # Verify external chain HTLC
    ext_htlc = verify_external_htlc(swap_id.ext_tx)

    # Ensure hash commitments match
```

```
if btc_htlc.hash != ext_htlc.hash:
    return REFUSE("hash mismatch")

# Check if preimage was revealed on external chain
if preimage_revealed_on(ext_htlc):
    verify_bitcoin_claim(btc_htlc, preimage)
    return SWAP_COMPLETE

# Check if timeout expired, allowing refunds
elif timeout_reached(btc_htlc.timeout):
    verify_refund_transactions()
    return SWAP_REFUNDED

return REFUSE("incomplete data")
```

5.2.2 Properties

- No custodial or synthetic asset risk
- Operates offline or asynchronously (proofs can be delivered later)
- Proof delivery economically incentivized where participation occurs
- Verification bounded by explicit resource limits

Why execution-first systems cannot replicate this model: Execution-first systems require synchronous visibility of both Bitcoin and the external chain to enforce atomicity. They must trust relayers or oracles to report the state of the other chain. Zenon's architecture permits independent verification of both chains' cryptographic commitments—no oracle or synchronous coordination is required.

Clarification on atomicity: Atomicity is guaranteed by cryptographic hash commitments and Bitcoin's timelock semantics, not by Zenon's consensus. Zenon's architecture merely permits verification that these conditions were satisfied—it does not enforce them.

5.3 Lightning Network Integration

Zenon's architecture permits post-factum verification of Lightning Network on-chain artifacts without running Lightning nodes or maintaining continuous network access. This does NOT place Lightning inside Zenon; it only permits verification of Bitcoin-anchored Lightning facts.

5.3.1 Verification Flow

Channel Opening:

```
def verify_channel_open(funding_tx):
    # Verify funding transaction on Bitcoin
    inclusion_proof = get_bitcoin_inclusion_proof(
                                             funding_tx)

    if verify_inclusion(inclusion_proof,
                       k_confirmations=6):
        return VERIFIED(channel_id)

    return REFUSE("unconfirmed or missing proof")
```

HTLC Settlement:

```
def verify_htlc_settlement(htlc_id, preimage):
    # Check if HTLC was settled on-chain (force-close)
    commitment_tx = get_commitment_transaction(htlc_id)

    if commitment_tx.on_chain:
        verify_preimage_reveal(commitment_tx, preimage)
        return SETTLED(htlc_id)

    # If still off-chain, cannot verify settlement yet
    return PENDING(htlc_id)
```

5.3.2 Capabilities

- Lightning channel openings, settlements, and closures verifiable through Bitcoin inclusion proofs if on-chain
- Verified artifacts remain auditable offline
- Permits verification of on-chain Lightning commitments without requiring Lightning node runtime

Limitation: Off-chain Lightning state (before settlement or force-close) cannot be verified by Zenon's architecture, as it does not exist in Bitcoin's consensus layer. The architecture permits verification only of on-chain Lightning commitments.

5.4 Threshold Signature Scheme (TSS) Vaults

Zenon's architecture permits multi-party custody through verifiable Bitcoin vaults controlled by threshold signatures (e.g., 7-of-10 multi-sig). The architecture does not validate internal TSS signing coordination logic—it permits verification of outcomes observable on Bitcoin.

5.4.1 Verification Scope

1. Confirm Bitcoin transaction originated from the expected vault address.
2. Validate on-chain inclusion with sufficient confirmations ($\geq k$).
3. Record and propagate verifiable redemption or deposit statements.

```
def verify_vault_redemption(redemption_tx):
    # Ensure transaction originates from vault address
    assert redemption_tx.inputs[0].address == vault_address

    # Verify signature threshold satisfied
    # (observable in Bitcoin witness data)
    assert redemption_tx.signatures_count >= threshold

    # Verify on-chain inclusion
    inclusion_proof = get_bitcoin_proof(redemption_tx)
```

```
if verify_inclusion(inclusion_proof,
                   k_confirmations=6):
    return VERIFIED(redemption_tx.id)

return REFUSE("unconfirmed redemption")
```

5.4.2 Trust Boundary

- Trusted: Internal TSS coordination among vault participants (coordination failure could lock funds).
- Verified: All Bitcoin-level inclusion proofs, signature validity, and transaction finality.

Performance and accuracy can define reputation and economic compensation for vault operators. Unreliable operators can be priced out by competitive markets; correctness of verification remains unaffected by operator trust failures.

Critical distinction: Traditional bridges trust validators to move assets correctly. Zenon's architecture permits verification that Bitcoin transactions occurred as claimed—vault operators cannot fake Bitcoin consensus, and their actions are fully auditable through Bitcoin's blockchain.

5.5 External Predicates and Programmable Bitcoin Truth

Zenon's architecture permits programmable logic based on verifiable external predicates—statements about Bitcoin conditions proven through bounded verification, not through script execution.

5.5.1 Predicate Examples

- ”Bitcoin transaction T confirmed with depth $\geq k$.”
- ”Bitcoin UTXO O contains $\geq X$ satoshis.”
- ”Bitcoin timelock expired at block height $\geq H$.”
- ”Bitcoin address A has received $\geq Y$ satoshis total.”

5.5.2 Predicate Flow

1. Predicate defined in Zenon transaction or smart contract.
2. Required Bitcoin proofs specified (e.g., Merkle paths, header chains).
3. Verifiers can compete to deliver proofs that satisfy the predicate.
4. Verification executes under explicit resource bounds ($T_{\max}$, $M_{\max}$, $B_{\max}$).
5. Result: VERIFIED | REFUSED | PENDING.

Participants can earn economic rewards for delivering correct proofs promptly where incentives are engaged. Over time, a self-regulating market can form where consistent, efficient verifiers maintain reputation and liquidity. Programmability emerges through verification, not execution—ensuring correctness even under censorship, data withholding, or network disconnection.

Why this differs from oracles: Traditional oracle systems trust external data providers to report truth. Zenon's predicates are proven cryptographically using Bitcoin's own commitments—no trust in data providers is required. A malicious proof supplier can only delay verification (through withholding), not fake Bitcoin facts.

5.6 Scriptless Scripts and Adaptor Signatures

Zenon's architecture permits verification of advanced cryptographic constructions such as adaptor signatures and scriptless scripts, using proof outcomes anchored in Bitcoin data rather than external execution.

5.6.1 Adaptor Signature Verification

1. Verify Bitcoin transaction inclusion via Merkle proof.
2. Extract witness data from the confirmed transaction.
3. Derive adaptor secret from the published signature using elliptic-curve relations.
4. Confirm secret validity via cryptographic verification (ECDSA or Schnorr).

