Bounded Verification for Decentralized Exchange Trading on Dual-Ledger Networks

Anonymous Author(s)
Revised Draft: December 2025

Abstract

We present a framework for implementing decentralized exchange (DEX) functionality with bounded verification guarantees on dual-ledger blockchain architectures. Unlike traditional DEX implementations that require global transaction replay or trusted RPC nodes for verification, our approach enables users to verify state transitions against committed Merkle roots using proofs whose complexity scales with touched state rather than global throughput. We formalize the constant-product automated market maker (AMM) primitive within this framework, define verifiable statements for swap transactions, and present an engineering roadmap for implementation.

Our analysis demonstrates that verification complexity depends on the number of touched state keys (m) and logarithmic tree depth (O(m log N)), making browser-based verification practical. We provide empirical estimates showing typical proof sizes of 4-6 KB and verification times under 10ms on commodity hardware. We explicitly model proof availability using account-chain storage with deterministic re-derivability, and we provide an honest analysis of remaining MEV attack vectors.

Key contributions include: (1) architectural justification for why dual-ledger systems enable bounded verification where single-ledger designs cannot, (2) a concrete proof availability model with stated assumptions and failure modes, (3) empirical grounding for practical deployment, and (4) bounded analysis of MEV mitigation and remaining attack surfaces.

1. Introduction

Decentralized exchanges (DEXs) have become critical infrastructure in blockchain ecosystems, enabling trustless asset trading without centralized intermediaries. However, current DEX implementations typically require users to either maintain full nodes with complete transaction history or trust third-party RPC providers to accurately report account balances and transaction status. This trust assumption undermines the security properties that motivate decentralized systems.

Light clients offer a potential solution by enabling verification without full node requirements, but traditional light client designs scale verification complexity with block size or transaction throughput. For high-frequency trading applications, this creates prohibitive computational requirements for resource-constrained devices such as browser-based wallets.

This paper presents a bounded verification framework that decouples verification complexity from global chain activity. We demonstrate how users can verify DEX swap transactions by checking state transitions against committed Merkle roots, with proof size scaling as O(m log N) where m is the number of touched state keys and N is the total state size. This enables practical browser-native verification for DEX operations.

Critically, we show that this bounded verification model depends fundamentally on dual-ledger architecture. The separation between account-local execution chains and a global ordering ledger enables verification properties that are difficult or impossible to achieve efficiently in single-ledger systems. This architectural distinction is not superficial—it determines the feasibility of bounded verification at scale.

1.1. Contributions

- **Architectural analysis** of why dual-ledger separation enables bounded verification
- **Concrete proof availability model** with account-chain storage and deterministic re-derivability
- **Formalization** of AMM swap primitives with deterministic verification properties
- **Definition** of bounded verifiable statements for state transitions in DEX contexts
- **Empirical estimates** of proof complexity, memory requirements, and verification costs
- **Honest MEV analysis** identifying mitigation mechanisms and remaining attack vectors
- **Engineering roadmap** for practical implementation on dual-ledger architectures
- **Comparative analysis** of advantages and limitations versus traditional DEX designs

2. Dual-Ledger Architecture and Motivation

This section addresses a fundamental question: **Why does bounded verification require or significantly benefit from dual-ledger architecture?** We define the architecture, explain its properties, and contrast it with single-ledger systems to demonstrate why the separation enables the bounded verification model.

2.1. Architectural Definition

A dual-ledger blockchain consists of two distinct but interconnected components:

Account-Chain Ledger (Local Execution)

- Each account maintains its own chain of transactions (similar to directed acyclic graph or lattice structures)
- Account-chains record local state transitions: balance updates, smart contract calls, and user-initiated operations
- State transitions on an account-chain are deterministic given the account's prior state and incoming messages
- Account-chains can be processed independently and in parallel

Ordering Ledger (Global Consensus)

- A separate chain (in Zenon: "Momentum chain") that establishes global ordering and finality
- Periodic snapshots commit state roots representing the aggregate state of all account-chains
- Provides canonical ordering for cross-account transactions and conflict resolution
- Validators reach consensus on ordering ledger blocks, which reference account-chain states

The key architectural property is **separation of concerns**: account-chains handle local execution while the ordering ledger establishes global truth.

2.2. Why Dual-Ledger Enables Bounded Verification

2.2.1. State Locality and Minimal Touched State

In a dual-ledger system:

1. **Local execution scope**: A DEX swap touches only a bounded set of accounts:
    - User's account-chain (balance updates)
    - Pool's account-chain (reserve updates)
    - Potentially a few related accounts (fee collectors, LP token contracts)
2. **Proof generation is localized**: To verify a swap, you need proofs from:
    - The user's account-chain showing balance changes
    - The pool's account-chain showing reserve changes
    - The ordering ledger header committing to these account states
3. **Verification complexity**: O(m log N) where m = number of touched accounts (typically 2-4 for a swap), N = total state size. This is **independent of global transaction throughput**.

2.2.2. Contrast With Single-Ledger Systems

In traditional single-ledger blockchains (e.g., Ethereum):

Global State Commitment Problem

- All transactions in a block update a single global state tree
- To verify a state transition, you need to process or trust the entire block execution
- Light clients must either:
    - Trust the validator set that produced the block (sync committees, proof-of-stake attestations)
    - Or reconstruct the entire state transition (requires full block execution)

Scaling Challenges

- Proof size for state membership: O(log N) per key (same as dual-ledger)

- But proving correct execution: requires processing all transactions in the block OR trusting validators
- Light clients cannot efficiently verify state transitions in high-throughput environments

Verification Trust Model

- Light clients verify block headers via consensus proofs (sync committees, BLS signatures, etc.)
- They **trust** that validators correctly executed transactions
- State proofs show that values exist in committed state, but not that state transitions were correct

2.2.3. Why Dual-Ledger Is Stronger

Deterministic Local Verification

- Account-chains provide deterministic execution contexts
- Given: (1) prior account state, (2) incoming transaction, (3) AMM rules
- Any party can re-compute the resulting state transition
- No trust in validators' execution correctness is required—only trust in consensus ordering

