Zenon Greenpaper Series

Zenon SPV Implementation Guide

Status: Community-authored companion (non-normative, non-official)

Authorship: Zenon Developer Commons (community)

Date: January 4, 2026 (Pre-prototype draft)

This guide is a practical, builder-oriented companion to A Verification-First Architecture for Dual-Ledger Systems. It describes how to implement a resource-bounded verifier (SPV-style) that follows the paper's refusal semantics. Pre-prototype note: numeric targets and economic estimates in this guide are illustrative and parameterized. Replace them with measurements from an implementation and observed network data before treating them as operational guarantees.

Abstract

This document provides a non-normative implementation guide for Zenon SPV verification. It maps the greenpaper's core objects—Momentum headers, the commitment root r_C, and commitment membership proofs of size O(log m)—into concrete interfaces, request/response shapes, caching strategies, and verifier outcomes. The guide emphasizes explicit refusal behavior: if required evidence is missing or exceeds declared bounds, the verifier returns REFUSED rather than guessing. This pre-prototype draft also includes worked examples with concrete resource calculations (bandwidth, computation, storage) to help implementers validate bounds and anticipate refusal rates before committing to development.

1. Scope

Zenon SPV, in the sense used by the greenpaper, is a lightweight verifier that validates stated facts by checking (i) a verified Momentum header chain, (ii) membership of relevant commitments under the per-header commitment root r_C, and (iii) account-chain segment integrity, all under explicit resource bounds.

1.1 Goals

- Independent verification: no trusted intermediary is required to validate the evidence that is presented.
- Bounded operation: the verifier enforces explicit limits on storage, bandwidth, and computation.
- Refusal semantics: insufficient evidence yields REFUSED, not probabilistic acceptance.
- Offline resilience: a verifier can resume from its last verified header and cached proofs.
- Security-first operation: mitigate light-client isolation risk via multi-source evidence and randomized peer selection.
- Transparent trade-offs: present explicit security, economic, and refusal assumptions without overstating claims.

1.2 Non-Goals

- Re-executing a global VM or reconstructing global state from genesis.

- Maintaining mempool state or participating in block production.
- Accepting statements without the paper's required evidence, even if "likely true."

2. Trust Roots and Evidence

Zenon SPV is anchored to a genesis trust root. From this root, the verifier accepts additional evidence only if it is cryptographically bound to the verified header chain and within declared limits.

2.1 Genesis Trust Root

A verifier MUST ship with (or be configured with) the genesis trust root for the Momentum header chain, including the genesis header hash and any consensus parameters required to validate successor headers as specified by the greenpaper.

2.2 Header Authenticity vs. Consensus Confidence

Header anchoring and hash linkage provide authenticity for the headers a verifier has obtained (i.e., that the verifier is checking membership proofs against an authentic r_C). Stronger confidence against reorgs requires additional header-chain evidence within a bounded policy window w. If the verifier cannot obtain sufficient window evidence under its bounds, it SHOULD return REFUSED, consistent with the paper's refusal semantics.

2.3 Policy Window Instantiation (Non-Normative)

Implementers need a concrete rule to instantiate the abstract "finality function" implied by the greenpaper. This guide treats the policy window w as a risk parameter chosen by the verifier operator, not a hard protocol constant.

Definition (policy finality check): Let k be the number of consecutive verified Momentum headers after height h (i.e., the evidence window depth). Define a policy function $f_{consensus}(k)$ = ACCEPT if $k \geq w$, otherwise REFUSED.

Policy window recommendations (illustrative; calibrate with observed reorg data):

- Low-value queries (UI previews): $w \geq 6$ headers (~1 minute if 10s cadence).

- Medium-value (payments, routine ops): $w \geq 60$ headers (~10 minutes).
- High-value (bridges/exchanges): $w \geq 360$ headers (~1 hour) or higher.

These tiers are placeholders until (a) a public reorg monitor exists for Zenon mainnet, and (b) prototypes report refusal rates under realistic peer sets. Implementations SHOULD log observed reorg depths and publish the distribution (max depth, p95 depth) to justify w.

