Header-Only Verification and the State Commitment Invariant

Research Note — Non-Normative, Proof-Oriented

Abstract

This document formalizes the minimum verification invariant required for a browser-native or light-client system to securely track blockchain state without executing transactions or storing full state. We show that header-only verification, when combined with cryptographic state commitments and bounded inclusion proofs, is sufficient to guarantee correctness under standard consensus safety assumptions. We further demonstrate how this invariant applies naturally to Zenon's Momentum-based architecture and generalizes beyond Bitcoin SPV. This note establishes the theoretical foundation without prescribing specific cryptographic primitives or protocol implementations.

1. Problem Statement

A full node verifies a blockchain by:

- Executing all state transitions
- Storing the full state
- Verifying every transaction

A light client cannot afford this.

Key insight: Execution replay is a sufficient but not necessary condition for correctness. A verifier can establish state validity without re-executing transitions if cryptographic commitments provide an equivalent guarantee.

The question is:

What is the minimum information a verifier must process to ensure it is following the correct chain and correct state evolution?

2. Definitions

Let:

- H_i be the header at height *i*
- S_i be the canonical state after applying all valid transitions up to height *i*
- $C_i = \text{Commit}(S_i)$ be a cryptographic commitment to the full state
- π_x denote a proof relative to query *x* under commitment C_i

We assume:

- A collision-resistant hash function Hash
- A binding state commitment scheme (defined below)
- A prefix-monotonic chain selection rule (defined below)

2.1. Commitment Scheme Specification

Let **Commit** be a binding, collision-resistant state commitment scheme supporting succinct membership and non-membership proofs. Examples include Merkle roots, Verkle roots, or polynomial commitments. The specific scheme is orthogonal to the invariant and may vary by implementation. We require only that:

- **Binding:** Given $C = \text{Commit}(S)$, it is computationally infeasible to find $S' \neq S$ such that $C = \text{Commit}(S')$
- **Succinct:** $|C|$ and proof sizes are sublinear in $|S|$
- **Verifiable:** Proofs can be checked without access to S
- **Canonical encoding:** Commit(S) must be derived from a unique, deterministic serialization of S

2.2. Chain Selection Formalism

We assume the chain selection function is **prefix-monotonic** and **fork-resolving**: extending the canonical chain strictly increases weight under standard consensus safety assumptions. Formally, if chain C' extends chain C, then:

```
Weight(C') > Weight(C)
```

This property holds for proof-of-work (cumulative difficulty), proof-of-stake (cumulative stake-weight), and momentum-based consensus (pillar quorum progression).

3. The Header-Only Verification Invariant

A verifier does not need to know S_i if it can verify:

$$[H_i \text{ is valid} \land C_i = \text{Commit}(S_i) \land C_i \text{ was agreed upon by consensus}]$$

Invariant (quotable form): A verifier need not execute transitions if consensus commits to a binding representation of post-transition state.

This is the core invariant.

Informally:

If the header commits to the state, and the header is valid, then the state is valid — even if the verifier never sees the state.

Lemma (State Commitment Consistency): Assuming deterministic state transition rules, if $C_i = \text{Commit}(S_i)$ and $C_{i+1} = \text{Commit}(S_{i+1})$, and H_{i+1} is valid with respect to H_i, then S_{i+1} is a deterministic function of S_i under the protocol rules.

Proof sketch: By construction, valid headers enforce state transition determinism. If two distinct states S_{i+1} and S'_{i+1} could both be valid successors of S_i, this would violate the binding property of Commit or the determinism of the protocol. Therefore, consensus on C_{i+1} implies consensus on the unique valid S_{i+1}. ■

4. What the Verifier Must Check

A header-only verifier checks:

1. **Chain linkage:** $\text{Hash}(H_{i-1}) \in H_i$
2. **Consensus validity:** Proof-of-Work, quorum signature, momentum rule, etc.
3. **State commitment consistency:** $C_i \in H_i$
4. **Chain selection:** $\arg\max_{\text{chain}} \text{Weight}(\{H_0 \ldots H_n\})$

No transaction execution is required.