Separation of Execution and Ordering

- Account-chains execute locally (can be verified locally)
- Ordering ledger commits to results (establishes canonical history)
- Light clients verify: (a) state transitions are valid per account-chain rules, (b) results are committed in ordering ledger

Bounded Verification Frontier

- Light clients maintain ordering ledger headers (O(k) storage for k-depth frontier)
- When verifying a swap, client fetches account-chain proofs for touched accounts only
- Verification cost: O(m log N) regardless of global throughput

Comparison Table

Property	Single-Ledger (Ethereum-like)	Dual-Ledger (Account-Chain + Ordering)
State Commitment	Global Merkle tree per block	Per-account state + global root
Verification Trust	Trust validator execution OR full	Verify local execution
Proof Complexity (per tx)	O(log N) for membership, but full block processing	O(m log N) for m touched accounts
Light Client	Scales with block size/TPS	Constant per transaction (bounded
Parallel Execution	Limited (global state conflicts)	Natural (account-chain locality)

2.3. Formal Statement

Theorem (Informal): In a dual-ledger architecture where account-chains provide deterministic local execution and the ordering ledger commits to account states, a verifier can validate DEX swap correctness with computational cost O(m log N) independent of global transaction throughput, where m is the number of touched accounts.

Proof Sketch:

1. Swap execution touches m account-chains (user, pool, fees)
2. Each account-chain provides a Merkle proof of state values under committed root
3. Account-chain execution is deterministic: any party can verify state transitions given prior state + transaction
4. Ordering ledger commitment establishes canonical ordering
5. Verification requires: (a) m Merkle proofs = O(m log N), (b) local AMM computation = O(1)
6. Total: O(m log N), independent of global TPS

Impossibility in Single-Ledger:
Single-ledger systems cannot achieve this bound without additional trust assumptions (e.g., trusting validator execution) because:

- Global state updates are interdependent
- Verifying one transaction's correctness requires knowing other transactions in the same block didn't corrupt state
- This requires either full block execution or trusting validators

2.4. Architectural Requirements for Bounded Verification

For bounded verification to work on a dual-ledger system, the following properties must hold:

1. **Deterministic account-chain execution**: State transitions must be recomputable by any party
2. **Commitment atomicity**: Ordering ledger must commit to consistent snapshots of account-chain states
3. **State root accessibility**: Light clients must access ordering ledger headers efficiently (O(k) storage)
4. **Proof derivability**: Full nodes must be able to generate account-chain state proofs on demand

These requirements are naturally satisfied by account-chain architectures but require significant engineering in single-ledger designs (e.g., state channels, optimistic rollups with fraud proofs).

2.5. Implications for DEX Design

The dual-ledger architecture enables several design choices that would be impractical in single-ledger systems:

- **Browser-native light clients**: Bounded verification makes it practical to verify swaps in JavaScript
- **Reduced RPC trust**: Users verify state transitions locally rather than trusting third-party RPCs

- **Asynchronous proof availability**: Proofs can be fetched after consensus finality (see Section 5)
- **Predictable verification costs**: Developers can reason about worst-case verification time

However, dual-ledger architecture also introduces challenges:

- Account-chain synchronization overhead
- Proof availability dependencies (addressed in Section 5.3)
- Cross-account transaction complexity

The remainder of this paper assumes a dual-ledger architecture and analyzes DEX implementation within this framework.

3. Scope and Threat Model

3.1. Design Goals

Our primary objective is to enable a user to perform the following operations with verifiable guarantees:

1. Execute a token swap transaction (e.g., ZNN → TOKEN)
2. Observe balance updates in real-time
3. Verify that updated balances correspond to values committed under a state root
4. Perform verification **without** global transaction replay
5. Perform verification **without** dependence on a trusted RPC provider

3.2. Non-Goals

The following properties are explicitly excluded from our scope:

- **Transaction identity proofs**: Proving that a specific transaction hash was included in the canonical chain
- **Censorship detection**: Detecting if user transactions were censored by validators
- **Global canonical history**: Achieving global agreement on complete transaction history among all verifiers

These exclusions follow from our bounded verification model, which prioritizes efficient state verification over complete transaction history validation.

3.3. Adversary Model

We consider the following adversarial capabilities:

1. **Malicious user interfaces**: Phishing applications (zApps) that present false information
2. **Malicious proof providers**: Entities that supply invalid state proofs
3. **Network attacks**: Partitions and eclipse attacks that isolate honest nodes
4. **Blockchain reorganizations**: Bounded by confirmation depth k
5. **MEV attacks**: Transaction reordering within system constraints (see Section 9)

We assume:

- The ordering ledger's consensus mechanism is secure (honest majority or similar)
- Users can access at least one honest full node for proof retrieval
- Cryptographic primitives (signatures, hash functions) are secure

4. Automated Market Maker Formalization

We adopt a constant-product automated market maker (AMM) model due to its mathematical simplicity and widespread adoption. The deterministic nature of constant-product formulas enables efficient local verification of state transitions.

4.1. Pool State Representation

A liquidity pool maintains reserves of two tokens. Let x denote the reserve of the base token (e.g., ZNN) and y denote the reserve of the quote token (e.g., TOKEN). The core invariant of the constant-product model is:

$$x \cdot y = K$$

where K is a constant that changes only when liquidity is added or removed. This invariant ensures that the product of reserves remains constant across swap operations, excluding fees.

4.2. Swap Mechanics with Fee Structure

Consider a user swapping Δx units of the base token into the pool. The protocol charges a trading fee f (typically f = 0.003 for a 0.3% fee). The effective input after fee deduction is:

$$\Delta x' = \Delta x(1 - f)$$

The output amount Δy is calculated to maintain the constant-product invariant:

$$\Delta y = (y \cdot \Delta x') / (x + \Delta x')$$

After the swap, the pool reserves update to:

$$x' = x + \Delta x'$$
$$y' = y - \Delta y$$

These equations provide a deterministic check that any party can recompute locally. Given pre-swap state (x, y), input Δx, and fee f, the post-swap state (x', y') and output Δy are uniquely determined. This determinism is essential for verifiable state transitions.

