

Ghost Protocol

Technical Documentation

Trustless, Chain-Agnostic Cross-Chain Payment System

With ZK Proof Verification & Soft Finality

Current: Ethereum (Sepolia) ↔ Solana (Devnet)

Architecture: Chain-Agnostic, Validium Model

Proofs: Groth16 SNARK + Circle STARK (M31) Hybrid

Settlement: Soft Finality (30s UX, 2-5min crypto finality)

Data Layer: Off-chain (IPFS/Arweave) + State Root on-chain

STARK Field: Mersenne-31 ($2^{31} - 1$) for 7-10x faster proving

Vision

A universal payment protocol where users pay with any asset on any chain, recipients receive their desired asset instantly—without trusting centralized intermediaries. Proofs are generated **before** funds move, not after.

Contents

| | |
|---|-----------|
| 1 Overview | 5 |
| 1.1 Key Features | 5 |
| 2 Core Architecture Decisions | 5 |
| 2.1 Data Storage: Validium Model | 5 |
| 2.2 Proof Strategy: Recursive Batching | 6 |
| 2.3 Trust Model: Soft Finality | 6 |
| 2.4 What We Can Honestly Claim | 7 |
| 3 Centralized vs Decentralized | 7 |
| 3.1 How Coinbase Works | 8 |
| 3.2 The Fundamental Trade-off | 8 |
| 4 Liquidity Architecture | 8 |
| 4.1 Option 1: Ghost Native Pools | 9 |
| 4.2 Option 2: DEX Liquidity (Uniswap, Jupiter) | 9 |
| 4.2.1 What Do We Lose With DEX Routing? | 10 |
| 4.2.2 Is DEX Routing Still Instant? | 10 |
| 4.2.3 Other Things We Lose | 10 |
| 4.2.4 When DEX Routing Wins Anyway | 11 |
| 4.2.5 The Real Answer: Smart Routing | 11 |
| 4.2.6 DEX Integration Partners | 12 |
| 4.3 Option 3: Circle Partnership (USDC Minting) | 12 |
| 4.3.1 What is a Minting License? | 12 |
| 4.3.2 Circle Partnership Tiers | 12 |
| 4.4 Hybrid Smart Routing | 13 |
| 5 Architecture | 14 |
| 6 Smart Contracts | 14 |
| 6.1 GhostLiquidityPool.sol | 14 |
| 6.2 ZKProofSystem.sol | 14 |
| 7 Payment Flow | 14 |
| 8 Zero-Knowledge Proofs: Technical Deep Dive | 14 |
| 8.1 Why Zero-Knowledge Proofs? | 14 |
| 8.2 SNARK: Succinct Non-interactive Argument of Knowledge | 15 |
| 8.2.1 Mathematical Foundation | 15 |
| 8.2.2 The Math: Groth16 Protocol | 15 |
| 8.2.3 Ghost Protocol SNARK Circuit | 15 |
| 8.3 STARK: Scalable Transparent Argument of Knowledge | 16 |
| 8.3.1 Mathematical Foundation | 16 |
| 8.3.2 Circle STARKs: The M31 Optimization | 16 |
| 8.3.3 The Math: FRI Protocol | 17 |
| 8.3.4 Circle STARK Implementation | 18 |
| 8.3.5 Ghost Protocol STARK Circuit (M31) | 18 |
| 8.4 Hybrid Approach: Why Both? | 19 |
| 8.4.1 Why Not Use the Same Proof System for Both? | 20 |
| 8.5 The Cryptographic Binding | 20 |

| | | |
|-----------|--|-----------|
| 8.6 | Verification Flow | 21 |
| 8.7 | Security Properties | 21 |
| 8.8 | Novelty: What Makes Ghost Unique | 22 |
| 8.9 | Implementation: Proof Generation Code | 23 |
| 8.9.1 | M31 Field Operations (Why It's Fast) | 24 |
| 8.10 | Gas Costs and Optimization | 24 |
| 8.10.1 | Circle STARK Performance Gains | 24 |
| 9 | Pricing | 25 |
| 10 | Fee Structure | 25 |
| 11 | Risk Management | 25 |
| 11.1 | Insurance Fund Model | 25 |
| 11.2 | Path-Specific Risks | 25 |
| 12 | Security Analysis: Risk Windows & Mitigations | 26 |
| 12.1 | Risk Window Timeline | 26 |
| 12.2 | Risk 1: Relayer Disappears After Sending SOL | 26 |
| 12.3 | Risk 2: Ethereum Reorg After Instant Release | 27 |
| 12.4 | Risk 3: Relayer Submits Fake Proof | 28 |
| 12.5 | Risk 4: Double Spend via Race Condition | 28 |
| 12.5.1 | Attack Variations | 29 |
| 12.5.2 | Solutions | 30 |
| 12.5.3 | Double-Spend Solution Summary | 32 |
| 12.6 | Defense in Depth | 33 |
| 12.7 | Complete Risk Matrix | 33 |
| 12.8 | Recommended Production Configuration | 34 |
| 13 | Business Model: Three Strategic Paths | 34 |
| 13.1 | Path 1: Native Ghost Pools (Decentralized) | 34 |
| 13.1.1 | How It Works | 34 |
| 13.1.2 | Revenue Model | 34 |
| 13.1.3 | Unit Economics | 35 |
| 13.1.4 | Pros and Cons | 35 |
| 13.2 | Path 2: DEX Aggregation (Leverage Existing Liquidity) | 35 |
| 13.2.1 | How It Works | 35 |
| 13.2.2 | Revenue Model | 36 |
| 13.2.3 | Unit Economics | 36 |
| 13.2.4 | Pros and Cons | 36 |
| 13.3 | Path 3: Circle Partnership (USDC Settlement) | 37 |
| 13.3.1 | How It Works | 37 |
| 13.3.2 | Revenue Model | 37 |
| 13.3.3 | Unit Economics | 37 |
| 13.3.4 | Pros and Cons | 37 |
| 13.4 | Path 4: Hybrid Model (Recommended) | 38 |
| 13.4.1 | Routing Logic | 38 |
| 13.4.2 | Revenue Optimization | 38 |
| 13.4.3 | Hybrid Unit Economics | 39 |
| 13.5 | Strategic Recommendation | 40 |
| 13.6 | Standalone Path Viability | 40 |