5. Inclusion Without Execution

A verifier issues a **state query** against S_i, receiving a proof π attesting to membership, non-membership, or value correctness under commitment C_i. Formally:

$$\pi = \text{Prove}(x \in S_i) \text{ or } \pi = \text{Prove}(x \notin S_i)$$

The verifier checks:

$$\text{Verify}(C_i, x, \pi) = \text{true}$$

Examples of state queries: "UTXO u exists", "account balance = 100", "contract storage slot s has value v".

5.1. Proof Type Distinction

The proof π may attest to different claims depending on the state model:

- **Membership:** "x exists in state" (e.g., UTXO presence)
- **Non-membership:** "x does not exist in state" (e.g., double-spend prevention)
- **Value correctness:** "x has value v" (e.g., account balance = 100)
- **Predicate satisfaction:** "x satisfies predicate P" (e.g., sufficient balance for transfer)

In practice, π may be structured as $\pi = (\pi_{\text{member}}, \pi_{\text{value}})$ to separately attest to existence and properties.

This decouples:

- **State storage** $\rightarrow$ held by full nodes or users
- **State verification** $\rightarrow$ done by anyone with access to a prover

6. Application to Bitcoin SPV

Bitcoin SPV is a special case where:

- C_i = Merkle root of transactions (not full state)
- Proofs are Merkle inclusion proofs
- State = UTXO set (implicit, not directly committed)

Bitcoin SPV works despite not committing to full state, which is why it has known limitations (e.g., inability to verify total supply without full validation). **Note:** Bitcoin commits to transaction ordering via the Merkle root, not to the resulting UTXO state directly.

Zenon improves on this by:

- Allowing explicit state commitments
- Separating execution from verification
- Enabling richer proof systems beyond Merkle trees

7. Application to Zenon (Momentum Architecture)

In Zenon:

- Momentum headers already form a canonical sequence
- State transitions are deterministic
- Execution is not globally shared via a VM

We can define:

$$C_i = \text{Commit}(\text{Account-Chains, Plasma State, zApp Interfaces})$$

The commitment need not expose individual components; it may bind an abstracted state delta or frontier representation, provided the mapping from protocol rules to committed state is deterministic. The specific serialization format and **granularity of commitment is an implementation choice** orthogonal to the invariant.

A browser light client only needs:

- Momentum headers
- Consensus validation
- Proofs tied to C_i

8. Why Global Execution Is Not Required

Traditional blockchains assume:

$$\text{Verification} \equiv \text{Execution}$$

But cryptographic commitments allow:

$$\text{Verification} \supset \text{Execution}$$

Execution becomes:

- A service (provided by full nodes or specialized provers)
- A proof generator (not a consensus requirement)
- Separable from verification (enabling stateless clients)

This separation is the foundation of SNARK-based systems, STARK rollups, and stateless client architectures.

9. Security Assumptions and Adversarial Model

9.1. Trust Assumptions

This model assumes:

- Honest majority of consensus participants (typically > 50% or > 2/3 depending on protocol)
- Soundness and binding of the commitment scheme
- Availability of at least one honest prover

These are identical to full-node assumptions, minus storage and execution costs.

9.2. Prover Availability Clarification

Unlike full-node models, the verifier does not assume persistent access to a *specific* prover, only that at least one honest prover exists somewhere in the network at the time a proof is requested. The verifier may query multiple provers and accept any valid proof. This is a weaker assumption than requiring the verifier to run a full node itself.

Correctness vs. Liveness: This model guarantees *correctness* assuming availability of at least one honest prover. Liveness and data availability are orthogonal concerns addressed by replication, caching, or economic incentives, not by the verification invariant itself. A verifier may fail to obtain a proof (liveness failure) without compromising security (correctness guarantee). **Critically:** proof withholding does not equal proof forgery—an adversary who withholds proofs causes liveness failure but cannot produce false proofs that pass verification.