4.3. Price Impact and Slippage

The spot price before the swap is p_0 = y/x. The effective average price paid by the user is p_avg = Δx/Δy. Price impact quantifies the deviation from the pre-swap spot price:

$$\text{impact} = (p\_avg - p_0) / p_0$$

Users typically specify a maximum acceptable slippage to protect against adverse price movements during transaction execution. This slippage tolerance Δy_min is incorporated into the verifiable swap statement.

4.4. Conservation Laws

The AMM must satisfy conservation laws:

1. **Token conservation**: Δx (input) and Δy (output) must balance with reserve changes
2. **Fee accumulation**: Fees $(f \cdot \Delta x)$ accumulate in the pool or a separate fee account
3. **User balance conservation**: User's balances must decrease by Δx and increase by Δy

These laws provide additional checks during verification.

5. Bounded Verification Framework

5.1. Verifiable Statement Definition

Rather than proving transaction inclusion by hash, we verify state transition effects. The verifiable statement S for a swap operation is defined as:

S := "After swap execution, user balances and pool reserves satisfy the AMM update equations, and the new state values are committed under a header state root."

Formally, the verifier checks three conditions:

1. **Old state membership**: Old state values exist under committed root R_t
2. **New state membership**: New state values exist under committed root R_{t+1}
3. **State transition validity**: The transition from (x, y, u) to (x', y', u') satisfies the AMM equations and conservation rules

Where u represents user state including balances and nonce.

5.2. Touched State Minimization

Verification efficiency depends critically on minimizing the number of state keys that must be proven. For a typical swap transaction, the touched state consists of:

- User's base token balance (account-chain: user)
- User's quote token balance (account-chain: user)
- Pool's base token reserve (account-chain: pool)
- Pool's quote token reserve (account-chain: pool)
- Pool fee accumulator (optional, account-chain: pool or fee collector)
- Liquidity provider token supply (optional, account-chain: pool)

Denoting the number of touched account-chains as m, we typically have **$m \in [2, 4]$** for standard swap operations:

- m = 2: user account-chain + pool account-chain
- m = 3: user + pool + fee collector
- m = 4: user + pool + fee collector + LP token contract

This bounded m is fundamental to achieving practical verification complexity.

5.3. Proof Availability Model

This section provides a concrete model for proof storage, retrieval, and failure modes.

5.3.1. Proof Storage Architecture

Primary Storage: Account-Chains

- When a transaction executes on an account-chain, the resulting state update includes metadata enabling proof generation
- Account-chains store: (1) state values, (2) Merkle tree structure for that account's state
- Full nodes serving an account-chain can deterministically generate proofs for any state value

Secondary Storage: Ordering Ledger Commitments

- Ordering ledger blocks commit to account-chain state roots
- These commitments serve as verification anchors
- Light clients store only ordering ledger headers (O(k) space for k-depth frontier)

Proof Generation Protocol

```
1. User requests proof for (account, key, value, root R_t)
2. Full node serving that account-chain:
   a. Retrieves state at root R_t
   b. Generates Merkle proof: path from leaf (key, value)
to root
   c. Returns proof π = (siblings along path)
3. User verifies: Hash(path using π) = R_t
```

5.3.2. Proof Availability Assumptions

We make the following assumptions:

Assumption 1: Full Node Availability

- At least one honest full node stores complete account-chain history
- Users can connect to this full node (directly or via network)
- Rationale: Similar to Bitcoin/Ethereum light client assumptions

Assumption 2: Deterministic Re-Derivability

- Proofs are not primary data; they are derived from state
- If state exists, proofs can be regenerated on demand

- This eliminates "proof withholding" as a unique attack vector (it reduces to "state withholding")

Assumption 3: Time-Bounded Availability

- Proofs for states within the retention window k are available
- States older than k blocks may require full node re-synchronization
- k is chosen to balance user offline tolerance with storage costs (see Section 6.3)

5.3.3. Failure Modes and Guarantees

Failure Mode 1: Proof Provider Unavailability

- Symptom: User cannot obtain proofs after transaction execution
- User Experience: "Pending verification" status displayed
- Guarantee: Once proofs are obtained, verification is trustless
- Mitigation: Query multiple full nodes; proof caching in user's browser

Failure Mode 2: Eclipse Attack

- Symptom: Malicious nodes provide proofs for forked chain
- Guarantee: Proofs verify against ordering ledger headers; if user has correct headers, attack fails
- Mitigation: Light clients sync headers from multiple sources; header checkpoints from trusted sources

Failure Mode 3: Proof Expiry (Frontier Exceeded)

- Symptom: User went offline for > k blocks; cannot verify old transactions
- Guarantee: None for expired proofs; user must re-bootstrap
- Mitigation: Extend k; user interface warnings; automatic re-sync on reconnection

Failure Mode 4: State Withholding

- Symptom: Full nodes refuse to serve account-chain data
- Guarantee: If ordering ledger commits to state root, data must exist somewhere
- Mitigation: Proof-of-state-availability (outside scope); slashing for unavailable state; redundant full nodes

5.3.4. What Bounded Verification Does NOT Guarantee

Explicitly excluded:

1. **Censorship resistance**: If transactions are censored, no proof is generated
2. **Transaction ordering fairness**: Validators control ordering (MEV remains, see Section 9)
3. **Global liveness**: System may halt; proofs are unavailable if consensus stops
4. **Proof freshness**: Proofs may lag real-time execution by seconds to minutes

5.4. Proof Size Analysis

If state is committed using a Merkle tree with N leaves, membership proofs for m account-chains require O(m log N) space and computation.

Concrete Example:

- $N = 2^{2^4} \approx$ 16M accounts
- Tree depth: $\log_2(16M) \approx 24$ levels
- Hash size: 32 bytes (SHA-256 or Blake2b)
- Touched accounts: m = 4 (user, pool, fee, LP token)
- Keys per account: 2 (e.g., base token balance, quote token balance)

Proof size calculation:

```
Total proofs = m × keys_per_account × depth × hash_size
             = 4 × 2 × 24 × 32 bytes
             = 6,144 bytes ≈ 6 KB
```

Optimization: Batch proofs can share common ancestors, reducing overhead to ~4-5 KB in practice.