2.4 Privacy Note (Non-Normative)

Query patterns can leak which accounts or commitments a verifier is interested in. Implementations SHOULD consider privacy-preserving retrieval (batching, randomized retrieval, or filter-style requests) so proof fetching does not trivially reveal the verifier's interests to a single peer.

2.5 Weak Subjectivity Considerations (Non-Normative)

Long-range attacks are a general risk for stake-based systems when a verifier has been offline for extended periods. If a verifier is offline for long horizons (e.g., many months to ~1 year), additional checkpoints MAY be required to prevent acceptance of a fabricated long-range fork that starts near genesis.

Operational recommendation: ship periodic checkpoint pairs (Momentum height, header hash) via software updates (e.g., quarterly). The verifier can treat the most recent embedded checkpoint as an additional trust anchor for long-offline recovery. This is compatible with refusal semantics: if no acceptable checkpoint is available for a long-offline verifier, it SHOULD REFUSE rather than guess.

3. Data Objects

3.1 Momentum Header (Verifier-Required Subset)

A SPV verifier does not require full node state, but it does require a stable subset of each Momentum header sufficient to: (i) link the header chain, (ii) validate the header under the consensus rules referenced by the paper, and (iii) extract the commitment root r_C for commitment membership checks.

Minimum fields (conceptual): height, prev_hash, header_hash, r_C, and consensus/meta fields required by the header validity predicate.

3.2 Commitment Membership Proof under r_C

For each commitment c referenced by a statement or account segment, the prover MUST supply a membership proof π_C that verifies c is included in the set committed under the corresponding Momentum root $r_C(h)$. The proof size is O(log m) where m is commitments per Momentum block.

Merkle branch sizing rule of thumb: If the membership proof is a Merkle branch with 32-byte hashes and tree depth $d = \text{ceil}(\log_2(m))$, then $\sigma_\pi \approx 32 \cdot d$ bytes (plus a small encoding overhead). Example: $m \approx 10^4 \Rightarrow d \approx 14 \Rightarrow 448$ bytes of hash material.

3.2.1 Merkle Tree Structure Assumption (Non-Normative)

This guide assumes a binary Merkle tree with SHA-256 hashes (32 bytes per node) for membership proofs under r_C. If Zenon uses a different authenticated structure (e.g., a Merkle Patricia Trie or another accumulator), adjust the depth and branch-length calculations accordingly. For comparison: Ethereum’s Merkle Patricia Trie has variable branch length; use measured branch sizes from real proofs.

3.3 Account-Chain Segment Bundle

An account-chain segment bundle for account A consists of a contiguous range of account blocks $B_A[i..j]$, together with all proof objects needed to bind each block (or each referenced commitment) to an authenticated Momentum root r_C at the appropriate height.

3.4 Resource Parameters

The guide uses the following parameters for concrete accounting. Implementations SHOULD measure them on target platforms and publish the measurements along with the exact implementation version.

- σ_H: average bytes per Momentum header (verifier-retained subset).
- σ_B: average bytes per account-chain block.
- σ_π: average bytes per proof object (e.g., Merkle branch bytes).

- w: policy window in Momentum headers for consensus confidence (Section 2.3).
- C_{verify}: per-proof verification cost (bounded by C_V).

4. Verifier Interfaces

The following interfaces are non-normative but are designed to map directly onto the greenpaper's verification predicates. Each interface returns one of three outcomes: ACCEPT, REJECT, or REFUSED.

4.1 Common Outcomes

- ACCEPT: the statement is verified under declared bounds.
- REJECT: evidence is present but invalid (broken linkage, failed cryptographic checks).
- REFUSED: evidence is missing, incomplete, or exceeds bounds; the verifier does not guess.

4.2 Header Verification

Verify a sequence of Momentum headers extending a locally trusted base. Implementations SHOULD obtain evidence from multiple sources where feasible and SHOULD enforce a policy window w appropriate to the value at risk.

```
VerifyHeaders(H[k..n], header_state) -> {ACCEPT, REJECT, REFUSED} +
header_state'
```

4.3 Commitment Membership Verification

```
VerifyCommitment(r_C(h), c, π_C) -> {ACCEPT, REJECT, REFUSED}
```

4.4 Account-Chain Segment Verification

```
VerifyAccountSegment(A, B_A[i..j], proofs, header_state) ->
{ACCEPT, REJECT, REFUSED}
```

A segment is ACCEPT only if (i) the account chain links correctly, (ii) every required proof verifies, and (iii) every referenced commitment is proven to be a member under an authenticated $r_C(h)$ whose header is within the verifier's retained evidence window.