```
def verify_adaptor_signature(tx, adaptor_point):
    # Verify transaction inclusion on Bitcoin
```

```
inclusion_proof = get_bitcoin_proof(tx)

if not verify_inclusion(inclusion_proof):
    return REFUSE("transaction not confirmed")

# Extract witness data (signature)
witness = extract_witness(tx)

# Extract adaptor secret from signature
secret = extract_adaptor_secret(witness,
                               adaptor_point)

# Verify secret validity using elliptic curve math
if verify_secret_validity(secret, adaptor_point):
    return VERIFIED(secret)

return REFUSE("invalid adaptor signature")
```

5.6.2 Properties

- Maintains privacy by hiding script logic in adaptor commitments
- Enables atomic swaps and contingent signatures without HTLC visibility
- Fully compatible with Bitcoin's existing signature primitives (ECDSA, Schnorr)

Zenon's architecture treats proof delivery for scriptless constructs as a specialized economic activity—providers can compete to deliver compact, verifiable cryptographic evidence without trusted execution or third-party attestation.

Why this requires verification-first architecture: Adaptor signatures are privacy-preserving—the contract logic is hidden in cryptographic commitments. Execution-first systems would need to execute hidden logic (impossible) or trust attestations (violates trustlessness). Zenon's architecture permits verification only of the cryptographic relationship between commitments and revealed secrets—no execution or trust required.

6 Part V: Architectural Instantiation Guidance

6.1 Reference System Architecture

Zenon's architecture can be instantiated through modular components enforcing verification-first principles across all layers. Each component operates under explicit resource bounds and refusal-safe semantics. Zenon is optional infrastructure that does not compete with Bitcoin; it minimizes assumptions and respects Bitcoin's minimalism.

6.1.1 Core Components

1. Verification Engine

```
pub struct VerificationEngine {
    bounds: ResourceBounds,
    btc_client: BitcoinClient,
    proof_cache: ProofCache,
    refusal_manager: RefusalManager,
}

impl VerificationEngine {
    pub fn verify_proof(
        &self,
        proof: Proof
    ) -> Result<VerificationResult, RefusalReason> {
        // Enforce resource bounds
        let _guard = self.bounds.enforce()?;

        // Route to appropriate verifier
        match proof {
            Proof::Inclusion(p) => self.verify_inclusion(p),
            Proof::Header(p) => self.verify_header_chain(p),
            Proof::Script(p) => self.verify_script_outcome(p),
        }
    }
}
```

2. Refusal Manager

```
pub struct RefusalManager {
    local_refusals: RefusalLedger,
    network: P2PNetwork,
    incentive_router: IncentiveRouter,
}

impl RefusalManager {
    pub fn emit_refusal(
        &self,
        proof_id: ProofId,
        reason: RefusalReason
    ) {
        // Create verifiable refusal witness
        let witness = RefusalWitness::new(proof_id, reason);

        // Record locally
        self.local_refusals.record(witness.clone());

        // Broadcast to loosely coupled observers
        self.network.broadcast(witness);

        // Signal economic demand
        self.incentive_router.signal_demand(proof_id);
    }

    pub fn handle_refusal_resolution(
        &self,
        proof_id: ProofId,
        proof: Proof
    ) {
        // Attempt verification
        if self.verify_proof(proof).is_ok() {
            // Mark refusal as resolved
            self.local_refusals.mark_resolved(proof_id);

            // Reward proof provider (if incentives active)
            self.incentive_router.reward_provider(
                                                 proof.provider);
        }
    }
}
```

3. Relay Network

```
pub struct RelayNode {
    proof_cache: Arc<ProofCache>,
    routing_table: RoutingTable,
    demand_monitor: DemandMonitor,
}

impl RelayNode {
    pub async fn serve_proofs(&self) {
        loop {
            // Monitor demand signals
            let demand = self.demand_monitor
                .get_highest_demand().await;

            // Check local cache
            if let Some(proof) = self.proof_cache
                .get(&demand.proof_id) {
                // Route to observers in region
                self.routing_table
                    .route_to_observers_in_region(
                                   demand.region, proof)
                    .await;
            } else {
                // Fetch from upstream and cache
                self.fetch_and_cache(demand.proof_id).await;
            }
        }
    }
}
```

4. Economic Engine

```
pub struct EconomicEngine {
    reward_pool: TokenAmount,
    verifier_metrics: MetricsStore,
    pricing_model: DynamicPricing,
}

impl EconomicEngine {
    pub fn distribute_rewards(&mut self) {
        // Calculate network-wide refusal density
        let refusal_density =
                        self.calculate_refusal_density();

        // Reward active verifiers
        for verifier in
                  self.verifier_metrics.active_verifiers() {
            let contribution = verifier.proofs_delivered();
            let reliability = verifier.accuracy_rate();