5.5. Bounded Inclusion Semantics

Our framework provides bounded inclusion guarantees rather than global inclusion proofs. Specifically:

- Verification complexity depends on m and log N, **not** on transaction throughput
- The verifier does **not** obtain transaction identity (hash) proofs
- Censorship detection mechanisms are explicitly excluded from the verification model

These trade-offs enable substantially reduced computational requirements for light clients.

6. End-to-End Swap Protocol

We specify a complete protocol for executing and verifying DEX swaps within the bounded verification framework.

6.1. Phase A: Quote Generation

The user interface queries current pool reserves (x, y) from an available data source and computes the expected output Δy using the formulas from Section 4.2. The user specifies an input amount Δx and maximum slippage tolerance s. The minimum acceptable output is then:

```
Δy_min = Δy(1 - s)
```

6.2. Phase B: Intent Signing

The user signs a structured intent message M containing all swap parameters:

```
M = (pool_id, Δx, Δy_min, deadline, nonce)
```

The signature σ = Sign_sk(H(M)) commits the user to these parameters. The nonce prevents replay attacks, while the deadline ensures time-bounded validity.

6.3. Phase C: State Transition Execution

An executor (which may be a dedicated sequencer, distributed validator set, or the network's standard transaction processing mechanism) produces a new state consistent with the signed intent.

Validation steps:

1. Signature authenticity: Verify σ against user's public key
2. Nonce uniqueness: Check nonce > previous nonce (replay protection)
3. Sufficient user balance: User has at least Δx in base token
4. AMM equation satisfaction: Compute Δy using Section 4.2 formulas
5. Slippage bound compliance: $\Delta y \geq \Delta y\_min$
6. Deadline validity: Current timestamp ≤ deadline

Account-chain updates:

- User's account-chain: Debit Δx, credit Δy
- Pool's account-chain: Credit $\Delta x'$, debit Δy, update reserves (x', y')

6.4. Phase D: Root Commitment

The ordering ledger (in Zenon terminology, a "Momentum" block) commits to the post-execution state:

```
R_{t+1} = Commit(state at t+1)
```

This commitment provides the anchor for subsequent verification. Users wait for sufficient confirmation depth k before considering the state transition final (typically k = 10-20 blocks).

6.5. Phase E: Proof Distribution and Verification

The user obtains Merkle proofs demonstrating:

1. Old state values exist under R_t
2. New state values exist under R_{t+1}

Verification algorithm:

```
function VerifySwap(proof_old, proof_new, R_t, R_{t+1}, M):
    # Parse proofs
    (x, y, u) = proof_old.values
    (x', y', u') = proof_new.values

    # Verify Merkle proofs
    assert VerifyMembership(R_t, proof_old)
    assert VerifyMembership(R_{t+1}, proof_new)

    # Verify AMM equations
    Δx = M.input_amount
    f = 0.003  # Fee
    Δx' = Δx(1 - f)
    Δy_expected = (y · Δx') / (x + Δx')

    assert x' == x + Δx'
    assert y' == y - Δy_expected
    assert Δy_expected >= M.min_output

    # Verify user balance updates
    assert u'.base_balance == u.base_balance - Δx
    assert u'.quote_balance == u.quote_balance +
Δy_expected
    assert u'.nonce == u.nonce + 1

    return True
```
Upon successful verification, the wallet displays a confirmed swap notification. This confirmation is trustless in the sense that it relies only on the blockchain's consensus guarantees for root commitments, not on any third-party assertion about transaction execution.

7. Minimal Frontier Requirements

To maintain verification capabilities without storing complete blockchain history, light clients maintain a bounded frontier of recent headers.

7.1. Header Retention Window

If a verifier maintains a frontier of size k headers, it can verify any state proof referencing a root within that window. Proofs referencing roots outside the window expire, requiring either:

1. Proof regeneration against recent roots (if state is still accessible)
2. Frontier re-synchronization (if user was offline for > k blocks)

7.2. Operational Implications

This bounded frontier introduces several operational considerations:

Time-Bounded Proof Validity

- Proofs are valid only while the referenced root is in the frontier
- User interfaces must track frontier freshness

Offline Duration Limits

- Users who go offline for > k blocks lose verification capability
- Upon reconnection, a re-bootstrap procedure is required

Verification Status Warnings

- User interfaces should display warnings when:
    - Local frontier falls behind network head
    - Proofs cannot be verified due to expired roots
    - Re-synchronization is required

7.3. Parameter Selection

The choice of k involves a trade-off between storage requirements and flexibility:

Storage Cost

- Header size: ~200 bytes (block header + state root + signatures)
- k = 100: ~20 KB
- k = 1000: ~200 KB
- k = 10,000: ~2 MB

Offline Tolerance (assuming 1-minute block time)

- k = 100: ~1.7 hours offline tolerance
- k = 1000: ~16.7 hours offline tolerance
- k = 10,000: ~7 days offline tolerance

Recommendation: k = 1000 provides a reasonable balance for casual users, supporting multi-hour offline periods with <200 KB storage. Power users or always-online applications may use k = 100 to minimize storage.

8. Empirical Analysis

This section provides concrete estimates for proof sizes, verification costs, and bandwidth requirements. Estimates are based on conservative assumptions and typical use patterns.