5. Verification Flows

The table below summarizes the primary SPV verification flows and expected outcomes.

Flow	Steps	Expected outcome
Bootstrap	Load genesis; fetch window H[g..g+w]; VerifyHeaders; store retained window.	ACCEPT if within bounds; REFUSED if window evidence cannot be obtained.
Account event	Fetch segment+proofs; ensure headers cover heights; verify memberships; verify linkage.	ACCEPT with valid proofs; REJECT on invalid linkage/proofs; REFUSED on missing evidence.
Offline resume	Load cached state; fetch successors; re-verify window; continue.	REFUSED if required historical evidence cannot be re-acquired under bounds.

6. Resource Accounting

This section provides a concrete, parameterized model for bandwidth, computation, and header storage. The intent is to let implementers pick explicit bounds and then check whether a requested verification fits those bounds.

6.1 Core Bounds

Bandwidth(account segment) $\in O(L_A(\sigma_B + \sigma_\pi))$.

Computation(account proofs) $\in O(L_A \cdot C_{verify})$, where C_{verify} is per-proof verification cost bounded by C_V.

Policy window storage (header retention) $= w \cdot \sigma_H$.

6.2 Concrete Parameter Table (Pre-Prototype)

Until a reference implementation publishes measurements, treat the values below as parameter choices to run scenarios. Replace the “Basis / notes” column with real data (mainnet sampling + WASM/mobile benchmarks) as soon as prototypes exist.

Metric	Conservative estimate	Aggressive estimate	Basis / notes
σ_H (Momentum header bytes)	800 B	1.5 KB	Sum of retained fields: prev_hash(32B), height(~8B), rC(32B), signatures /metadata (dominant). Measure from real headers.
σ_B (account block bytes)	500 B	2 KB	Measure from real account blocks; depends on signatures/fields.
σ_π (Merkle branch bytes)	416 B (d=13)	640 B (d=20)	≈ 32 · d bytes (+ overhead). $d=\mathrm{ceil}(\log_2 m)$.
w (policy window, headers)	12	60	Calibrate to consensus confidence; see Section 2.3 tiers.
C_{hash} (ms per hash op)	0.1–1 ms/hash	≈ 5 ms/hash	Benchmark in WASM/mobile; depends on implementation and device class.
C_{verify} (per membership)	$d \cdot C_{hash}$	$d \cdot C_{hash}$	For Merkle membership: verify d hashes; add signature checks as needed.

Example window storage: if w=60 and σ_H=1 KB, then retained-window storage is ~60 KB, which is typically trivial for mobile and browser clients.

6.3 Worked Example: Verifying a 100-Block Wallet History

Assume the prover supplies L_A=100 account blocks plus one membership proof per block under r_C. For a 10-second Momentum cadence and ~100–10,000 active accounts per Momentum, a realistic commitment count is $m \approx 10^3$–10^4 (e.g., $m \approx 8000$), giving $d = \mathrm{ceil}(\log_2 m) \approx 13$. This avoids unrealistic depths (e.g., $d=30$ would imply ~1 billion commitments).

```
Example parameters (illustrative; measure and replace):
L_A = 100 blocks
σ_B = 500 bytes (conservative)
m ≈ 8000 commitments per Momentum → d = ceil(log2 m) = 13
σ_π ≈ 32 × 13 = 416 bytes (Merkle hash material; encoding overhead
omitted)
C_hash = 1 ms/hash (mobile/WASM mid-range target)

Bandwidth:
100 × (500B + 416B) = 91,600B ≈ 92KB

Computation (Merkle membership only):
100 × (13 hashes) × (1 ms/hash) = 1300 ms = 1.3 s

Verdict (example policy):
If mobile compute budget is 2 s/query and bandwidth budget is
1–3MB/query,
this query is ACCEPT (fits bounds).
```

This worked example is intentionally concrete: report your own tuple (device, runtime, L_A, m, d, proof type, p50/p95 latency) in conformance reports. If your proof structure differs from a binary SHA-256 Merkle tree, compute d and σ_π from measured proof objects instead.