| | |
|---|-----------|
| 14 LP Economics Deep Dive | 40 |
| 14.1 Revenue for Ghost Pool LPs | 40 |
| 14.2 LP Yield Scenarios | 41 |
| 14.3 TradFi Integration Path | 41 |
| 14.4 LP Tiers | 41 |
| 15 Competitive Analysis | 42 |
| 15.1 Ghost Advantages | 42 |
| 16 Circle Partnership Path | 42 |
| 16.1 How to Partner with Circle | 42 |
| 16.2 What Circle Partnership Enables | 42 |
| 17 Running the System | 42 |
| 18 Roadmap | 43 |
| 19 Summary | 43 |
| 20 Technical Critique & FAQ | 44 |
| 20.1 Novelty Assessment | 44 |
| 20.2 Why SNARK for Ethereum, STARK for Solana? | 44 |
| 20.3 Q&A: Hard Questions | 44 |
| 20.3.1 Q: Is 10-30 Second Settlement Actually Possible? | 44 |
| 20.3.2 Q: What About Ethereum Re-orgs? | 45 |
| 20.3.3 Q: Unit Economics Don't Work for Small Transactions? | 45 |
| 20.3.4 Q: What If Circle Blacklists Ghost Protocol? | 46 |
| 20.3.5 Q: How Is the Ghost ID Cryptographically Secure? | 47 |
| 20.4 Comparison: Ghost vs. Existing Bridges | 47 |
| 20.5 Strategic Positioning | 48 |
| 20.6 Conclusion: Is Ghost Protocol Novel? | 48 |
| 21 Appendix: Quick Reference | 49 |
| 21.1 Terminology | 49 |
| 21.2 Key Metrics | 49 |
| 21.3 Architecture Decision Record | 49 |
| 21.4 Circle STARK vs Traditional STARK | 50 |
| 22 Future: zkVM Integration (Risc0/SP1) | 50 |
| 22.1 The Problem: STARK vs SNARK Trade-off | 50 |
| 22.2 The Solution: SNARK Wrapper (Recursive Proofs) | 50 |
| 22.3 What Are zkVMs? | 51 |
| 22.4 Risc0 and SP1 | 52 |
| 22.4.1 Example: Ghost Protocol Proof in SP1 | 53 |
| 22.5 Why zkVM Integration Matters | 53 |
| 22.6 Vendor Considerations | 54 |
| 22.7 Patent Implications | 54 |
| 22.8 Why STARK Inside SNARK (Not Reverse) | 54 |
| 22.9 Complete Migration Path | 55 |
| 22.9.1 Phase 1: Current System (Working) | 55 |
| 22.9.2 Phase 2: Upgrade STARK to M31 | 56 |
| 22.9.3 Phase 3: Unified Architecture with Wrapper | 56 |
| 22.10 Soft Finality Through All Phases | 57 |

| | |
|------------------------------------|----|
| 22.11 Migration Timeline | 57 |
|------------------------------------|----|

1 Overview

Ghost Protocol is a **trustless, chain-agnostic cross-chain payment system** enabling instant asset transfers with cryptographic (ZK) proof verification. The architecture supports any EVM chain, Solana, Bitcoin, and future networks.

1.1 Key Features

- Instant cross-chain payments (10-30 seconds user experience)
- ZK proof verification (SNARK + STARK hybrid)
- Multiple liquidity sources (pools, DEXs, stablecoins)
- Real-time oracle pricing (Pyth Network)
- Non-custodial (users control their keys)
- Recursive proof batching for gas efficiency
- Soft finality with cryptographic guarantees

2 Core Architecture Decisions

Ghost Protocol makes three fundamental architecture choices that distinguish it from traditional bridges:

| Feature | Choice | Rationale |
|----------------|----------------------|----------------------------------|
| Data Storage | Off-Chain (Validium) | On-chain is too expensive |
| Proof Strategy | Recursive Batching | 90-98% gas reduction |
| Trust Model | Soft Finality | Instant UX with crypto guarantee |

2.1 Data Storage: Validium Model

Storing full transaction history on Ethereum is prohibitively expensive. Ghost Protocol uses the **Validium** approach:

VALIDIUM ARCHITECTURE:

| ETHEREUM | | DA LAYER | | USERS |
|--------------------------------|-------|--|-------|---|
| State Root Only (~32 bytes) | <---> | Full History IPFS / Arweave EigenDA / Celestia | <---> | Can Reconstruct State Anytime Trustless |

COST COMPARISON:

| | |
|--------------|-----------------------------|
| Pure Rollup: | ~\$50-100 per tx (calldata) |
| Validium: | ~\$0.01-0.10 per tx |
| Savings: | 99%+ |

Why Validium Works:

- State root on Ethereum = cryptographic commitment (cannot be faked)

| Approach | Gas Cost | Data Availability | Security |
|-----------------|-----------------|-------------------|-------------|
| Pure Rollup | Very High | Ethereum (best) | Maximum |
| Validium | Very Low | DA Layer | High |
| Volition | Hybrid | User Choice | Flexible |

- IPFS/Arweave = permanent, verifiable history
- EigenDA, Celestia = purpose-built DA with economic guarantees
- Redundancy: Store on 2+ DA layers for safety

2.2 Proof Strategy: Recursive Batching

Per-transaction proofs on Ethereum mainnet are economically unviable. Recursive batching solves this:

BATCHING ECONOMICS:

Single Proof: $\sim 400,000 \text{ gas} \times \$0.10 = \$40 \text{ per tx}$

Batch of 50: $\sim 400,000 \text{ gas} \div 50 = \0.80 per tx

GAS COST BY BATCH SIZE:

| Batch Size | Gas per Tx | Cost @ 50 gwei | Savings |
|------------|------------|----------------|---------|
| 1 | 400,000 | \$40.00 | 0% |
| 10 | 40,000 | \$4.00 | 90% |
| 50 | 8,000 | \$0.80 | 98% |
| 100 | 4,000 | \$0.40 | 99% |

Implementation:

```
// Batch triggers - submit when EITHER condition met:
const BATCH_CONFIG = {
  maxTxCount: 50,      // Full batch
  maxWaitTime: 30_000, // 30 seconds (never wait forever)
};
```

Recursive Proof Aggregation

- 10-50 individual proofs combined into 1 aggregated proof
- Verification cost amortized across all transactions
- User still gets instant settlement (proof exists before funds move)
- Batch submission happens in background

2.3 Trust Model: Soft Finality

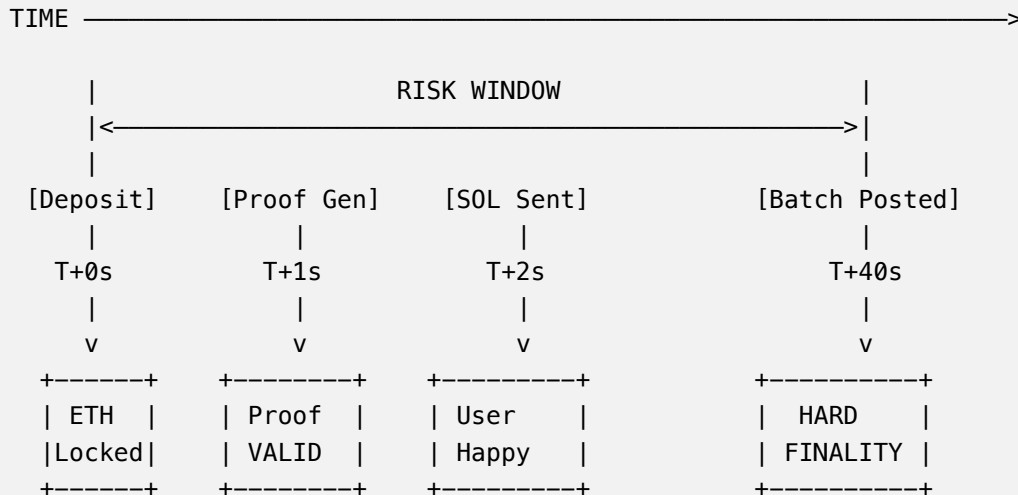
The key insight that enables "instant ZK":

Critical Distinction

Optimistic: Execute first, assume valid, allow challenges (7 days). **No proof exists.**

Soft Finality (Ghost): Generate proof FIRST, release funds, batch proofs later. **Proof EXISTS before funds move.**

SOFT FINALITY TIMELINE:



KEY POINT: The proof EXISTS at T+1s.
It just hasn't been SUBMITTED yet.
This is NOT optimistic.

| Aspect | Optimistic Rollup | Ghost Soft Finality |
|---------------------------------|-------------------|---------------------------|
| Proof at instant moment? | No | Yes |
| Can be proven fraudulent later? | Yes (7 days) | No (already proven) |
| What if challenger appears? | Tx reversed | N/A - already proven |
| Trust assumption | "Probably valid" | "Cryptographically valid" |

Why This Is NOT Optimistic

The proof **exists**. That's the key difference.