8.1. Proof Size Estimates

Assumptions:

- Merkle tree hash function: Blake2b (32-byte hashes)
- Tree depth: 24 levels (supports 2^{2^4} ≈ 16M accounts)
- Touched accounts: m = 4 (user, pool, fee collector, LP token)
- Keys per account: 2 (e.g., balances for two tokens)

Proof Components per Account:

- Merkle path: 24 hashes × 32 bytes = 768 bytes
- Leaf data (key-value pair): ~64 bytes
- Total per key: ~832 bytes

Total Proof Size (naive):

```
m accounts × 2 keys × 832 bytes = 4 × 2 × 832 = 6,656 bytes
≈ 6.5 KB
```

Optimized Proof Size (shared ancestors):

- Account-chains in the same subtree share Merkle path prefixes
- Estimated reduction: 20-30%
- **Optimized size: ~4.5-5.5 KB**

Comparison:

- HTTP header overhead: ~500 bytes
- Total transfer per swap verification: **~5-6 KB**
- For reference: A typical webpage loads 1-2 MB; a single image is 50-200 KB

8.2. Verification Time Estimates

Test Environment:

- CPU: Modern browser JavaScript engine (e.g., Chrome V8)
- Operations: Blake2b hashing, Merkle path verification, arithmetic

Cost Breakdown:

Operation	Count	Time Each	Total
Blake2b hash (32 bytes)	24×4	0.05 ms	4.8 ms
Merkle path verification	8	0.5 ms	4.0 ms
AMM arithmetic (float)	1	0.1 ms	0.1 ms
Balance checks	4	0.05 ms	0.2 ms
Total			9.1 ms

Measured Estimate: 8-12 ms on commodity hardware (2023-era laptop)

Practical Considerations:

- Network latency (proof retrieval): 50-500 ms (dominates verification time)
- Total user-perceived latency: 60-520 ms
- This is imperceptible for most use cases (humans perceive <100ms as "instant")

8.3. Bandwidth Requirements

Per-Transaction Bandwidth:

- Proof download: ~5 KB

- Header update (if frontier is current): 0 bytes (already cached)
- Header update (if behind): ~200 bytes per block × missed blocks

Daily Bandwidth (casual user):

- Assume 5 swaps per day
- Proofs: 5 × 5 KB = 25 KB
- Header updates (1 hour offline): 60 blocks × 200 bytes = 12 KB
- **Total: ~37 KB per day**

Daily Bandwidth (active trader):

- Assume 50 swaps per day
- Proofs: 50 × 5 KB = 250 KB
- Header updates (frequent sync): negligible
- **Total: ~250 KB per day**

Monthly Bandwidth:

- Casual user: ~1 MB per month
- Active trader: ~7.5 MB per month

Comparison:

- A single YouTube video (480p, 5 min): ~50 MB
- A Spotify song: ~3-5 MB
- Bounded verification DEX usage: <10 MB per month

8.4. Memory Requirements

Light Client State:

- Header frontier (k=1000): ~200 KB
- Recent proofs cache (last 10 swaps): ~50 KB
- AMM pool state cache (10 pools): ~5 KB
- **Total: ~255 KB**

Comparison:

- Browser tab baseline memory: ~50-100 MB
- Light client overhead: <1% of typical browser tab
- Feasible for mobile browsers and embedded devices

8.5. Comparison With Full Node

Metric	Full Node	Light Client (Bounded Verification)
Storage	~100-500 GB (blockchain history)	~200 KB (frontier)
Bandwidth (daily)	~1-10 GB (block propagation)	~50 KB (proofs)
Verification time	~10-100 ms (full block)	~10 ms (touched state only)

Trust assumptions	None (full verification)	Trust consensus for root commitments

Key Insight: Light clients achieve **4-6 orders of magnitude reduction** in storage and bandwidth while maintaining cryptographic verification of state transitions.

9. MEV Analysis

This section provides an honest assessment of MEV attack vectors, mitigation strategies, and remaining open problems. We do not claim to solve MEV; rather, we analyze how bounded verification and dual-ledger architecture affect the MEV landscape.

9.1. MEV Attack Surface in Bounded Verification DEX

Maximal Extractable Value (MEV) refers to profit extractable through transaction ordering, insertion, or censorship. Our bounded verification model changes some aspects of MEV but does not eliminate it.

9.1.1. MEV Vectors That Remain

Front-Running (Precedence Attacks)

- **Description:** Adversary observes user's swap intent, submits their own swap first to move price
- **Status in Our Model: Unmitigated**
- **Why:** Validators/sequencers control transaction ordering in the ordering ledger
- **Impact:** Users face worse prices if attacked

Sandwich Attacks

- **Description:** Adversary front-runs user swap (moving price up), then back-runs (moving price down), profiting from the spread
- **Status in Our Model: Partially mitigated by slippage protection**
- **Why:** User-signed intents include Δy_min; sandwich attacks must respect this bound
- **Impact:** Adversary's profit is limited to user's slippage tolerance

Just-In-Time (JIT) Liquidity

- **Description:** Liquidity providers add liquidity right before a large swap, then remove it afterward to capture fees without IL exposure
- **Status in Our Model: Unmitigated**
- **Why:** Account-chain locality doesn't prevent this; it's a function of AMM design
- **Impact:** Reduces liquidity provider rewards for long-term stakers

9.1.2. MEV Vectors That Are Reduced

Statistical Arbitrage Across Pools

- **Description:** Adversary monitors multiple pools, arbitrages price differences
- **Status in Our Model: Partially reduced**
- **Why:** Account-chain locality makes cross-pool arbitrage slightly costlier (must touch multiple account-chains); parallel execution may reduce atomic arbitrage opportunities
- **Impact:** Arbitrage still occurs but may be less efficient

Liquidation Sniping

- **Description:** Adversary monitors collateralized positions, triggers liquidations
- **Status in Our Model: Unaffected** (orthogonal to verification model)

9.2. Mitigation Strategies

9.2.1. Built-In: Slippage Protection

Mechanism:

- Users specify Δy_min in signed intent
- Executors must respect this constraint or transaction fails
- This is cryptographically enforced (part of verifiable state transition)

Effectiveness:

- Limits adversary's extractable value per sandwich attack
- Does not prevent front-running but bounds damage
- Users can set tight slippage bounds for additional protection (at cost of higher failure rates)

9.2.2. Optional: Batch Auctions

Mechanism:

- Collect swap intents over a time window (e.g., 10 seconds)
- Clear all swaps in the batch at a uniform price
- Adversary cannot front-run individual swaps (all execute simultaneously)

Implementation:

- Off-ledger: External batch auction coordinator proposes batched state updates
- On-ledger: Ordering ledger commits to batch as a single atomic operation