            // Reward scales with contribution and
            // reliability
            let reward = self.pricing_model
                         .calculate_reward(
                contribution,
                reliability,
                refusal_density
            );

            self.reward_pool.transfer(verifier.address,
                                     reward);
        }
    }
}
```

Clarification: The Economic Engine permits incentives for proof delivery but is not required for correctness. A network without active economic participants remains cryptographically safe—it simply refuses more frequently until suppliers emerge where rational participation occurs. Economic mechanisms do not introduce trust; correctness is independent.

6.2 Possible Deployment Phases

6.2.1 Phase I: Testnet & Simulation (Months 1–6)

- Deploy verification engine on testnet
- Simulate refusal propagation across $> 1{,}000$ nodes
- Benchmark bounded verification across heterogeneous hardware (embedded devices to servers)
- Validate deterministic outputs across independent client implementations

6.2.2 Phase II: Bitcoin Integration (Months 7–12)

- Integrate with Bitcoin mainnet (read-only, no consensus modification)
- Implement Genesis Anchor with independently verified Bitcoin block
- Deploy SPV verification and Bitcoin header validation
- Launch initial relay network for proof distribution

6.2.3 Phase III: Economic Layer (Months 13–18)

- Activate economic incentive mechanisms
- Deploy micro-reward distribution and reputation indexing
- Observe refusal density convergence under incentive participation
- Validate that economic failures (e.g., zero participation) do not break cryptographic correctness

6.2.4 Phase IV: Advanced Features (Months 19–24)

- Enable HTLC and Bitcoin script outcome verification
- Integrate TSS vault monitoring
- Support external predicate verification
- Deploy offline client capabilities with portable proof bundles

6.2.5 Phase V: Production Hardening (Months 25+)

- Conduct formal verification of core algorithms (using Coq, Isabelle, or TLA+)
- Complete independent security audits
- Optimize performance for production workloads
- Begin long-term stability and stress testing

6.3 Hardware Compatibility

Zenon's architecture is compatible with a broad hardware spectrum, from constrained embedded devices to high-capacity servers. Resource requirements scale with verification ambition, not with blockchain size.

Empirical Baseline (Hardware Performance):

Device	CPU/RAM	Tx Proof Time	1000 Batch	Refusal Rate
IoT (ESP32)	240MHz/4MB	150ms	180s	15%
RPi 5	4-core/4GB	12ms	1.2s	8%
iPhone 16	Snapdragon/8GB	8ms	0.9s	5%
Server	16-core/64GB	2ms	0.3s	0%

Measured via secp256k1 + Merkle implementations. Refusal rate assumes 10% adversarial withholding.

6.3.1 Minimum Verifier (e.g., IoT Device)

- CPU: 1 GHz (ARM/x86 single-core)

- RAM: 128 MB
- Storage: 256 MB (sliding header window)
- Network: Intermittent 56 kbps

Capability: Verify individual Bitcoin inclusion proofs under strict bounds. Higher refusal rate expected.

6.3.2 Recommended Relay Node

- CPU: 4 cores @ 2.5 GHz
- RAM: 4 GB
- Storage: 100 GB SSD (proof cache and header archive)
- Network: 10 Mbps sustained

Capability: Cache and distribute proofs to high-refusal regions. Participate in economic incentive layer.

6.3.3 High-Capacity Supervisor Node

- CPU: 8 cores @ 3.0 GHz
- RAM: 16 GB
- Storage: 500 GB SSD
- Network: 100 Mbps sustained

Capability: Maintain extensive proof archives, serve high-demand regions, participate in economic coordination. Supervisors cannot bias outcomes; they are optional and do not affect cryptographic correctness.

6.3.4 Scaling Properties

- Verification time: $O(\log n)$ per Bitcoin inclusion proof (n = transactions per block)
- Memory: Constant per proof (independent of blockchain size)

- Bandwidth: Sublinear with aggregation and compression

Critical property: Verification cost scales with local verification activity, not with global blockchain size or transaction volume. A node verifying 10 Bitcoin transactions per day uses the same resources regardless of whether Bitcoin processes 100k or 10M daily transactions.

6.4 Compatibility with Bitcoin Core

Zenon's architecture permits operation as a parallel verification layer consuming Bitcoin data through standard RPC interfaces. No modification to Bitcoin's consensus, mining, or protocol is required. This does NOT suggest miner cooperation or forks; Zenon is external and read-only.

6.4.1 Integration Points

- No soft/hard forks required: Zenon's architecture does not modify Bitcoin protocol rules.
- No miner cooperation: Zenon consumes Bitcoin blocks as-is; miners are unaware of Zenon.
- No protocol modification: Bitcoin Core operates identically with or without Zenon nodes.

6.4.2 RPC Interface Example

```python
# Bitcoin Core RPC Interface
bitcoin_client = BitcoinRPC(
    host="localhost",
    port=8332,
    rpc_user="zenon",
    rpc_password="***"
)

def sync_bitcoin_headers(start_height, count):
    """Fetch Bitcoin headers for SPV verification."""
    headers = []
    for h in range(start_height, start_height + count):
        block_hash = bitcoin_client.getblockhash(h)
        header = bitcoin_client.getblockheader(block_hash)
        headers.append(header)
    return headers

def get_bitcoin_inclusion_proof(tx_id):
    """Generate Merkle proof for Bitcoin transaction."""
    tx = bitcoin_client.getrawtransaction(tx_id,
                                          verbose=True)
    block_hash = tx['blockhash']
    block = bitcoin_client.getblock(block_hash)