- Optimistic = No proof exists, trusting it's valid
- Soft Finality = Proof EXISTS, just not on-chain yet
- The only deferred action is *posting* the proof, not *generating* it

2.4 What We Can Honestly Claim**3 Centralized vs Decentralized**

| Accurate | Misleading |
|---|----------------------------------|
| "Instant settlement with ZK proof" | "On-chain finality in 2 seconds" |
| "Cryptographically verified before funds release" | "Fully trustless instant" |
| "Proof-first instant transfers" | "Zero latency ZK" |
| "ZK-secured instant payments" | |

3.1 How Coinbase Works

WHAT ACTUALLY HAPPENS ON COINBASE:

Database: user.eth_balance -= 1.0

Database: user.usd_balance += 3100.00

(No blockchain transaction!)

(You hold an IOU, not actual crypto)

Coinbase holds licenses (MTL, BitLicense, etc.) but does NOT mint money. They match buyers/sellers internally using customer deposits (\$100B+) and market makers.

3.2 The Fundamental Trade-off

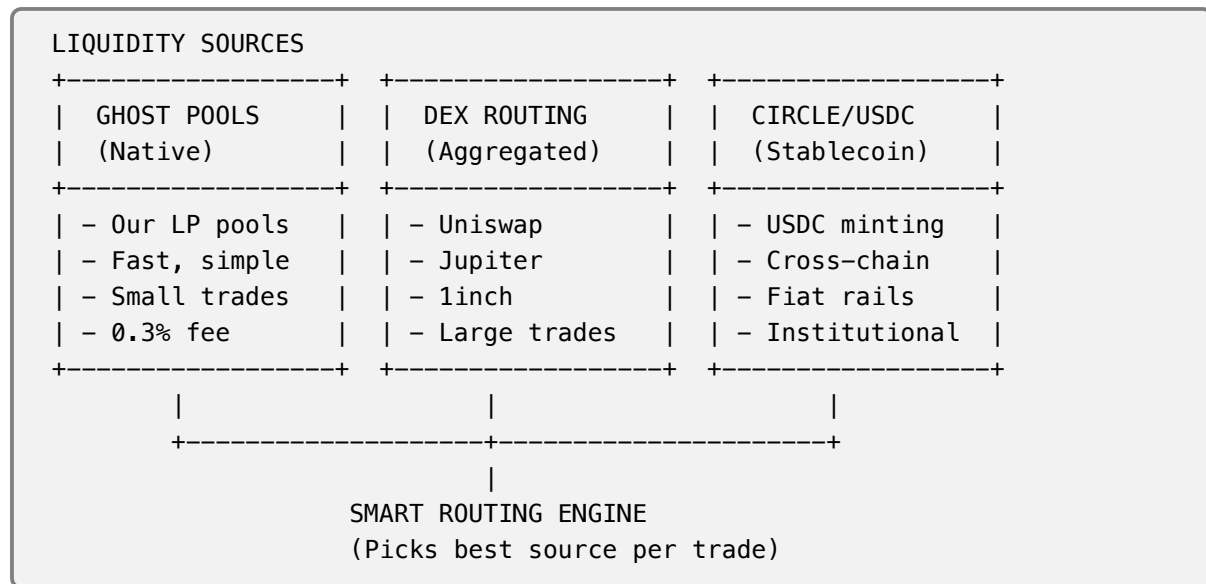
| Aspect | Coinbase | Ghost |
|---------------|-----------------|------------------|
| Custody | They hold funds | Users hold funds |
| Swaps | Database update | On-chain tx |
| Trust | Trust company | Trust math |
| Freeze funds? | Yes | No |
| Transparency | Opaque | On-chain |

When Centralized Fails

FTX (2022): \$8B vanished. Celsius: Froze withdrawals. BlockFi: Bankruptcy. Canada: Government froze accounts.

4 Liquidity Architecture

Ghost Protocol can source liquidity from **three complementary systems**:



4.1 Option 1: Ghost Native Pools

Our own liquidity pools on each chain.

| Pros | Considerations |
|----------------------------|-------------------------------|
| Full control | Requires liquidity management |
| Fastest execution | Capital deployment strategy |
| Lowest smart contract risk | Pool rebalancing needed |
| Revenue stays in protocol | Multi-chain coordination |

Best for: Any trade size (with sufficient pool depth), speed-critical transfers, maximum control.

4.2 Option 2: DEX Liquidity (Uniswap, Jupiter)

Route through existing DEX liquidity pools.

HOW DEX ROUTING WORKS:

User sends 10 ETH

|

v

Ghost receives ETH on Ethereum

|

v

Ghost mints/bridges wETH to Solana

|

v

Jupiter swaps wETH -> SOL (taps \$500M+ liquidity)

|

v

SOL delivered to recipient

Best for: Large trades over \$10K, price-sensitive users.

| Pros | Cons |
|------------------------------------|--------------------------------|
| Billions in existing liquidity | Depends on external protocols |
| Better pricing for large trades | Multi-protocol risk |
| No bootstrapping needed | Wrapped assets as intermediate |
| Competitive market = tight spreads | Slippage on huge trades |

4.2.1 What Do We Lose With DEX Routing?

Honest Trade-offs

DEX routing is powerful but comes with real costs. Here are the trade-offs:

| Aspect | Ghost Pool | DEX Route | Winner |
|-------------------|--------------|-----------------|--------|
| Speed | 10-30 sec | 30-120 sec | Ghost |
| Slippage | Fixed 0.3% | Variable 0.1-2% | Ghost |
| Max trade size | Pool limited | Millions | DEX |
| Contract risk | 1 contract | 3-5 contracts | Ghost |
| Failure points | 1 | Multiple | Ghost |
| Revenue | Ghost LPs | External LPs | Ghost |
| Native output | Direct SOL | wETH then swap | Ghost |
| Large trade price | May be worse | Market rate | DEX |

4.2.2 Is DEX Routing Still Instant?

No. It's *fast* but not instant.