7. Refusal Cookbook

The table below summarizes common conditions and the correct verifier outcome.

Condition	**Outcome**	**Reason**
Missing header for referenced height h	REFUSED	Cannot bind proof to an authenticated $r_C(h)$.
Missing membership proof π_C	REFUSED	Evidence is insufficient within declared bounds.
Proof present but fails verification	REJECT	Evidence is invalid cryptographically.

Policy window not satisfied (k<w)	REFUSED	Consensus confidence requirement not met under chosen policy.
Inconsistent header views across peers	REFUSED	Multi-source evidence disagrees; refuse until consistent view is established.
Segment exceeds bandwidth/compute bounds	REFUSED	Verifier must respect declared constraints.

8. Conformance Testing

An implementation SHOULD provide deterministic tests for the following cases:

- Valid header chain extension within policy window.
- Broken header linkage (prev_hash mismatch) → REJECT.
- Valid commitment membership proof under r_C.
- Invalid membership proof → REJECT.
- Missing proof or missing required headers → REFUSED.
- Account segment with correct linkage and bindings → ACCEPT.
- Inconsistent peer responses for the same header range → REFUSED (until reconciled).

9. Security Considerations (Non-Normative)

- Isolation/eclipsing: diversify peers; randomize selection; require multi-source confirmation for critical header ranges.
- Reorg policy: bind a risk tier to w, and log observed reorg depths to justify w.
- Availability failure: treat missing proofs as REFUSED; design UX around progressive verification and caching.
- Privacy leakage: avoid single-peer “tell me everything about account A”; prefer batching or randomized retrieval.

9.1 Adversarial Scenario Analysis (Non-Normative)

The table below summarizes common adversarial scenarios, the verifier's correct response under refusal semantics, and implementation-level mitigations.

Attack	**Adversary goal**	**Verifier response**	**Mitigation cost**
Eclipse / isolation	Isolate verifier; serve biased headers/proofs	REFUSED (multi-peer disagreement) or REFUSED (insufficient sources)	Require k≥3 peers; randomize; rotate
Proof withholding	Force REFUSED by not serving proofs	REFUSED (data unavailable)	Cache + fallback peers; proof relays
Equivocation / fork split	Serve two valid forks to confuse	REFUSED until k≥w on a consistent chain	Increase w for high-value queries
Invalid-proof flooding (DoS)	Waste verifier compute	REJECT (fast-fail) + rate-limit	Per-peer quotas; ban on repeated failures
Long-range attack	Rewrite history far back	REFUSED or REJECT depending on checkpoint policy	Periodic checkpoints (Section 2.5)

```
def FetchWithRedundancy(query, peers, k=3):
# Fetch the same object from k randomly sampled peers
sample = RandomSample(peers, k)
responses = [FetchFromPeer(p, query) for p in sample]
if not AllAgree(responses):
return REFUSED # suspected isolation or fork
return responses[0]
```

10. Implementation Checklist for Prototype Builders

- [ ] Measure σ_B, σ_π, and σ_H from testnet/mainnet samples.
- [ ] Benchmark C_verify on target platforms (browser WASM, mobile native).

- [ ] Implement multi-peer header fetching with consistency checks (Section 9.1).
- [ ] Log refusal rates by category (DATA_UNAVAILABLE vs COST_EXCEEDED vs WINDOW_NOT_MET).
- [ ] Test simulated network partition (should return REFUSED, not REJECT).
- [ ] Validate policy window w against observed reorg data (Appendix D.4).
- [ ] Publish a conformance report: (device, runtime, p50/p95 latency, refusal rate, parameter values).