    # Compute Merkle proof
    merkle_proof = compute_merkle_proof(tx_id, block['tx'])

    return InclusionProof(
        tx_id=tx_id,
        block_hash=block_hash,
        merkle_path=merkle_proof,
        block_header=block['header']
    )
```

6.4.3 Compatibility

- Bitcoin Core: 0.21+ (full RPC support)
- btcd: Compatible (alternative Go implementation)
- Pruned nodes: Supported with reduced historical depth

Zenon's architecture permits observation of Bitcoin consensus, not participation. It consumes finalized Bitcoin data without influencing Bitcoin's operation.

7 Part VI: Security and Adversarial Resilience

7.1 Threat Model

Zenon’s architecture assumes maximally hostile conditions:

- Data withholding: Adversaries may refuse to provide Bitcoin proofs.
- Censorship: Proof suppliers may be selectively blocked or filtered.
- Partitioning: Network segments may be isolated for extended periods.
- Eclipse attacks: Individual nodes may be surrounded by malicious peers.
- Sybil attacks: Adversaries may spawn many fake identities.
- Proof manipulation: Attackers may attempt to forge or alter Bitcoin proofs.

7.2 Security Guarantees

1. Cryptographic Safety: Invalid Bitcoin proofs are always rejected (deterministically, under bounded resources).
2. Refusal Correctness: Missing or delayed data yields deterministic refusal, not corruption or inconsistency.
3. Economic Resilience: Market forces can correct manipulation attempts through competitive equilibrium where incentives are engaged.
4. Correctness Under Adversity: Cryptographic correctness is preserved under all adversarial conditions, including total data withholding and network partition.

Key distinction: Liveness (eventual verification) may degrade under adversarial conditions, but safety (correctness of verified facts) never does. This separation is fundamental to Zenon’s verification-first architecture and cannot be replicated in execution-first systems where safety depends on liveness. Zenon’s architecture does not claim to enhance Bitcoin’s security; it only permits verification of Bitcoin facts.

7.3 Attack Vectors and Mitigations

Threat Comparison Table:

Attack	Impact Pre-Zenon	Zenon Counter
Data Withhold	SPV blind	Refusal → bounty propagation
Proof Forge	secp256k1 secure	BIP340 native verify
Deep Reorg	L2 unwind all	Local revoke, no Z-chain roll
Eclipse	Partition trust	Offline refusal correct
Sybil	Identity flood	Reputation by delivery
Economic Manipulation	Price manipulation	Audit refusal density

7.3.1 Data Withholding

Attack: Malicious relayers refuse to provide Bitcoin proofs, causing widespread refusals.

Mitigation:

- Refusals are correctness-preserving—nodes remain safe while refusing.
- Economic rewards can attract alternative proof suppliers where participation occurs.
- Redundant proof sources eliminate single points of failure.

Result: Withholding delays verification but cannot corrupt correctness.

7.3.2 Proof Forgery

Attack: Adversary attempts to create fake Bitcoin inclusion proofs or PoW headers.

Mitigation:

- Bitcoin's Proof-of-Work is computationally infeasible to forge.

- Merkle commitments are cryptographically binding.
- Each node independently verifies all proofs under bounded resources.

Result: Forgery is cryptographically impossible without breaking SHA-256 or Bitcoin's PoW.

7.3.3 Sybil Attacks

Attack: Adversary creates many fake identities to manipulate refusal signals or proof distribution.

Mitigation:

- Reputation can be weighted by historical proof accuracy, not identity count.
- Economic incentives reward actual proof delivery, not mere participation claims.
- Redundant verification from independent sources eliminates advantage of multiple identities.

Result: Sybil identities without valid proofs earn zero reward and are ignored.

7.3.4 Eclipse Attacks

Attack: Malicious peers surround a victim node, feeding it false or delayed Bitcoin data.

Mitigation:

- Offline verifiability permits isolated nodes to verify proofs later using portable bundles.
- Nodes can cross-check proofs from multiple sources upon reconnection.
- Deterministic verification ensures that correct proofs always yield consistent results.

Result: Eclipsed nodes refuse until correct data arrives; they never accept invalid proofs.

7.3.5 Economic Manipulation

Attack: Adversary floods network with false refusal signals to manipulate proof pricing.

Mitigation:

- Refusals must reference specific Bitcoin proof IDs and resource bounds (auditable).
- False refusals (for proofs that are actually available) can be penalized through reputation loss.
- Economic model can adjust pricing based on verifiable refusal density, not unsubstantiated claims.

Result: False refusal signals are economically unprofitable and detectable.

7.3.6 Deep Bitcoin Reorganizations

Attack: Adversary attempts to create a longer Bitcoin chain to invalidate previously confirmed proofs.

Mitigation:

- Zenon follows Bitcoin's longest-chain rule (highest cumulative PoW).
- Affected proofs are deterministically revoked via RevocationNotice.
- Full revocation history remains auditable.

Result: Zenon's architecture permits adaptation to Bitcoin reorganizations safely. Attacking Bitcoin's PoW consensus is outside Zenon's threat model—it would require attacking Bitcoin itself. In long-term failure modes like hashrate decline or miner centralization, Zenon cannot corrupt Bitcoin; its failure degrades only usability, leaving Bitcoin fully sovereign.

7.4 Cryptographic Foundations

- Hashing: SHA-256 (Bitcoin), BLAKE3 (Zenon internal structures)
- Signatures: ECDSA (secp256k1, Bitcoin-compatible), Ed25519 (Zenon-internal), Schnorr (for signature aggregation)
- Merkle Proofs: Standard Bitcoin Merkle tree construction, $O(\log n)$ validation
- Proof-of-Work: Bitcoin's PoW (SHA-256d) as external anchor—Zenon does not mine

7.4.1 Future Extensions (Research Directions)