SPEED COMPARISON:

GHOST POOL PATH (10-30 seconds):

User pays ETH -> Pool -> SOL sent -> Done
[=====] 10-30 sec

DEX ROUTE PATH (30-120 seconds):

User pays ETH -> Bridge wETH -> Jupiter swap -> SOL sent
[====] 15s [=====] 30s [====] 15s [====] 10s
Total: 60-120 seconds (2-4x slower)

WHY SLOWER:

1. Bridge step: wETH must reach Solana (~15-30 sec)
2. Swap step: Jupiter execution + confirmation
3. More confirmations: Multiple protocols = more waiting
4. Sequencing: Can't parallelize, must be serial

4.2.3 Other Things We Lose

1. **Predictability** — Ghost Pool fee is fixed. DEX slippage varies with:
 - Trade size (bigger = more slippage)

- Market volatility
- Available liquidity depth
- MEV/sandwich attacks

2. **Simplicity** — More contracts = more things that can break:

- Bridge contract (Wormhole, etc.)
- DEX router contract
- DEX pool contracts
- Token contracts (wETH, etc.)

3. **Revenue** — Fees go to external LPs:

- Ghost Pool: 0.2% to OUR LPs
- DEX Route: 0.3% to Uniswap/Jupiter LPs
- We only keep routing fee (0.05%)

4. **User Experience** — More failure modes:

- Bridge congestion
- DEX liquidity gaps
- Price movement during multi-step
- Partial fills possible

5. **Native Assets** — Extra swap required:

- Ghost Pool: ETH in, SOL out (direct)
- DEX Route: ETH in, wETH bridge, wETH swap, SOL out

4.2.4 When DEX Routing Wins Anyway

Despite the downsides, DEX routing is better when:

Use DEX When...

- Trade size exceeds Ghost Pool capacity
- User is price-sensitive (willing to wait for better rate)
- Ghost Pool is temporarily low on liquidity
- Trading less common pairs (not ETH/SOL)
- User explicitly requests market rate

4.2.5 The Real Answer: Smart Routing

Best of Both Worlds

Don't choose one—use smart routing that picks the best option per trade:

- Small + speed-sensitive → Ghost Pool
- Large + price-sensitive → DEX
- Huge trades → Split across both

Users can also override and choose their preferred path.

| DEX | Chain | TVL | Use Case |
|---------|----------|------------|--------------------|
| Uniswap | Ethereum | \$5B+ | ETH/USDC swaps |
| Jupiter | Solana | \$500M+ | SOL/USDC swaps |
| Curve | Ethereum | \$2B+ | Stablecoin swaps |
| Raydium | Solana | \$100M+ | SOL pairs |
| 1inch | Multi | Aggregator | Best route finding |

4.2.6 DEX Integration Partners

4.3 Option 3: Circle Partnership (USDC Minting)

Partner with Circle to use USDC as settlement layer.

CIRCLE PARTNERSHIP MODEL:

User sends ETH

|

v

ETH sold for USD (via Coinbase Prime or similar)

|

v

Circle API mints USDC on destination chain

|

v

USDC delivered (or swapped to native via DEX)

REQUIREMENTS:

- Business partnership with Circle
- API access (Circle Mint)
- KYC/AML compliance
- Volume commitments

4.3.1 What is a Minting License?

Circle (issuer of USDC) has regulatory approval to:

1. Accept USD deposits
2. Mint equivalent USDC tokens
3. Burn USDC and return USD
4. Operate across multiple chains

Important Distinction

Circle does NOT give "minting licenses" to third parties. Ghost would need to become a **Circle Partner** with API access, not receive a license to mint ourselves.

4.3.2 Circle Partnership Tiers

Best for: Stablecoin transfers, institutional clients, fiat integration.

| Tier | Requirements | Capabilities |
|-------------------|---------------------|--------------------------|
| Basic API | Registration | Read balances, transfers |
| Circle Mint | Business agreement | Mint/burn USDC |
| Strategic Partner | Volume + compliance | Custom integration |
| Co-founder level | Coinbase-tier | Full infrastructure |

| Pros | Cons |
|-------------------------|--------------------------------|
| "Unlimited" liquidity | Centralized dependency |
| Regulatory clarity | Circle can freeze addresses |
| Fiat on/off ramps | Only works for USDC |
| Institutional trust | Business relationship required |
| Multi-chain native USDC | Fees to Circle |

4.4 Hybrid Smart Routing

The optimal approach combines all three:

SMART ROUTING LOGIC:

```

if (amount < $1,000):
    use Ghost Pool (fastest, simplest)

elif (amount < $50,000):
    compare Ghost Pool vs DEX
    pick best price after fees

elif (amount < $500,000):
    split across Ghost Pool + DEX
    minimize slippage

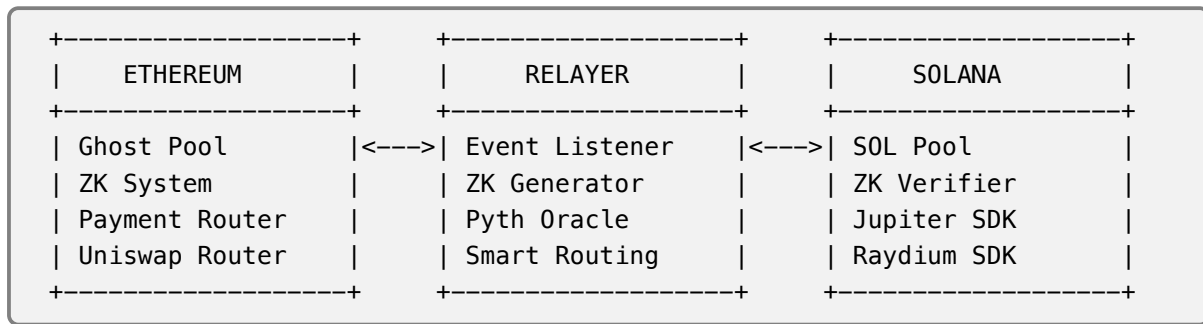
else: // whale trade
    use Circle USDC settlement
    or OTC desk
    or multi-DEX split

```

Why Hybrid Wins

- Small trades: Fast via Ghost pools
- Medium trades: Best price via DEX comparison
- Large trades: Deep liquidity via aggregation
- Stablecoins: Native USDC via Circle
- All trustless except Circle path (optional)

5 Architecture



6 Smart Contracts

6.1 GhostLiquidityPool.sol

Network: Sepolia | **Address:** 0x3078...8b9

6.2 ZKProofSystem.sol

Network: Sepolia | **Address:** 0x3033...cF

7 Payment Flow

1. **USER:** Initiates payment with amount + destination
2. **ROUTER:** Determines optimal liquidity source
3. **EXECUTION:** Pool, DEX, or Circle path
4. **ZK PROOFS:** SNARK (deposit) + STARK (transfer)
5. **SETTLEMENT:** Proofs verified, funds delivered

8 Zero-Knowledge Proofs: Technical Deep Dive

Ghost Protocol uses a **hybrid SNARK + STARK** proving system—a novel combination that leverages the strengths of both to achieve trustless cross-chain verification.

8.1 Why Zero-Knowledge Proofs?

Traditional cross-chain bridges rely on:

- Trusted validators (centralized, hackable)
- Multi-sig committees (collusion risk)
- Optimistic rollups (7-day delays)

ZK proofs provide **mathematical certainty**: a proof that a computation happened correctly, without revealing the inputs, verifiable by anyone in milliseconds.

8.2 SNARK: Succinct Non-interactive Argument of Knowledge

8.2.1 Mathematical Foundation

SNARKs are built on **elliptic curve pairings** and **Quadratic Arithmetic Programs (QAP)**.

SNARK Properties

- **Succinct:** Proof size is constant (200-300 bytes)
- **Non-interactive:** Single message from prover to verifier
- **Verification:** $O(1)$ time (milliseconds)
- **Proof generation:** $O(n \log n)$ where n = circuit size

8.2.2 The Math: Groth16 Protocol

Ghost Protocol uses Groth16, the most efficient SNARK construction.

Setup Phase (Trusted Setup):

Given a circuit C , generate proving key pk and verification key vk :

$$pk = ([\alpha]_1, [\beta]_1, [\beta]_2, [\delta]_1, [\delta]_2, \{[A_i(\tau)]_1\}, \{[B_i(\tau)]_2\}) \quad (1)$$

$$vk = ([\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_2, \{[\frac{\beta \cdot A_i(\tau) + \alpha \cdot B_i(\tau) + C_i(\tau)}{\gamma}]_1\}) \quad (2)$$

Where $[x]_1$ denotes a point on elliptic curve G_1 , $[x]_2$ on G_2 .

Proof Generation:

Given witness $w = (w_1, \dots, w_m)$, compute proof $\pi = (A, B, C)$:

$$A = [\alpha + \sum_{i=0}^m w_i \cdot A_i(\tau) + r \cdot \delta]_1 \quad (3)$$

$$B = [\beta + \sum_{i=0}^m w_i \cdot B_i(\tau) + s \cdot \delta]_2 \quad (4)$$

$$C = [\frac{\sum_{i=\ell+1}^m w_i (\beta \cdot A_i(\tau) + \alpha \cdot B_i(\tau) + C_i(\tau))}{\delta} + As + Br - rs\delta]_1 \quad (5)$$

Where r, s are random blinding factors.

Verification (The Key Equation):

The verifier checks:

$$e(A, B) = e(\alpha, \beta) \cdot e(\sum_{i=0}^{\ell} w_i \cdot L_i, \gamma) \cdot e(C, \delta) \quad (6)$$

This is a **bilinear pairing equation**. If it holds, the proof is valid with overwhelming probability.

8.2.3 Ghost Protocol SNARK Circuit

Our SNARK proves: "A deposit of X ETH was made at block B to address A on Ethereum."