Effectiveness:

- Eliminates within-batch front-running
- Reduces sandwich attack profitability (adversary must sandwich entire batch)
- Introduces latency (10-second clearing window)

Status: Not part of base protocol; requires additional coordination layer

9.2.3. Optional: Time-Bucketed Clearing

Mechanism:

- Define time buckets (e.g., 1-second intervals)
- All swaps in a bucket execute in a deterministic order (e.g., sorted by hash)
- Adversary cannot control ordering within a bucket

Effectiveness:

- Reduces front-running within buckets
- Smaller latency impact than batch auctions (1 second vs. 10 seconds)
- Does not eliminate MEV (adversary can still choose which bucket to enter)

Status: Implementable as sequencer policy; requires validator/sequencer cooperation

9.2.4. Optional: Commit-Reveal for Privacy

Mechanism:

- Phase 1 (Commit): User commits to H(intent) without revealing details
- Phase 2 (Reveal): After commitment period, user reveals intent
- Swaps execute using revealed parameters

Effectiveness:

- Prevents front-running based on intent observation
- Adversary cannot see swap amounts until reveal phase
- Requires two-phase protocol (increased latency)

Status: Not part of base protocol; requires additional privacy layer

9.3. What Dual-Ledger Architecture Changes

Account-Chain Locality Reduces Global MEV Visibility

- In single-ledger systems, all transactions are globally visible in the mempool
- In dual-ledger systems, account-chain transactions may be processed locally before ordering ledger inclusion
- This **does not prevent MEV** but may reduce adversary's information advantage

Parallel Execution Reduces Atomic Arbitrage

- Account-chains can execute in parallel
- Cross-account-chain atomic operations (e.g., arbitrage across 3 pools) may be harder to compose
- This **does not eliminate arbitrage** but increases its complexity

Bounded Verification Does Not Affect MEV

- Verification model is orthogonal to transaction ordering
- Light clients verify state transitions regardless of MEV extraction
- Users still face worse prices if attacked; they simply verify the attack occurred

9.4. Open Problems

Problem 1: Fair Transaction Ordering

- **Status:** Unsolved in bounded verification model
- **Mitigation:** Requires consensus-level changes (e.g., threshold encryption, fair ordering protocols)
- **Recommendation:** Adopt batch auctions or time-bucketing as sequencer policy

Problem 2: Censorship Resistance

- **Status:** Explicitly out of scope (see Section 3.2)
- **Mitigation:** Requires validator set diversity, slashing for censorship
- **Recommendation:** Social coordination; no protocol-level solution proposed

Problem 3: Cross-Domain MEV

- **Status:** Unsolved (e.g., MEV between DEX and lending protocols)
- **Mitigation:** Requires MEV smoothing or redistribution mechanisms
- **Recommendation:** Future work on MEV-aware protocol design

9.5. Honest Summary

What we solve:

- Bounded verification enables users to verify that MEV attacks occurred (state transitions are correct)
- Slippage protection limits damage from sandwich attacks

What we do NOT solve:

- Front-running (validators control ordering)
- JIT liquidity (AMM design issue)
- Censorship (explicitly out of scope)
- Fair ordering (requires consensus changes)

Our position: MEV is a hard problem that requires consensus-level solutions or economic redesign. Bounded verification provides transparency (users can verify they were attacked) but does not prevent MEV extraction. We view this as an acceptable trade-off given the benefits of bounded verification, but acknowledge it as a limitation.

10. Implementation Roadmap

We outline a phased engineering approach to implementing bounded verification DEX functionality.

10.1. Phase 0: Specification and Determinism

The foundation requires establishing canonical representations and deterministic computation. Deliverables include:

- Canonical encoding specification for intent message M
- Normative pseudocode for AMM swap mathematics

- Comprehensive invariant checks and rejection conditions
- Test vector suites covering:
    - Normal operation
    - Insufficient balance
    - Stale nonce
    - Slippage violations
    - Numerical edge cases (rounding, overflow)

Acceptance Criteria: Independent implementations must compute identical outputs (Δy values and rejection decisions) for all test vectors.

10.2. Phase 1: State Commitment Infrastructure

This phase implements the cryptographic commitment and proof mechanisms. Required components include:

- Key-value schema definitions for:
    - User balances (per token)
    - Pool reserves (per pool)
    - Nonces (per account)
- Proof format specifications:
    - Merkle membership proofs
    - Batch proof optimizations (shared ancestors)
- Verifier library functions:
    - `VerifyMembership(root, key, value, proof)`
    - `VerifySwap(old_state, new_state, intent)`

Acceptance Criteria: A browser-based verifier can validate swap correctness using only block headers (roots), proofs for touched keys, and the signed intent message.

10.3. Phase 2: Execution Layer Design

Two architectural approaches are viable for state transition execution:

Option A: On-Ledger Execution

- Blockchain's native consensus directly computes state transitions
- Account-chains execute swaps as native transactions
- Ordering ledger commits to account-chain states

Option B: Off-Ledger Execution

- External executor (sequencer) proposes state updates
- Validators check state transition validity
- Ordering ledger commits to validated states

Critical Specifications:

- Authorization policies for state update proposals
- Replay protection via nonce monotonicity

- Deterministic rounding rules for numerical computations

Acceptance Criteria: No two honest implementations can compute different post-states given identical pre-state and intent.

10.4. Phase 3: Proof Availability Infrastructure

Implement the proof storage and retrieval system (see Section 5.3):

- Account-chain storage for state and proof metadata
- Full node proof generation API
- Light client proof request protocol
- Proof caching and expiry policies

Acceptance Criteria: Light clients can request and verify proofs with <100ms latency in normal operation.

10.5. Phase 4: MEV Mitigation Strategies (Optional)

Implement optional MEV mitigations (see Section 9.2):

- Commit-reveal batch auctions
- Time-bucketed clearing
- User-signed price limits (already part of slippage mechanism)

Acceptance Criteria: Batch auctions demonstrably reduce front-running in simulated environments.

