HOSTILE TECHNICAL REVIEW
Bounded Inclusion Without Merkle Trees

Adversarial Implementation Analysis • December 2025

1. INSTANTIATION SKETCH

Data Structures

```
// Header commitment struct Header { bytes32 state_root; // R_i = hash(S_i) uint256
resource_bound; // β_i bytes quorum_sig; // σ_i (BLS aggregate or threshold) uint64 height;
bytes32 prev_header_hash; } // State proof (Merkle-Patricia) struct StateProof { bytes32
account_key; // keccak256(address) bytes rlp_account; // RLP(nonce, balance, storage_root,
code_hash) bytes[] merkle_path; // siblings from leaf to root uint8[] path_indices; //
left/right indicators } // Verifier state (minimal) struct LightClient { mapping(uint64 =>
bytes32) header_hashes; // height → H(header) bytes32 latest_finalized; // tip of finalized
chain uint64 checkpoint_height; // last validated checkpoint // NO transaction history // NO
full state } // Transaction (abstract) struct Transaction { address from; address to; uint256
amount; uint256 nonce; bytes data; uint256 gas_limit; bytes signature; } // Verification result
enum VerificationResult { ACCEPT, REJECT_INVALID_HEADER_SIG, REJECT_INVALID_STATE_PROOF,
REJECT_PRECONDITION_FAIL, REJECT_POSTCONDITION_FAIL, REJECT_INCONSISTENT_STATE }
```

Message Formats

```
// Request state proof message GetStateProof { bytes32 state_root; // which state to prove
against bytes32 account_key; // which account } // Response message StateProofResponse {
StateProof proof; bytes32 state_root; // echo for verification } // Header sync message
HeaderChain { Header[] headers; // consecutive headers uint64 start_height; } // Verification
query message VerifyInclusion { Transaction tx; uint64 block_height; StateProof[] pre_proofs; //
state before StateProof[] post_proofs; // state after }
```

Rejection Conditions

1. Header signature invalid under quorum key 2. State proof path does not verify against claimed root 3. Account balance insufficient for transaction (precondition) 4. Post-state balance change inconsistent with transaction effect 5. State root mismatch between consecutive headers 6. Resource bound β_i exceeded (aggregate, not per-tx) 7. Nonce discontinuity (if enforced)

2. MINIMAL VERIFIER ALGORITHM

```
// Minimal light verifier for single account class MinimalBoundedVerifier { Header
latest_header; bytes32 tracked_account; // Verify transaction inclusion function
verify_inclusion( Transaction tx, Header H_prev, Header H_curr, StateProof proof_pre, StateProof
proof_post ) -> VerificationResult { // 1. Validate headers if
(!verify_quorum_sig(H_curr.quorum_sig, H_curr)) { return REJECT_INVALID_HEADER_SIG; } if
(hash(H_prev) != H_curr.prev_header_hash) { return REJECT_INVALID_HEADER_SIG; } // 2. Verify
state proofs if (!merkle_verify(proof_pre, H_prev.state_root)) { return
REJECT_INVALID_STATE_PROOF; } if (!merkle_verify(proof_post, H_curr.state_root)) { return
REJECT_INVALID_STATE_PROOF; } // 3. Extract account states Account acct_pre =
decode_account(proof_pre.rlp_account); Account acct_post =
decode_account(proof_post.rlp_account); // 4. Check preconditions if (acct_pre.balance <
tx.amount) { return REJECT_PRECONDITION_FAIL; } if (acct_pre.nonce != tx.nonce) { return
REJECT_PRECONDITION_FAIL; } // 5. Check state consistency (WEAK CHECK - inequality) int256
balance_delta = acct_pre.balance - acct_post.balance; if (balance_delta < tx.amount) { //
Account decreased by less than tx amount - impossible return REJECT_POSTCONDITION_FAIL; } // 6.
Nonce advancement check if (acct_post.nonce < acct_pre.nonce + 1) { return
REJECT_INCONSISTENT_STATE; } // ACCEPT: Transaction effect consistent with state transition //
NOTE: Does NOT prove tx was executed, only that effect is consistent return ACCEPT; } //
Assumption: Header finality via quorum signatures // Assumption: Merkle-Patricia tree for state
commitment // Assumption: No concurrent transactions visible (weak check allows them) //
Assumption: Validator quorum is honest (cannot verify otherwise) } // CRITICAL ASSUMPTIONS FOR
CORRECTNESS: // A1: Quorum threshold f < n/3 (Byzantine fault tolerance) // A2: Signature scheme
is unforgeable (BLS/threshold signatures) // A3: Hash function collision-resistant (keccak256 or
stronger) // A4: Canonical transaction ordering enforced by validators // A5: State proof
provider is available (data availability) // A6: No state proof equivocation (same root → same
proof)
```