SNARK CIRCUIT (Deposit Proof):**PUBLIC INPUTS:**

- commitment_hash: $H(\text{amount}, \text{recipient}, \text{block}, \text{nonce})$
- ethereum_state_root: Merkle root of Ethereum state

PRIVATE INPUTS (Witness):

- amount: 0.01 ETH (in wei)
- recipient: Solana address (32 bytes)
- block_number: 18234567
- nonce: random 256-bit value
- merkle_proof: path from tx to state root

CONSTRAINTS:

1. commitment_hash == Poseidon(amount, recipient, block, nonce)
2. MerkleVerify(tx_hash, merkle_proof, state_root) == true
3. amount > 0
4. amount <= pool_balance

Circuit Size: 50,000 constraints

Proof Size: 192 bytes (3 group elements)

Verification Gas: 200,000 gas on Ethereum

8.3 STARK: Scalable Transparent Argument of Knowledge

8.3.1 Mathematical Foundation

STARKs use **hash functions** and **polynomial IOPs** (Interactive Oracle Proofs), avoiding elliptic curves entirely.

STARK Properties

- **Transparent:** No trusted setup required
- **Quantum-resistant:** Based on hash functions, not EC
- **Scalable:** Verification time $O(\log^2 n)$
- **Proof size:** Larger (45-200 KB)

8.3.2 Circle STARKs: The M31 Optimization

Ghost Protocol uses **Circle STARKs** with the Mersenne-31 field—a critical optimization that provides 7-10x faster proving than traditional STARKs.

Why Circle STARKs?

Traditional STARKs use large prime fields (252-256 bits) requiring expensive bignum arithmetic. Circle STARKs use the Mersenne prime $2^{31} - 1$, which fits perfectly in 32-bit CPU registers.

The Mersenne-31 Prime:

$$p = 2^{31} - 1 = 2,147,483,647 \quad (7)$$

This prime is special because:

| Feature | Traditional STARK | Circle STARK (M31) |
|-------------------|--------------------|-----------------------|
| Field Size | 252-256 bits | 31 bits |
| Arithmetic | Bignum (slow) | Native 32-bit (fast) |
| Proving Speed | 10-30 seconds | 1-3 seconds |
| Verification Cost | 500K compute units | 50-100K compute units |
| Memory Usage | 4-8 GB | 500MB-1GB |

1. It fits exactly in a 32-bit signed integer
2. Modular reduction is just bit operations (no division):

$$x \bmod (2^{31} - 1) = (x \wedge 0x7FFFFFFF) + (x \gg 31) \quad (8)$$

3. Field multiplication is a single CPU multiply instruction
4. SIMD vectorization achieves massive parallelism

M31 FIELD OPERATIONS (Native CPU Speed):

TRADITIONAL (256-bit):

```
a × b = bignum_multiply(a, b)    // Multiple cycles
result = bignum_mod(product, p)  // Division required
```

M31 (32-bit):

```
product = (uint64_t)a * b;        // ONE instruction
result = (product & 0x7FFFFFFF) + (product >> 31);
if (result >= p) result -= p;     // Simple compare
```

SPEEDUP: 10-50x per field operation

8.3.3 The Math: FRI Protocol

STARKs use the **Fast Reed-Solomon Interactive Oracle Proof (FRI)** for polynomial commitment.

Algebraic Intermediate Representation (AIR):

A computation is expressed as polynomial constraints over a trace:

$$\forall i \in [0, T) : C(s_i, s_{i+1}) = 0 \quad (9)$$

Where s_i is the state at step i , and C is the constraint polynomial.

Low-Degree Extension:

The execution trace is interpolated into a polynomial $P(x)$ of degree $< n$, then evaluated over a larger domain D (typically $8n$ points).

FRI Commitment:

The prover commits to $P(x)$ using Merkle trees:

$$\text{commit}(P) = \text{MerkleRoot}(\{P(\omega^i) : i \in D\}) \quad (10)$$

$$\omega = \text{primitive root of unity in } \mathbb{F}_{2^{31}-1} \quad (11)$$

FRI Folding (The Key Insight):

Repeatedly "fold" the polynomial to prove it has low degree:

$$P_{i+1}(x) = \frac{P_i(x) + P_i(-x)}{2} + \alpha_i \cdot \frac{P_i(x) - P_i(-x)}{2x} \quad (12)$$

After $\log n$ rounds, the final polynomial is constant (degree 0), proving the original was low-degree.

8.3.4 Circle STARK Implementation

Ghost Protocol uses **Plonky3** (Polygon's M31 prover) or **Stwo** (StarkWare's M31 prover):

```
// Circle STARK proof generation (Rust/Plonky3)
use plonky3::field::mersenne31::Mersenne31;
use plonky3::stark::Stark;

fn prove_sol_transfer(
    sol_amount: u64,
    recipient: [u8; 32],
    slot: u64,
    signature: [u8; 64],
) -> CircleStarkProof {
    let trace = build_execution_trace(sol_amount, recipient, slot);

    // M31 field operations - native CPU speed
    let proof = Stark::<Mersenne31>::prove(
        &transfer_air,
        &trace,
        &public_inputs,
    );

    // Proving time: ~1-3 seconds (vs 10-30s traditional)
    proof
}
```