10.6. Phase 5: User Experience Design

The user-facing interface should abstract underlying complexity. Essential features include:

- Token discovery mechanisms (search, trending pairs)
- Streamlined single-click buy/sell operations
- Clear visualization of:
    - Price impact and slippage warnings
    - Verification status (verified, pending, degraded)
    - Estimated time to finality
- Frontier management:
    - Automatic header synchronization
    - Offline warnings
    - Re-bootstrap procedures

Acceptance Criteria: Users never encounter technical terminology (Merkle proofs, state roots, consensus mechanisms) during normal trading operations.

11. Comparative Analysis

11.1. Advantages Over Traditional Architectures

11.1.1. Reduced RPC Trust Assumptions

Conventional wallet implementations typically trust RPC providers for:

- Current state queries (balances, pool reserves)
- Transaction status (pending, confirmed, failed)
- Event logs (swap events, transfers)

Our bounded verification framework enables users to validate state changes against cryptographic commitments, reducing reliance on third-party infrastructure. While users may still query RPCs for convenience, they can verify critical state transitions independently.

11.1.2. Verification Complexity Bounded by Touched State

Traditional light clients often require processing overhead proportional to block size or transaction count. Our approach achieves O(m log N) complexity where m is touched state (typically 2-4 account-chains for swaps) and N is total state size.

This remains constant regardless of whether the network processes 10 or 10,000 transactions per block, providing predictable resource requirements for client devices.

11.1.3. Browser-Native Verification

The combination of bounded proof sizes (~5 KB) and minimal computational requirements (~10 ms) enables practical browser-based verification. A JavaScript implementation maintaining an O(k) header frontier can serve as a trustless verification layer for web-based trading interfaces.

This contrasts sharply with full node requirements or even header-chain light clients that scale with global activity.

11.1.4. Predictable Resource Requirements

Developers can reason about worst-case verification costs:

- Storage: O(k) for frontier size k
- Bandwidth: O(m log N) per transaction
- Computation: O(m log N) hashing + O(1) arithmetic

This predictability enables better UX design and capacity planning.

11.2. Trade-offs and Limitations

11.2.1. Reduced Transaction Identity Guarantees

Our framework emphasizes state transition effects over transaction identity. Users verify that their balance changed correctly but cannot always prove that a specific transaction hash was included in the canonical chain.

Block explorers in this model display state outcomes rather than comprehensive transaction logs. Some users may perceive this as reduced auditability, even though security properties depend on state validity rather than transaction identity.

11.2.2. Proof Availability Dependencies

Verification requires timely access to state proofs. If proof providers are unavailable or slow to respond, users may observe confirmed swaps (on-chain finality) before obtaining verification proofs.

User interfaces must handle this asynchrony gracefully, potentially showing "pending verification" states. In adversarial scenarios, malicious proof providers could selectively withhold proofs, though users could source proofs from alternative providers or regenerate them from available state data (see Section 5.3).

11.2.3. Frontier Expiry Challenges

Users who go offline longer than the retention window k face verification degradation. Upon reconnection, they must re-bootstrap their header frontier before verifying recent transactions.

This creates a failure mode absent in traditional architectures where RPC queries work regardless of client offline duration. Practical implementations require clear user warnings and streamlined re-synchronization procedures.

11.2.4. Censorship Detection Limitations

Our model does not provide censorship resistance proofs. If a user's transaction is censored, they cannot generate cryptographic proof of that censorship using bounded verification alone.

Users must rely on traditional mechanisms (transaction resubmission, alternative RPC endpoints, social coordination) to detect and respond to censorship. This is a conscious trade-off to achieve bounded computational complexity.

11.2.5. MEV Exposure

As analyzed in Section 9, our model does not solve MEV. Users remain vulnerable to front-running, sandwich attacks (mitigated by slippage protection), and other ordering-based attacks. This is a limitation shared with most DEX architectures.

11.3. Comparison Table

Property	Single-Ledger (Ethereum)	Dual-Ledger (Bounded Verification)
Light Client Storage	~100 MB (sync committee)	~200 KB (frontier)
Verification Complexity	O(block size) or trust validators	O(m log N), m = 2-4
Transaction Identity	Full (transaction hash proofs)	Limited (state transition only)
Censorship Detection	Possible via mempool monitoring	Not supported
Browser-Native	Difficult (complex crypto)	Practical (~10 ms, ~5 KB proofs)
MEV Exposure	High (global mempool)	Moderate (slippage protection)
Trust Assumptions	Validator honest majority	Consensus + proof availability

12. Related Work

Our work builds on several research areas in blockchain scalability and light client protocols.

12.1. Light Client Protocols

Traditional SPV (Simplified Payment Verification) clients validate block headers and Merkle proofs for specific transactions [1]. Our approach extends this paradigm to state verification rather than transaction verification.

Ethereum light clients using sync committees [3] or ZK proofs aim for similar trust minimization but often require more complex cryptography or validator set tracking. Our bounded model achieves simpler implementation at the cost of reduced scope (no censorship detection).

12.2. State Channel and Layer-2 Solutions

State channels and optimistic rollups [4, 5] also emphasize state transition validity over complete transaction history. However, these solutions typically require challenge periods and fraud proofs. Our framework operates within layer-1 consensus, using committed roots as the source of truth without additional challenge mechanisms.

12.3. Automated Market Maker Literature

Constant-product AMMs originated with Uniswap [2] and have been extensively analyzed for properties including impermanent loss, arbitrage dynamics, and MEV exposure [6, 8]. Our contribution is not to the AMM mechanism itself but to the verification layer, demonstrating how AMM operations can be verified with bounded computational requirements.

12.4. MEV Research

Flash Boys 2.0 [6] established the MEV threat model for decentralized exchanges. Our work accepts MEV as a consensus-level problem and focuses on verification transparency rather than MEV prevention. We view this as complementary to consensus-level MEV mitigation strategies [7].

12.5. Dual-Ledger and DAG-Based Architectures

Account-chain architectures (e.g., Nano, IOTA, Holochain) separate local execution from global ordering. Our contribution is demonstrating how this separation enables bounded verification for complex applications like DEXs, not just simple value transfers.