Explicit Assumptions Required: • Byzantine quorum: > 2/3 validators honest • Data availability: Can retrieve state proofs on demand • No equivocation: Single canonical state per root • Signature unforgeability: ECDSA/BLS secure • Collision resistance: Hash functions secure • Canonical ordering: Validators agree on transaction order

3. ADVERSARIAL ATTACK ATTEMPT

Attack Vector 1: Unobserved State Manipulation

Setup: • Verifier tracks account A (Alice) • Adversary controls validator majority • Blockchain has accounts A, B, C (C untracked) **Attack:** 1. Honest state: A=100, B=50, C=200 (total supply=350) 2. Adversary creates invalid transaction: Mint(C, +50) (total supply=400) 3. Adversary includes valid transaction: Transfer(A→B, 10) 4. Produces header H_i with state root R_i covering all changes 5. Provides valid state proofs for A and B to verifier **Verifier sees:** • A: 100 → 90 (valid, matches transfer) • B: 50 → 60 (valid, matches transfer) • Header signed by quorum ✓ **Result:** VERIFIER ACCEPTS **Reality:** Adversary violated conservation law (minted tokens) **Why attack succeeds:** Partial observability (Lemma 2). Verifier cannot detect inflation affecting untracked accounts. This is NOT a bug—it's fundamental to light client security model. **Mitigation:** Trust quorum OR track all accounts (defeats purpose) OR fraud proofs

Attack Vector 2: Transaction Substitution

Setup: • Alice submits: Transfer(Alice→Bob, 100) with nonce=5 • Adversary intercepts **Attack:** 1. Adversary creates different transaction: Transfer(Alice→Charlie, 100) with nonce=5 2. Same precondition: Alice.balance ≥ 100, nonce=5 3. Same postcondition: Alice.balance -= 100, nonce=6 4. Execute adversary's transaction, not Alice's 5. Provide state proofs showing consistent state change **Verifier sees:** • Pre: Alice balance=150, nonce=5 • Post: Alice balance=50, nonce=6 • Change consistent with claimed transaction ✓ **Result:** VERIFIER ACCEPTS (weak inclusion) **Reality:** Different transaction executed **Why attack succeeds:** Weak inclusion (Definition 1) only verifies effect equivalence, not transaction identity. Verifier cannot distinguish transactions with identical pre/post conditions without transaction hash commitment. **Mitigation:** Require transaction-specific identifier in state OR accept weak inclusion for non-identity-sensitive applications

Attack Vector 3: Selective Header Withholding

Setup: • Verifier at height h, waiting for h+1 • Two forks exist (network partition) **Attack:** 1. Fork A (honest): Contains Alice's transaction, state: Alice=50 2. Fork B (censored): Excludes Alice's transaction, state: Alice=100 3. Adversary shows verifier only Fork B headers 4. Fork B eventually becomes canonical (by luck or stake weight) **Verifier sees:** • Headers valid, signatures valid • No state change for Alice (transaction absent) **Result:** VERIFIER ACCEPTS Fork B (transaction censored) **Reality:** Transaction never included **Why attack succeeds:** Bounded inclusion verifies correctness, not completeness. Cannot detect censorship without comparing against alternative forks or requiring transaction inclusion proofs. **Mitigation:** Monitor multiple forks OR fraud proofs showing withheld valid transactions OR social consensus fallback

Attack Vector 4: Gas Limit Evasion

Setup: • Header commits β_i = 1,000,000 gas • Transaction claims gas_cost = 21,000 **Attack:** 1. Transaction actually contains contract call with 500,000 gas consumption 2. Adversary validates header with quorum signature 3. Light client cannot execute contract to verify actual cost **Verifier sees:** • Transaction claims 21,000 gas • Header β_i = 1,000,000 (sufficient) • State transition consistent ✓ **Result:** VERIFIER ACCEPTS **Reality:** Cannot verify per-transaction cost without execution **Why attack succeeds:** Per-transaction gas cost unverifiable by light clients (acknowledged in Section 3.3). Verifier trusts header's aggregate bound, not individual transaction costs. **Mitigation:** Trust quorum to enforce gas limits OR use worst-case gas estimates OR require gas cost commitments in state (expensive)