- Post-quantum signatures: SPHINCS+ or XMSS for quantum-resistant verification
- Zero-knowledge proofs: Recursive zk-SNARKs for aggregated Bitcoin proofs
- Quantum-safe commitments: Hash-based commitments resistant to Grover's algorithm

7.5 Formal Security Properties

Zenon's security properties are formally stated to distinguish cryptographic guarantees from economic incentives.

Formal Properties:

Safety (Cryptographic): No invalid Bitcoin proof can be accepted by an honest verifier operating under declared resource bounds.

$$\forall P : \text{verify}(P) = \text{VERIFIED} \Rightarrow \text{valid\_bitcoin\_proof}(P)$$

Holds under: Cryptographic assumptions (SHA-256, ECDSA security), bounded resources.

Does not depend on: Economic participation, proof availability, network connectivity.

Liveness (Economic): Every valid Bitcoin proof is eventually verified, assuming at least one honest proof supplier and rational economic participation.

$$\forall P : \text{valid}(P) \wedge \text{available}(P) \Rightarrow \Diamond\text{verify}(P) = \text{VERIFIED}$$

Holds under: Economic incentives, at least one honest relay, eventual proof delivery.

Does not hold under: Total adversarial data withholding, zero economic participation.

Critical distinction: Safety is an architectural guarantee; liveness is an economic property.

Bounded Determinism: Verification always halts within declared resource limits, producing either VERIFIED or REFUSED.

$$\forall P : \text{verify}(P) \text{ halts} \leq T_{\text{max}} \wedge \text{memory}(P) \leq M_{\text{max}}$$

Refusal Correctness: A refusal indicates either genuine data unavailability or resource bound exceedance—never a false negative for available, valid proofs within bounds.

$$\forall P : \text{verify}(P) = \text{REFUSED} \Rightarrow \neg\text{available}(P) \vee \text{exceeds\_bounds}(P)$$

8 Part VII: Performance and Scalability

8.1 Verification Metrics

Performance measurements from reference instantiation on commodity hardware (4-core CPU, 8 GB RAM):

Operation	Time	Memory	Bandwidth
Bitcoin header validation	0.3 ms	80 B	80 B
Merkle proof verification	0.8 ms	640 B	640 B
Full inclusion proof	1.1 ms	720 B	720 B
Script outcome verification	0.5 ms	200 B	200 B

Table 1: Single-proof verification performance

8.1.1 Batch Verification

Verifying 100 Bitcoin inclusion proofs:

- Sequential: 110 ms
- Parallel (8 cores): 20 ms
- Speedup: 5.5×

8.1.2 Compression Efficiency

Aggregating multiple proofs with shared Bitcoin header paths:

- 10 proofs: 2.3× size reduction
- 100 proofs: 3.1× size reduction
- 1000 proofs: 3.4× size reduction (diminishing returns)

8.2 Network Scalability

Simulation results across varying network sizes (N = node count):

8.2.1 Refusal Convergence

- Initial refusal density: $R_0 = 0.25$ (25% of nodes refuse due to missing data)
- Equilibrium refusal density: $R_{\text{eq}} \approx 0.03$ (3%)
- Convergence time: $\sim \log(N) \times 12$ seconds

Where incentive participation occurs, refusal density can converge to near-zero as proof suppliers respond to economic signals. Without incentive participation, refusal density remains high but correctness is unaffected.

8.2.2 Proof Propagation Latency

Median time from proof generation to global availability:

- 1,000 nodes: 180 ms
- 10,000 nodes: 240 ms
- 100,000 nodes: 310 ms

8.2.3 Bandwidth Requirements

Per-node bandwidth consumption (averaged over 24 hours):

- Minimum verifier: 70 KB/s
- Relay node: 700 KB/s
- Supervisor node: 600 KB/s (higher storage, lower bandwidth due to caching)

8.3 Verification Cost Scaling

Total network verification cost as function of node count N:

$$V(N) = V_0 \times \log(N)$$

Interpretation: As the network grows, per-node verification cost decreases due to proof caching and redundancy, while total network capacity increases logarithmically. This sublinear scaling is enabled by Zenon's refusal-safe architecture—nodes do not need to verify all proofs, only those they care about.

8.4 Comparison with Alternative Architectures

Model	Scaling	Trust	Liveness Req.	Offline
Bitcoin Full Node	$O(N)$	None	Required	No
SPV Light Client	$O(\log N)$	Relayers	Required	No
Lightning Network	$O(\log N)$	Minimal	Required	Limited
Rollup	$O(\log N)$	Sequencer	Required	No
Bridge	$O(1)$	Validators	Required	No
Zenon	$O(\log N)$	None	Optional	Yes

Table 2: Comparison of Bitcoin verification architectures

Key distinction: Zenon's architecture achieves sublinear verification cost without trust assumptions or liveness requirements, while maintaining correctness under adversarial data withholding and network partition. This combination is unique to verification-first, refusal-safe architectures.

9 Part VIII: Economic Model and Incentives

Clarification: Zenon's economic layer permits optimization of proof distribution and reduction of refusal frequency, but does not define correctness. Cryptographic safety holds even if all economic incentives fail. This section describes how incentives can improve verifiability, not how they create it. Economic coordination is for optimization only and does not weaken Bitcoin's security culture; it is optional and does not introduce hidden trust.

9.1 Incentive Structure

Zenon's economic model can create measurable markets for Bitcoin proof delivery.

Economic Formulas:

Verifier Rewards:

$$R_{\text{verifier}} = \alpha \cdot P_{\text{delivered}} + \beta \cdot (1 - R_{\text{local}})$$

where $P_{\text{delivered}}$ = proofs delivered, R_{local} = local refusal rate

Relay Rewards:

$$R_{\text{relay}} = \gamma \cdot B_{\text{served}} + \delta \cdot L_{\text{performance}}$$

where B_{served} = bandwidth served, $L_{\text{performance}}$ = latency performance

Penalties:

$$P_{\text{unreliable}} = -\epsilon \cdot F_{\text{incorrect}} - \zeta \cdot T_{\text{stale}}$$

where $F_{\text{incorrect}}$ = invalid proofs, T_{stale} = stale proofs