13. Future Work

Several directions could extend this research:

13.1. Zero-Knowledge Proofs

ZK proofs could compress state proofs further (constant-size proofs regardless of m) or enable privacy-preserving verification (hiding swap amounts while proving validity). This would require integrating ZK circuits for AMM verification.

13.2. Formal Verification

Formally verifying the AMM transition logic (e.g., using Coq or Isabelle) would increase confidence in implementation correctness and determinism. This is particularly important given the security-critical nature of DEX operations.

13.3. Cross-Chain Bounded Verification

Extending bounded verification to multi-chain DEX aggregators would enable users to verify swaps across different blockchain networks. This requires solving cross-chain state commitment and proof aggregation challenges.

13.4. Empirical Evaluation

Real-world deployment would validate theoretical complexity bounds and identify practical optimization opportunities. Key metrics include:

- Actual proof sizes in production environments
- Verification latency under network congestion
- User retention for light clients vs. full nodes

13.5. MEV-Aware Protocol Design

Integrating bounded verification with consensus-level MEV mitigation (e.g., threshold encryption, fair ordering) could provide both efficient verification and reduced MEV exposure. This is an open research problem.

14. Conclusion

We have presented a framework for bounded verification of DEX swap transactions that decouples verification complexity from global blockchain throughput. By focusing verification on state transition effects rather than transaction identity, we achieve O(m log N) proof size and computation, where m is the number of touched account-chains (typically 2-4) and N is total state size.

Key findings:

1. **Dual-ledger architecture is essential**: The separation between account-chain execution and ordering ledger consensus enables bounded verification properties unattainable in single-ledger systems.

2. **Practical browser-native verification**: Empirical estimates show ~5 KB proofs, ~10 ms verification time, and <10 MB monthly bandwidth for active traders.

3. **Concrete proof availability model**: Account-chain storage with deterministic re-derivability provides a feasible approach with well-defined failure modes.

4. **Honest MEV assessment**: Bounded verification does not solve MEV; slippage protection mitigates sandwich attacks, but front-running and other ordering-based attacks remain.

Our analysis demonstrates clear advantages in reduced RPC trust assumptions and predictable resource requirements. However, these benefits come with trade-offs including limited transaction identity guarantees, proof availability dependencies, and frontier expiry challenges.

For use cases where state validity is paramount and transaction auditability is secondary, bounded verification offers a compelling design point in the space of light client protocols. The engineering roadmap we have outlined provides a practical path toward implementation on dual-ledger blockchain architectures.

The appropriateness of these trade-offs depends on application requirements and threat models. We believe the transparency enabled by bounded verification (users can cryptographically verify state transitions) combined with the practical feasibility (browser-native, low bandwidth) makes this approach valuable for real-world DEX deployments.

References

[1] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[2] Adams, H., Zinsmeister, N., & Robinson, D. (2020). Uniswap v2 core. Technical report.

[3] Buterin, V. (2015). Ethereum: A next-generation smart contract and decentralized application platform. Ethereum White Paper.

[4] Poon, J., & Dryja, T. (2016). The Bitcoin Lightning Network: Scalable off-chain instant payments.

[5] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., & Felten, E. W. (2018). Arbitrum: Scalable, private smart contracts. In USENIX Security Symposium.

[6] Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., ... & Juels, A. (2020). Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In IEEE S&P.

[7] Bentov, I., Kumaresan, R., & Miller, A. (2017). Instantaneous decentralized poker. In ASIACRYPT.

[8] Zhang, Y., Chen, X., & Park, D. (2018). Formal specification of constant product market maker model and implementation. arXiv preprint arXiv:1811.11150.

[9] Ben-Sasson, E., Chiesa, A., Tromer, E., & Virza, M. (2014). Succinct non-interactive zero knowledge for a von Neumann architecture. In USENIX Security.

[10] Gavin Wood. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper.

Appendix A: Notation and Definitions

Account-Chain: A per-account ledger recording state transitions for a single address.

Ordering Ledger: A global consensus chain that establishes canonical ordering and commits to account-chain states.

Touched State (m): Number of account-chains modified by a transaction.

State Size (N): Total number of accounts in the system.

Frontier Size (k): Number of recent ordering ledger headers retained by a light client.

Verification Complexity: O(m log N) - hashing operations required to verify a transaction.

Proof Size: O(m log N) - bytes required to transmit Merkle proofs.

Bounded Inclusion: Verification that state values exist under committed roots, without proving transaction identity.

Appendix B: Threat Model Summary

Adversary Capability	Mitigation	Residual Risk
Malicious UI (phishing)	Client-side verification	User must use legitimate client
Invalid state proofs	Merkle proof verification	None (cryptographically prevented)
Eclipse attack	Multi-peer header sync	Requires majority node collusion
Chain reorganization	Confirmation depth k	Attacks within k blocks succeed
Front-running (MEV)	Slippage protection	Adversary can still order txs
Proof withholding	Query multiple providers	Requires at least one honest node
Frontier expiry	Re-synchronization	User inconvenience, no security loss

Appendix C: Implementation Checklist

For teams implementing bounded verification DEX:

- [ ] Phase 0: Deterministic AMM specification
- [ ] Phase 0: Test vectors for all edge cases
- [ ] Phase 1: Merkle tree implementation (Blake2b)
- [ ] Phase 1: Proof format specification
- [ ] Phase 1: JavaScript verifier library
- [ ] Phase 2: Account-chain execution layer
- [ ] Phase 2: Ordering ledger state commitment
- [ ] Phase 3: Full node proof generation API
- [ ] Phase 3: Light client proof request protocol
- [ ] Phase 4 (optional): Batch auction mechanism
- [ ] Phase 5: User interface (swap, verify, warnings)
- [ ] Phase 5: Frontier management (sync, expiry, re-bootstrap)
- [ ] Testing: Adversarial proof generation
- [ ] Testing: Frontier expiry scenarios
- [ ] Testing: MEV attack simulations
- [ ] Documentation: User guides (non-technical)
- [ ] Documentation: Developer API reference

END OF PAPER