VERDICT: All four attacks succeed under stated assumptions. This is EXPECTED—the paper explicitly acknowledges partial observability (Theorem 1 Corollary, Lemma 2), weak inclusion (Definition 1), and gas unverifiability (Section 3.3). These are not flaws in the model but fundamental trade-offs of light client verification. The model holds IF AND ONLY IF: • Verifier accepts partial security (tracks subset of state) • Applications tolerate weak inclusion (effect equivalence suffices) • Trust in validator quorum for global invariants • Censorship resistance handled separately Under these conditions, bounded inclusion provides correct verification for tracked accounts.

4. COMPARATIVE FALSIFICATION TABLE

Model	Verified	Assumed	Silent Failures	Global/Local
Bitcoin SPV	Header PoW Tx in block	Longest chain honest Miners validate	Invalid tx included Double-spend in block	Local only (header chain global)
Ethereum Light	Header consensus State proof Tx Merkle proof	Majority honest DA available	Censorship Invalid state (other accounts) Gas manipulation	Local verification Global state assumed
Optimistic Rollup	State root posted Fraud proof period	At least 1 honest challenger DA available	Fraud if no challenger Sequencer liveness L1 censorship	Global via fraud proofs Local after finality
ZK Rollup	State root Validity proof ZK proof correct	Trusted setup (if SNARK) Prover incentivized	Censorship Sequencer liveness Prover bugs	Global verification Local instant
Bounded Inclusion	Header consensus State consistency Resource bounds	Majority honest DA for state proofs Canonical ordering	Untracked account manipulation Tx substitution Censorship Per-tx gas cost	Local for tracked accounts Global trust in quorum

Key Observations: 1. **Bounded Inclusion vs Ethereum Light Client:** • Ethereum: Verifies tx $\in$ block via Merkle proof (transaction identity) • Bounded: Verifies effect consistent with state (transaction effect) • Trade-off: Bounded eliminates per-tx proofs but sacrifices identity guarantee 2. **vs Optimistic Rollups:** • Optimistic: Global security via fraud proofs (anyone can challenge) • Bounded: Local security via state proofs (only tracked accounts) • Trade-off: Bounded faster (no challenge period) but weaker global guarantees 3. **vs ZK Rollups:** • ZK: Cryptographic proof of global validity (expensive to generate) • Bounded: Cryptographic proof of local consistency (cheap to verify) • Trade-off: Bounded scales better for light clients but requires quorum trust **Unique Position:** Bounded inclusion is the ONLY model that achieves O(k) verification complexity for k accounts independent of transaction volume, at the cost of weak inclusion and partial observability. All other models verify stronger properties (transaction identity, global validity) but with higher complexity or latency.

5. IMPOSSIBILITY BOUNDARY

What This Model CANNOT Guarantee Under Any Circumstances: 1. CORRECTNESS (Partial Only) • Cannot verify: Global state consistency for untracked accounts • Cannot verify: Conservation laws (total supply, invariants) • Cannot verify: Per-transaction gas costs without execution • Cannot verify: Transaction identity (only effect equivalence) • Cannot verify: Absence of state manipulation in unobserved regions **2. FAIRNESS (Out of Scope)** • Cannot verify: Transaction ordering is fair • Cannot verify: No front-running / MEV extraction • Cannot verify: Absence of adversarial scheduling • Cannot guarantee: Optimal or canonical ordering **3. CENSORSHIP RESISTANCE (Out of Scope)** • Cannot detect: Valid transactions excluded from blocks • Cannot detect: Selective header withholding • Cannot detect: Network partitions hiding alternative forks • Cannot enforce: Transaction inclusion **4. GLOBAL AUDITABILITY (Requires Full Node)** • Cannot verify: All transactions in block are valid • Cannot verify: Block satisfies all protocol rules • Cannot verify: No double-spending outside tracked accounts • Cannot reconstruct: Complete execution trace from headers **Fundamental Limitations: Information-Theoretic:** Light clients with partial state access CANNOT verify global properties. This is not a flaw—it's a consequence of the CAP theorem analog for verification: you cannot have scalability (partial verification), completeness (global properties), and independence (no trust assumptions) simultaneously. **Computational:** Verifying execution without executing is impossible. Light clients must either trust validators (bounded inclusion), wait for fraud proofs (optimistic), or verify cryptographic proofs (ZK). Bounded inclusion chooses trust. **Cryptographic:** Collision-resistant commitments reveal nothing about preimage beyond the commitment. State roots commit to global state but light clients can only verify local fragments. **Distributed Systems:** In asynchronous networks with Byzantine actors, consensus requires quorum assumptions. Light clients inherently trust the quorum. Bounded inclusion makes this explicit rather than hiding it behind Merkle proofs. **The Core Trade-Off:** Bounded inclusion sacrifices: • Transaction identity verification • Global state validity • Censorship detection • Ordering fairness In exchange for: • O(k) complexity for k accounts • No per-transaction proofs • Constant verification time • Browser-native feasibility This trade-off is ACCEPTABLE if: • Application needs balance/state verification, not transaction identity • Validator quorum is sufficiently decentralized • Censorship resistance handled via social/economic mechanisms • Global auditability provided by full nodes (for fraud detection) It is UNACCEPTABLE if: • Application requires proof of specific transaction execution (timestamping, notarization) • Cannot trust any validator set (adversarial environment) • Need cryptographic guarantee of global validity • Censorship is primary threat