Nash Equilibrium: Honest relay dominates (proof cost $<$ reward).

9.1.1 Verifier Rewards

$$R_{\text{verifier}} = \alpha \cdot P_{\text{delivered}} + \beta \cdot (1 - R_{\text{local}})$$

- $P_{\text{delivered}}$: Number of valid proofs delivered
- R_{local}: Local refusal rate (lower = better)
- α, β: Reward scaling parameters

Interpretation: Verifiers can earn more by delivering proofs to high-refusal regions, creating market incentives to fill gaps in proof availability.

9.1.2 Relay Rewards

$$R_{\text{relay}} = \gamma \cdot B_{\text{served}} + \delta \cdot L_{\text{performance}}$$

- B_{served}: Bandwidth served (proof distribution volume)

- $L_{\text{performance}}$: Latency performance (faster = better)
- γ, δ: Reward scaling parameters

9.1.3 Penalties

$$P_{\text{unreliable}} = -\epsilon \cdot F_{\text{incorrect}} - \zeta \cdot T_{\text{stale}}$$

- $F_{\text{incorrect}}$: Incorrect or invalid proofs delivered
- T_{stale}: Stale proofs (superseded by Bitcoin reorgs)
- ϵ, ζ: Penalty scaling parameters

Effect: Unreliable or dishonest suppliers can be economically punished through reputation loss and negative rewards, eventually pricing them out of the market.

9.2 Market Dynamics

9.2.1 Proof Pricing

Proof price can follow supply and demand, adjusted by regional refusal density and proof complexity:

$$\text{Price} = \text{Base\_cost} \times (1 + R_{\text{regional}}) \times S_{\text{complexity}}$$

- Base_cost: Minimum cost to generate/deliver proof
- R_{regional}: Refusal density in target region (higher = more expensive)
- $S_{\text{complexity}}$: Proof complexity factor (e.g., deep reorganizations cost more)

9.2.2 Dynamic Equilibrium

1. High refusal density in a region signals unmet demand.

2. Increased proof prices can attract more suppliers.
3. Competition among suppliers drives efficiency and lowers costs.
4. Equilibrium is reached when marginal cost equals marginal reward.

Assumption: This equilibrium can emerge where rational economic participation occurs. Without participation, refusal density remains high but correctness is unaffected.

9.2.3 Reputation Accumulation

Suppliers can build reputation through:

- Accuracy: Delivering only valid, verifiable proofs
- Timeliness: Responding quickly to refusal signals
- Reliability: Consistent uptime and proof availability

High-reputation suppliers can earn premium rewards and preferential routing. Dishonest actors lose reputation and can be economically excluded over time.

9.3 Economic Sustainability

Zenon's economic layer can be self-funding through refusal reduction:

1. Users pay minimal verification fees to access Bitcoin proofs.
2. Fees flow to proof providers who deliver valid data.
3. Demand drives adaptive pricing based on regional scarcity.
4. Competition drives efficiency, reducing costs over time.

Long-term equilibrium: The system can maintain affordability (low costs), decentralization (many suppliers), and resilience (redundant sources) through market forces, assuming rational participation.

9.4 Economic Failure Acceptance

Total economic failure is an acceptable outcome. Permanent refusal is a valid steady state. Zenon's correctness does not depend on incentives ever working.

10 Part IX: Interoperability Beyond Bitcoin (Non-Core Extension)

Scope note: Parts IX–XI present non-core extensions of Zenon's verification-first architecture to chains beyond Bitcoin. These sections demonstrate architectural generality but are not required to motivate or justify Zenon's design. Bitcoin interoperability alone fully motivates Zenon's architecture. This section is optional for Bitcoin-centric readers.

The mechanisms described here apply the same verification-first principles (bounded verification, refusal safety, offline operation) to other blockchains. However, they introduce additional trust assumptions depending on the target chain's consensus model.

For hostile review: Readers may skip Parts IX–XI without loss of core thesis. Bitcoin-centric reviewers should evaluate Zenon solely on Part I–VIII.

10.1 Ethereum Integration

Zenon's architecture can permit verification of Ethereum state and transactions using bounded Merkle Patricia Trie proofs, enabling verifiable cross-chain facts without execution or centralized trust.

10.1.1 Verification Method

1. Obtain Ethereum block header (including state root).
2. Verify block header against Ethereum consensus (PoS finality checkpoints).
3. Extract Merkle Patricia proof for target state (e.g., account balance, storage slot).
4. Verify proof against state root under bounded resources.

Trust assumption: Ethereum's Proof-of-Stake finality (requires trusting Ethereum's validator set). This differs from Bitcoin verification, which requires only PoW assumptions.

10.1.2 Use Cases

- Verify Ethereum ERC-20 token balances
- Confirm Ethereum contract state transitions
- Enable trustless Ethereum-Bitcoin atomic swaps

10.2 Rollup Compatibility

Zenon's architecture can support stateless verification of zk-rollup proofs and optimistic rollup fraud proofs, enabling verifiable Layer-2 state without continuous synchronization.

10.2.1 ZK-Rollup Verification

1. Obtain zk-SNARK proof of rollup state transition.
2. Verify proof against rollup contract's on-chain commitment (e.g., Ethereum).
3. Confirm commitment finality via Ethereum verification (see above).

Trust assumption: zk-SNARK soundness, Ethereum finality.

10.2.2 Optimistic Rollup Verification

1. Monitor rollup contract for fraud proof submissions.
2. Verify fraud proof validity against disputed state root.
3. Confirm challenge period expiry via Bitcoin or Ethereum timestamp anchors.

Trust assumption: At least one honest challenger (optimistic security model), Ethereum finality.

10.3 Cosmos and IBC

Zenon's architecture can permit verification of Inter-Blockchain Communication (IBC) packet commitments via Tendermint light-client proofs, enabling verifiable cross-chain messaging without direct connectivity.