Appendix A. Reference Pseudocode (Non-Normative)

The pseudocode below sketches the control flow implied by the interfaces above, including basic hardening (multi-peer header acquisition) and explicit refusal propagation.

```
function VerifyAccountEvent(A, query, opts):
    segment, proofs = FetchAccountSegment(A, query.range)
    if segment missing or proofs missing:
        return REFUSED

    heights = ExtractReferencedHeights(segment, proofs)

    # Ensure policy window coverage
    if not HeaderStateCovers(heights, w=opts.w):
        headers = FetchHeadersToCover(heights, peers=opts.peers,
            min_sources=2)
        outcome = VerifyHeaders(headers, header_state)
        if outcome != ACCEPT:
            return outcome

    # Verify per-commitment membership under authenticated r_C(h)
    for each commitment c in ExtractCommitments(segment, proofs):
        h = HeightForCommitment(c)
        rC = header_state.rC[h]
        πC = proofs.membership[c]
        if πC missing:
            return REFUSED
        if VerifyCommitment(rC, c, πC) != ACCEPT:
```

```
return REJECT

if not VerifyAccountChainLinkage(segment):
return REJECT

return ACCEPT
```

Appendix B. Optional Succinct Proof Integration (Non-Normative)

Some deployments may prefer to replace per-commitment Merkle membership proofs with a succinct proof system that attests membership for one or more commitments under r_C. This can reduce transmitted proof material per membership, depending on batching strategy and proof system. These choices are outside the greenpaper's normative core: treat them as optional enhancements and publish measured costs (proof size, verification latency, prover overhead) on target platforms.

Appendix C. Proof Availability Analysis (Non-Normative, Pre-Prototype)

The greenpaper's refusal semantics make availability a first-class concern: missing evidence yields REFUSED. This appendix provides a parameterized and estimate-driven way to reason about refusal rates and the economics of serving historical proofs. Treat the numbers as order-of-magnitude guidance; replace them with measured mainnet data.

C.1 Archival Cost Estimation (Desk Research; Illustrative)

Assume a 10-second Momentum cadence. Then blocks/day = 8640. Over ~3 years, Momentum headers ≈ 3 · 365 · 8640 ≈ 9.5M headers. If $\sigma_H \approx 1$ KB retained/header, header retention for full history is ~9.5 GB.

A conservative archival budget might include: (i) headers (~10 GB), (ii) account-chain history (~10 GB), and (iii) historical proof bundles (~50 GB), for a total on the order of ~70 GB. This is deliberately coarse; measure real sizes once proof formats are finalized.

Using AWS S3 list pricing as a rough baseline, S3 Standard storage is commonly quoted at ~$0.023/GB-month for the first 50 TB, and data transfer out to the public internet is commonly priced around ~$0.09/GB for the first 10 TB/month (tiered thereafter). Costs vary by region and service; treat these as illustrative.

```
Illustrative monthly cost (order-of-magnitude):
Storage: 70 GB × $0.023/GB-month ≈ $1.61/month
Egress: 65 GB/month × $0.09/GB ≈ $5.85/month
Total: ≈ $7.50/month per archival endpoint
```

If a proof-serving operator receives meaningful ongoing rewards, the marginal cost of serving tens of GB and moderate egress is plausibly small. However, real economics depend on workload (QPS), proof bundle sizes, and the number of independent archival operators.

C.2 Refusal Rate Projections (Model-Based; Illustrative)

Expected REFUSED rates depend on how many peers serve the required historical evidence. As a simple model, let p be the fraction of peers that can serve the required age of data; then the probability that a verifier sampling k peers can find at least one serving peer is 1 - (1-p)^k. This suggests that multi-peer retrieval can reduce REFUSED rates when p is non-trivial.

Illustrative projections (replace with measured network data):

- Fresh queries (<1 hour): refusal <1–2% if data is widely cached.
- Recent (<30 days): refusal ~2–10% depending on pruning policies.
- Older (>1 year): refusal ~40–60% if only a minority of operators serve deep history.

Mitigation: implement proof caching at the client. Even a ~1 GB cache of validated proof bundles can materially reduce repeat-query refusals. Also consider optional proof-relay endpoints (CDN-style) that serve immutable bundles; verification remains end-to-end.