6. CORE INVARIANT

THE SINGLE CORE INVARIANT: "Deterministic state transitions with collision-resistant commitments enable verification of local state consistency without enumerating the transactions that produced that state." **ENTIRE PAPER AS CONSEQUENCE:** 1. **From determinism:** If $\delta(S_i, T_i)$ is unique for any T_i, then verifying S_{i+1} verifies that SOME valid T_i was executed, without knowing what T_i contained. 2. **From collision-resistance:** State root $R_i = h(S_i)$ uniquely identifies S_i, so verifying consistency with R_i verifies consistency with the actual state. 3. **From resource bounds:** β_i limits worst-case verification cost, making light client verification tractable even as transaction volume grows unbounded. 4. **Consequence 1 (Weak Inclusion):** A transaction t is "included" if its effects appear in the state, regardless of whether t itself or some effect-equivalent t' was executed. 5. **Consequence 2 (Partial Observability):** Verifying k accounts requires $O(k \cdot \log|S|)$ proofs, independent of $|T|$, because state access is orthogonal to transaction enumeration. 6. **Consequence 3 (Asymptotic Advantage):** As $|T| \to \infty$ while k remains fixed, bounded inclusion cost stays $O(k \cdot \log|S|)$ while Merkle inclusion grows $O(k \cdot \log|T|)$. 7. **Consequence 4 (Trust Model):** Cannot verify global properties without global state, so must trust quorum for unobserved accounts. 8. **Consequence 5 (Merkle Redundancy):** Transaction list Merkle proofs add no security beyond state consistency checks, because invalid transactions cannot produce valid state under deterministic execution. 9. **Consequence 6 (Fraud Proofs Still Needed):** Existence proofs don't imply reconstructability, so fraud proofs remain necessary for full auditability. 10. **Consequence 7 (Application Scope):** Works for state-querying applications (balances, contract state) but not for identity-dependent applications (timestamping). **IMPLICATION:** The entire model collapses if ANY of these fails: • Determinism violated $\rightarrow$ Cannot infer validity from state • Collision-resistance broken $\rightarrow$ State roots don't uniquely identify states • Resource bounds exceeded $\rightarrow$ Verification becomes unbounded • Quorum compromised $\rightarrow$ No guarantee unobserved state is valid The paper is essentially an elaborate exploration of this single invariant's consequences for blockchain verification.

7. TOP 5 REVIEWER OBJECTIONS

OBJECTION 1: "This is just SPV with extra steps" Severity: CRITICAL - Questions novelty **Argument:** Bitcoin SPV already does header-only verification. Ethereum light clients already use state proofs. What's new here? You've just removed Merkle proofs from Ethereum light clients and called it a contribution. **Evidence to resolve:** • Formal complexity analysis showing O(k·log|S|) vs O(k·log|T|) with concrete blockchain data (e.g., Ethereum mainnet with 1000 tx/block vs 10^8 accounts) • Benchmark showing actual bandwidth reduction for realistic light client workloads • Proof that transaction-list Merkle proofs provide zero additional security given state commitments (information-theoretic argument) **Counter-argument:** The paper does provide formal analysis (Proposition 1, Theorem 1) but lacks empirical validation. Need real-world measurements.