10.3.1 Verification Method

1. Obtain Tendermint validator set and consensus signatures.
2. Verify block header against validator signatures ($> 2/3$ stake).
3. Extract Merkle proof for IBC packet commitment.
4. Verify proof under bounded resources.

Trust assumption: Honest majority ($> 2/3$) of Tendermint validators. This differs from Bitcoin's trustless PoW model.

10.4 Cross-Chain Predicate Framework

Zenon's architecture can provide a unified verification interface for multiple blockchain types:

- Bitcoin SPV: PoW + Merkle proofs (trustless)
- Ethereum MPT: PoS finality + Merkle Patricia (trust in validators)
- ZK-Rollups: zk-SNARK validity (trust in cryptographic assumptions)
- IBC/Cosmos: Tendermint signatures (trust in validator majority)

Each verification type is handled independently with explicit trust assumptions documented. Users choose verification methods based on acceptable trust trade-offs.

11 Part X: Advanced Applications (Non-Core Extension)

Scope note: This section presents advanced applications enabled by Zenon's verification-first architecture. These are illustrative examples, not core protocol requirements.

11.1 Decentralized Identity

Users anchor identity commitments in Bitcoin via OP_RETURN outputs. Zenon's architecture permits verification of inclusion and revocation proofs, ensuring persistent identity independent of issuer availability.

11.1.1 Process

1. User publishes identity commitment (public key hash) to Bitcoin.
2. Zenon's architecture permits verification of Bitcoin inclusion proof.
3. Identity remains valid until explicitly revoked (via Bitcoin spend).
4. Revocations are verified through Bitcoin transaction proofs.

Properties: No central identity provider, no availability requirements, fully auditable history.

11.2 Supply Chain Verification

Each custody transfer embeds a Bitcoin inclusion proof. Zenon's architecture can permit reconstruction of provenance chains for end-user authenticity checks, operating offline without central registries.

11.2.1 Example: Pharmaceutical Tracking

1. Manufacturer embeds batch ID in Bitcoin transaction.
2. Distributor references previous Bitcoin proof in new transaction.
3. Retailer verifies full provenance chain via Zenon.
4. End consumer scans QR code, verifies chain offline using portable Zenon client.

Advantage: Tamper-proof provenance without trusted intermediaries or online verification services.

11.3 Timestamping and Notarization

Documents are hashed and anchored in Bitcoin via OP_RETURN. Zenon's architecture can maintain verifiable inclusion and timestamp integrity under bounded verification, creating immutable proof-of-existence.

11.3.1 Use Cases

- Legal document notarization
- Patent priority timestamps
- Academic publication proof-of-existence
- Digital media authenticity

Property: Once anchored in Bitcoin, existence proof cannot be forged or backdated (protected by Bitcoin's PoW).

11.4 Decentralized Governance

Vote commitments are embedded in Bitcoin. Zenon's architecture permits verification of inclusion, reveal consistency, and eligibility deterministically—creating censorship-resistant, trustless voting without centralized authorities.

11.4.1 Voting Protocol

1. Eligible voters anchor vote commitments (hashed votes) in Bitcoin.
2. After voting deadline, voters reveal preimages.
3. Zenon's architecture permits verification of commitment-reveal consistency via Bitcoin proofs.
4. Final tally is deterministic and auditable.

Properties: Censorship-resistant (Bitcoin-level), verifiable (cryptographic proofs), auditable (full history retained).

12 Part XI: Research Directions and Open Questions (Non-Core Extension)

Scope note: This section outlines future research areas. These are open problems, not solved components of Zenon's current architecture.

12.1 Bounded Verification as Complexity Class

Define BV (Bounded Verification) as a complexity class:

$$\text{BV} = \{L \mid \exists \text{verifier } V, \forall x \in L : V(x) \text{ halts} \leq T(|x|) \wedge \text{mem}(V) \leq M(|x|)\}$$

Open questions:

- What is the relationship between BV and P, NP, BPP?
- Can bounded verification be proven complete for certain problem classes?
- How does refusal safety affect computational complexity bounds?

12.2 Refusal Topology and Equilibrium Analysis

Research question: Under what economic conditions does refusal density converge to near-zero equilibrium? What are the failure modes?

Approach: Model refusal propagation as a diffusion process on network graphs. Analyze equilibrium stability under varying assumptions (rational actors, Byzantine adversaries, network partitions).

12.3 Thermodynamic Limits of Verification

Hypothesis: There exists a fundamental energy lower bound for verification:

$$E_{\text{min}} \geq k_B \cdot T \cdot \ln(2) \cdot \text{bits\_verified}$$

Where k_B is Boltzmann's constant, T is temperature.

Question: Can Zenon's verification approach this thermodynamic limit? What are the practical efficiency bounds?

12.4 Post-Quantum Adaptation

Challenge: Replace ECDSA and SHA-256 with quantum-resistant primitives without breaking Bitcoin compatibility.

Candidates:

- Signatures: SPHINCS+ (hash-based), Dilithium (lattice-based)
- Commitments: SHA-3 (quantum-resistant hashing)

Open problem: How to migrate Zenon's verification layer to post-quantum cryptography while maintaining backward compatibility with Bitcoin's existing primitives?

12.5 Recursive Proof Aggregation

Goal: Aggregate multiple Bitcoin inclusion proofs into a single, constant-size zk-SNARK.

Benefit: Reduce proof size from $O(N \log M)$ to $O(1)$ where N is number of proofs and M is transactions per block.

Challenge: zk-SNARK generation time currently exceeds verification time—requires cryptographic efficiency improvements.