Appendix D. dPoS Reorg Analysis (Desk Research; Non-Normative)

D.1 Threat Model for Scheduled dPoS

Zenon Phase 0 is commonly described as scheduled block production under delegated proof-of-stake (dPoS). Primary reorg vectors include: (a) stake-weighted collusion or key compromise sufficient to rewrite recent history, and (b) network partition (or severe propagation delay) causing temporary forks.

Stake attack (illustrative cost model): If an adversary needs to control >50% of consensus weight, then the capital cost scales with the amount of staked ZNN required. Let N be the number of active producers and S be average effective stake per producer; then a crude lower bound for attack capital is ~0.5 · N · S. Multiply by market price to estimate fiat cost. These inputs MUST be replaced with real stake distribution data.

D.2 Policy Window Justification (Comparative Heuristics)

In dPoS-style systems, longer confirmation windows reduce the probability that a temporary fork or coordinated producer set can successfully reverse a statement after it is considered final. As a non-normative heuristic (not a guarantee):

- w=6 provides “fast UI” confidence for low-value queries.
- w=60 provides medium-value confidence and aligns with “minutes-scale” settlement windows used operationally in other systems.
- w=360 provides high-value conservatism for bridges and exchange operations.

D.3 Conservative Safety Model (Toy Model; Illustrative)

Define a toy policy risk function: $f_{consensus}(k)$ = P(stake_attack) + P(partition_k). As an illustrative model, assume P(partition_k) ≈ exp(-λk) for some resilience factor λ>0. This is not a proof; it is a placeholder to motivate why larger w improves safety under plausible network behavior.

```
Example (illustrative only):
Assume P(stake_attack) < 1e-6 (cost-prohibitive)
Assume λ = 0.1

For w = 60:
f_consensus(60) ≈ 1e-6 + exp(-6) ≈ 0.0025
```

D.4 Measurement Mandate

These estimates MUST be replaced with historical mainnet data: (i) maximum observed reorg depth, (ii) p95 confirmation latency, (iii) stake distribution metrics (e.g., concentration/Gini), and (iv) observed fork resolution behavior under network stress. Production deployments SHOULD publish this telemetry to justify w in a transparent way.

Appendix E. Comparative Analysis (Desk Research; Non-Normative)

The table below situates Zenon SPV relative to several light-client / verification models. Numbers are approximate and depend heavily on implementation and configuration; treat them as directional.

System	Verification model	Proof size	Verification time	DA assumption	Trust model
Zenon SPV (this guide)	Bounded verification + explicit refusal	O(log m) Merkle branch (~0.4–1.5KB typical)	Merkle verify: d hashes (e.g., 13 hashes)	Archival operators (Pillars/Nodes) serve proofs	Genesis trust root + header chain + policy window
Bitcoin SPV (BIP37)	Header chain + Merkle inclusion	O(log n) branch (~0.3–1KB)	Few–tens of hashes	Full nodes serve data (often altruistic)	Genesis block + PoW chain
Ethereum Portal (state/light data)	Gossip/DHT-style retrieval; state proofs vary	Variable (depends on proof type)	Variable (often higher than pure Merkle)	DHT participants; incentives evolving	Often relies on checkpoi nting / weak sub jectivity
Mina	Recursive proof / succinct chain	Succinct proof elements (order of KB); chain state ~22KB	SNARK verification (imp lementation-dep endent)	Provers funded by protocol rewards	Genesis proof + S NARK-ve rifiable state
zkSync Lite	Validity proof posted to Ethereum L1	SNARK proof posted per batch (size depends on system)	Client checks L1 inclusion; proof verify done on L1	Ethereum calldata / contract state availability	L1 smart contract security

Trade-offs: Zenon SPV's advantage is explicit refusal semantics and minimal cryptographic complexity (Merkle-style proofs) relative to recursive-proof systems. A key disadvantage is that data availability depends on economically motivated proof serving, not a protocol-enforced DA layer like Ethereum calldata; this pushes design attention to incentives, caching, and refusal UX.

Consistency check: This revision removes unrealistic Merkle depths (e.g., d=30) from worked examples, uses consistent units (B/KB/GB), and standardizes subscripts for σ_H, σ_B, σ_π throughout.