OBJECTION 2: "Weak inclusion breaks too many use cases" Severity: HIGH - Limits practical applicability **Argument:** If I can't verify transaction identity, I can't use this for: • Timestamp proofs (need specific transaction hash) • Notarization (need proof data was included) • Audit trails (need specific transaction sequence) • Cross-chain bridges (need exact transaction commitment) These are not niche applications—they're fundamental blockchain use cases. Your model excludes them by design. **Evidence to resolve:** • Survey of blockchain applications categorized by identity-sensitivity • Percentage of real-world transactions that require strong vs weak inclusion • Alternative designs for identity-critical applications in bounded inclusion framework • Formal definition of "identity-sensitive" vs "effect-sensitive" applications **Counter-argument:** Paper acknowledges this (Section 5.1, 6.1) but doesn't quantify impact. Need empirical analysis of application requirements.

OBJECTION 3: "Partial observability makes this unsafe" Severity: HIGH - Security concern **Argument:** You admit verifiers can't detect minting, burning, or invariant violations in untracked accounts (Attack Vector 1). This means: • Total supply can be inflated undetectably • Protocol rules can be violated silently • Adversary can steal from unobserved accounts with impunity This is not acceptable for a financial system. Traditional light clients at least verify Merkle roots cover ALL transactions—you've removed even that minimal global guarantee. **Evidence to resolve:** • Formal comparison showing traditional light clients have equivalent partial observability • Proof that Merkle roots provide no additional global security without full validation • Analysis of fraud proof effectiveness in detecting unobserved violations • Game-theoretic analysis: cost of attacking unobserved accounts vs detection probability **Counter-argument:** Paper claims equivalence to existing light clients (Theorem 1, Lemma 2) but doesn't prove traditional Merkle-based light clients have same limitations. This needs explicit proof or they're not equivalent.

OBJECTION 4: "Zenon is the only instantiation—is this even general?" Severity: MODERATE - Questions generality **Argument:** Your only concrete example is Zenon (Section 9). Zenon's account-chain architecture is unusual—most blockchains use global state machines. Can this even work on: • Ethereum (global state, no account chains) • Cosmos (IBC, cross-chain state) • Solana (parallel execution, no deterministic ordering) If bounded inclusion only works for account-chain architectures, it's not a general verification paradigm—it's a Zenon-specific optimization. **Evidence to resolve:** • Concrete instantiation for Ethereum (global Merkle-Patricia state) • Proof of compatibility with non-account-chain architectures • Performance comparison: Zenon account-chains vs Ethereum global state • Identify structural requirements: which blockchain designs are compatible? **Counter-argument:** Paper presents abstract model (Sections 3-7) independent of Zenon, but lacks proof that other architectures satisfy assumptions. Need more examples.

OBJECTION 5: "Asymptotic claims are misleading" Severity: MODERATE - Technical accuracy **Argument:** You claim O(k) complexity but actual cost is O(k·log|S|) for state proofs (Proposition 1 corrected version). The paper also claims O(1) per transaction in multiple places (Section 8) when it's actually O(log|S|). This is misleading. Moreover, your comparison to Merkle inclusion O(k·log|T|) only shows advantage when |T| >> |S|. For typical blockchains where transactions affect many accounts, |T| ≈ |S| per block, so the advantage is logarithmic at best—not the dramatic improvement suggested. **Evidence to resolve:** • Correct all O(1) claims to O(log|S|) with footnotes • Empirical analysis: real blockchain data showing |T| vs |S| relationship • Break-even analysis: at what |T|/|S| ratio does bounded inclusion outperform Merkle? • Concrete numbers: bandwidth for 1000 tx block with 100 affected accounts **Counter-argument:** Paper has corrected most claims (Section 6.4, 8.2) but some O(1) references remain. More importantly, lacks empirical validation of asymptotic advantage.

FINAL HOSTILE VERDICT

ACCEPT WITH MAJOR REVISIONS Core contribution is valid: Bounded inclusion correctly formalizes verification without transaction enumeration under stated assumptions. The model is sound. **But requires:** 1. Empirical validation (benchmarks, real blockchain data) 2. Additional instantiations beyond Zenon 3. Explicit comparison showing traditional light clients have equivalent limitations 4. Quantification of application scope (how many use cases tolerate weak inclusion?) 5. Correction of all remaining O(1) claims **Without these fixes:** Paper makes claims not supported by evidence. **With fixes:** Solid contribution to light client literature. **Confidence:** 95% - Model is correct, presentation needs work.