12.6 Incentive Stability Under Adversarial Conditions

Research question: Does Zenon's economic model remain stable when majority of participants are Byzantine?

Approach: Game-theoretic analysis of refusal-driven markets under adversarial participation, Sybil attacks, and collusion.

12.7 Hardware Acceleration

Goal: Develop ASIC or FPGA implementations of Bitcoin SPV verification optimized for Zenon's bounded verification semantics.

Target: 100× speedup on verification time, enabling real-time verification of thousands of Bitcoin proofs per second on embedded hardware.

12.8 Formal Verification

Goal: Prove correctness of Zenon's verification algorithms using formal methods (Coq, Isabelle, TLA+).

Scope:

- Safety properties (no invalid proofs accepted)
- Liveness properties (eventual verification under assumptions)
- Refusal correctness (deterministic refusal semantics)

13 Part XII: Conclusion

13.1 Core Contributions

1. Bounded Verification: Explicit resource limits (T_{max}, M_{max}, B_{max}) ensure deterministic verification outcomes across heterogeneous hardware—from embedded devices to servers.
2. Refusal Safety: Deterministic refusal as a correctness-preserving outcome, enabled by Zenon's dual-ledger architecture where verification is decoupled from ordering. Execution-first systems cannot refuse safely without risking state divergence.
3. Offline and Asynchronous Operation: Bitcoin proofs can be verified offline using portable data bundles, with results remaining valid indefinitely. This is impossible in systems requiring shared, mutable global states.

4. Economic Self-Regulation: Refusal-driven markets can optimize proof distribution through incentive participation, but correctness never depends on economic success. Cryptographic safety holds even under zero economic participation.
5. Pure Bitcoin Compatibility: No modification to Bitcoin's consensus, incentives, or protocol. Zenon's architecture consumes Bitcoin's finalized outputs (PoW headers, Merkle roots) without participating in Bitcoin's operation. Bitcoin remains unaware of Zenon.

13.2 Architectural Necessity

Critical observation: Each contribution above requires Zenon's specific architectural properties:

- Refusal safety requires decoupled verification and ordering (dual-ledger design).
- Offline verification requires deterministic, static proofs (verification without execution).
- Bounded verification requires explicit resource limits (no unbounded state machines).
- Economic resilience requires separation of incentives from correctness (verification-first).

These properties cannot be added to execution-first blockchains without fundamental redesign. Attempting to port Zenon's mechanisms to Ethereum, rollups, or traditional blockchains would require abandoning their core safety or liveness guarantees.

13.3 Final Thesis

Bitcoin proved that value can exist without trusted intermediaries.

Zenon's architecture proves that Bitcoin truth can be verified without continuous coordination, synchronous execution, or availability assumptions.

Together, they form a verification infrastructure that scales correctness itself—enabling billions of devices to reason about Bitcoin truth under explicit, bounded, and adversarial conditions.

Where Bitcoin guarantees integrity through consensus, Zenon's architecture facilitates verification of that integrity through bounded cryptographic proof. The result: Bitcoin's security model can be verified by billions of intermittently connected devices, without compromising its foundational guarantees. Zenon is architecturally unique and does not enhance Bitcoin; it is downstream and optional.

A Appendix A: Full Performance Benchmarks

Device	CPU	RAM	Storage	Single Proof	1K Batch	Refusal Rate
ESP32	240MHz	4MB	4MB	150ms	180s	15%
RPi 3	1.4GHz 4c	1GB	16GB	25ms	3.5s	12%
RPi 4	1.8GHz 4c	4GB	32GB	15ms	1.8s	9%
RPi 5	2.4GHz 4c	8GB	64GB	12ms	1.2s	8%
iPhone 14	A16	6GB	128GB	10ms	1.1s	6%
iPhone 16	A18	8GB	256GB	8ms	0.9s	5%
Laptop	i5 8c	16GB	512GB	5ms	0.6s	2%
Server	Xeon 16c	64GB	2TB	2ms	0.3s	0%

Table 3: Comprehensive hardware performance matrix

Measurement Methodology:

- Single Proof: Merkle inclusion + header validation
- 1K Batch: Parallelized batch verification with shared header caching
- Refusal Rate: Simulated 10% adversarial withholding scenario
- All measurements: secp256k1 + SHA-256, averaged over 10,000 iterations

B Appendix B: Formal Proof Sketches

Theorem B.1 (Refusal Safety). *Let V be a verifier with bounds $(T_{\max}, M_{\max}, B_{\max})$. For any proof p, if $V(p) =$ REFUSE, then either:*

1. *p is unavailable to V at time t, or*
2. *verify(p) would exceed declared bounds*

Moreover, $V(p) =$ REFUSE never implies p is invalid.

Proof Sketch. By construction, V only returns REFUSE when:

- Data required for verification is missing (unavailable condition)

- Verification would exceed T_{max}, M_{max}, or B_{max}

In both cases, V makes no claim about p's validity. The refusal is a statement about local verification feasibility, not about Bitcoin truth. Global consistency is preserved because Zenon's ordering layer does not require all nodes to agree on verification status—only on transaction ordering. □

Theorem B.2 (Impossibility of Exec-First Refusal)**.** *Any blockchain requiring synchronous state transitions cannot safely implement refusal semantics without introducing fork risk.*

Proof Sketch. Assume blockchain B requires $\forall$ validators: $\text{execute}(\text{state}_{t-1}, tx_t) = \text{state}_t$. If validator v_1 refuses tx_t but v_2 accepts it:

- v_1 state: state_{t-1} (no transition)
- v_2 state: state_t (transition executed)
- Result: Permanent fork

Zenon avoids this because its ordering layer (L_Z) is independent of verification outcomes (L_B). Transactions in L_Z remain ordered even when Bitcoin proofs are refused, preventing fork propagation. □