9.3. Header Availability Model

Header availability is not a consensus problem and is not specific to any particular verification architecture. Any verifier using header-only verification (including Bitcoin SPV clients) must address eclipse resistance and data sourcing. This invariant neither improves nor worsens these assumptions; it simply establishes the correctness guarantee given header availability. Header acquisition strategies (bonded relayers, P2P sync, multi-source aggregation) are therefore implementation concerns, not cryptographic or ledger-semantic limitations of the verification model itself.

9.4. Adversarial Capabilities

The adversary may:

- Control network scheduling and message delivery
- Withhold proofs or delay responses

- Attempt to present alternative state claims

However, the adversary **cannot**:

- Forge valid headers without controlling consensus majority
- Break the binding property of Commit
- Produce valid proofs for false claims

10. Implications

This invariant enables:

- Browser-native nodes (no server-side execution required)
- Header-only sync (sublinear bandwidth)
- Stateless verification (constant memory)
- Bounded proofs (deterministic resource limits)
- Distributed proof generation (decentralized infrastructure)

It also explains why Zenon fits a fifth architectural class: *deterministic-interface, proof-first systems* — distinct from account-based VMs, UTXO chains, state channels, and rollups.

11. Summary

A verifier does not need to execute state transitions if it can verify state commitments embedded in consensus-validated headers. This separation of verification from execution is the foundation upon which Bitcoin SPV, Zenon light clients, and browser-native verification are built.

Security Theorem (Informal): Security is bounded by the minimum of header-chain finality and consensus safety, without amplification of either attack surface. **The system is non-amplifying:** compromising one security domain does not weaken the other. Formally, if the header chain provides security $S_{headers}$ and consensus provides security $S_{consensus}$, then the combined system achieves security $\min(S_{headers}, S_{consensus})$. An adversary must compromise at least one security domain; compromising both provides no additional attack capability beyond compromising the weaker domain.

The invariant presented here is general and does not prescribe specific cryptographic primitives, consensus mechanisms, or proof systems. It establishes the *minimum* correctness requirements for any header-only verification scheme.

12. Next Research Directions

- Minimal state frontier proofs (what is the smallest witness set?)
- Bounded inclusion without Merkle trees (polynomial commitments, accumulators)
- Deterministic proof size limits (worst-case complexity bounds)
- Proof amortization across momentums (batch verification)
- Supervisor / sentry proof delegation (trust-minimized watchtowers)
- Cross-chain header verification (bridge security models)
- Dynamic commitment scheme upgrades (migration without hard forks)

13. Non-Goals and Limitations

This note intentionally **does not**:

- Specify concrete cryptographic primitives or hash functions
- Propose a final protocol or implementation
- Address network-layer concerns (peer discovery, routing)
- Define incentive structures for proof providers
- Compete with or replace full protocol specifications

These are important engineering questions, but they are orthogonal to the core invariant. This restraint is deliberate: the invariant should remain stable even as implementation details evolve.

13.1. What Header-Only Verification Does Not Do

Header-only verification as defined here does **not**:

- Execute state transitions of the verified chain
- Validate application-specific logic or scripts
- Track complete state sets (e.g., UTXO sets, account balances)
- Enforce policy rules of the external chain
- Resolve forks or consensus disputes globally

It verifies *only* that a commitment C_i to some state S_i was agreed upon by the consensus mechanism of the chain being verified, and that proofs against C_i can be checked without access to S_i. Application-specific semantics and execution are separate concerns.

13.2. Relationship to Application Implementations

This invariant serves as a theoretical foundation upon which specific verification systems (such as Bitcoin SPV, Ethereum light clients, or cross-chain bridges) can be built. Application-specific implementations will add:

- Concrete commitment schemes (Merkle trees, Verkle tries, etc.)
- Specific proof formats and verification procedures
- Application semantics (UTXO validation, account balance checks, etc.)
- Economic incentives and operational infrastructure
- Network-layer protocols for header distribution

These additions do not change the core invariant—they instantiate it with specific choices appropriate to their use case.

Acknowledgments

This work builds on foundational concepts from Bitcoin SPV (Nakamoto, 2008), stateless client research (Ethereum Foundation), and cryptographic accumulator theory. The formalization is intended to clarify the verification model underlying Zenon's Momentum architecture.