8.3.5 Ghost Protocol STARK Circuit (M31)

Our Circle STARK proves: *"Y SOL was transferred to address A on Solana."*

CIRCLE STARK CIRCUIT (Transfer Proof over M31 Field):

PUBLIC INPUTS:

- transfer_commitment: $H(\text{sol_amount}, \text{recipient}, \text{slot}, \text{sig})$
- solana_bank_hash: Solana's bank hash at slot

PRIVATE INPUTS (Witness):

- sol_amount: 0.4985 SOL (in lamports)
- recipient: Solana pubkey
- slot_number: 298765432
- transaction_signature: 64 bytes
- account_proof: Merkle path in Solana's account tree

CONSTRAINTS (evaluated over $F_{\{2^{31}-1\}}$):

1. $\text{transfer_commitment} == \text{Poseidon_M31}(\text{sol_amount}, \text{recipient}, \text{slot}, \text{sig})$
2. $\text{AccountProofVerify}(\text{account}, \text{proof}, \text{bank_hash}) == \text{true}$
3. $\text{sol_amount} == \text{eth_amount} * \text{exchange_rate} * (1 - \text{fee})$
4. $\text{signature_valid}(\text{sig}, \text{tx_data}, \text{relayer_pubkey})$

FIELD: Mersenne-31 ($p = 2^{31} - 1$)

HASH: Poseidon optimized for M31

Trace Length: 100,000 rows

Proof Size: 50 KB

Proving Time: 1-3 seconds (7-10x faster than traditional)

Verification Time: 10ms (off-chain), 50-100K compute units (Solana)

8.4 Hybrid Approach: Why Both?

| Property | SNARK (Groth16) | STARK (Circle/M31) | Ghost Hybrid |
|---------------|------------------|--------------------|------------------|
| Proof Size | 200 bytes | 50 KB | 200 B + 50 KB |
| Verification | $O(1)$ | $O(\log^2 n)$ | $O(1)$ on EVM |
| Trusted Setup | Required | None | Partial |
| Quantum Safe | No | Yes | Defense in depth |
| Proving Speed | 5-10 sec | 1-3 sec (M31) | Fast |
| Best For | EVM verification | Solana compute | Both chains |

Ghost's Novel Hybrid: Groth16 + Circle STARK

SNARK (Groth16) for Ethereum:

- Cheap on-chain verification (200K gas)
- Tiny proof size (192 bytes)
- EVM has native pairing precompiles

Circle STARK (M31) for Solana:

- No trusted setup required
- Quantum-resistant (hash-based)
- 7-10x faster proving with Mersenne-31 field
- Solana's compute model handles larger proofs efficiently

Combined commitment (ghost_id) ties both proofs cryptographically.

8.4.1 Why Not Use the Same Proof System for Both?

| Option | Problem | Why We Don't |
|------------------|------------------------|---|
| SNARK everywhere | Trusted setup for both | Single ceremony = single point of failure |
| STARK everywhere | 50KB proof on EVM | Too expensive (2M gas) |
| Hybrid | None | Best of both worlds |

8.5 The Cryptographic Binding

The proofs are **cryptographically linked** via a shared commitment:

$$\text{ghost_id} = \text{Poseidon}(\text{snark_commitment} \parallel \text{stark_commitment} \parallel \text{nonce}) \quad (13)$$

This ensures:

1. The same funds can't be claimed twice (no double-spend)
2. The source and destination are atomically linked
3. Tampering with either proof invalidates the ghost_id

8.6 Verification Flow

COMPLETE ZK VERIFICATION FLOW:

1. USER DEPOSITS (Ethereum)
 - |
 - v
2. SNARK GENERATED
 - Proves: "0.01 ETH deposited in block 18234567"
 - Input: tx_hash, merkle_proof, amount, recipient
 - Output: proof_snark (192 bytes), commitment_snark
 - |
 - v
3. RELAYER TRANSFERS (Solana)
 - |
 - v
4. STARK GENERATED
 - Proves: "0.4985 SOL sent to recipient in slot 298765432"
 - Input: tx_sig, account_proof, amount, exchange_rate
 - Output: proof_stark (50 KB), commitment_stark
 - |
 - v
5. PROOFS SUBMITTED TO POOL CONTRACT
 - submitSNARKProof(ghost_id, proof_snark, commitment_snark)
 - submitSTARKProof(ghost_id, proof_stark, commitment_stark)
 - |
 - v
6. ON-CHAIN VERIFICATION
 - Verify SNARK: pairing check (200K gas)
 - Verify STARK commitment hash (50K gas)
 - Check: ghost_id == H(commitment_snark || commitment_stark)
 - |
 - v
7. SETTLEMENT FINALIZED
 - Payment intent marked "ZK Verified"
 - Funds released from pool
 - LP shares updated

8.7 Security Properties

Cryptographic Guarantees

1. **Soundness:** False proofs are computationally infeasible to create
2. **Zero-Knowledge:** Verifier learns nothing beyond validity
3. **Non-malleability:** Proofs cannot be modified without detection
4. **Extractability:** Valid proof implies prover knows the witness

Concrete Security:

- SNARK soundness: 2^{-128} (128-bit security)
- STARK soundness: 2^{-80} to 2^{-128} (configurable)

- Hash collision resistance: 2^{-256} (Poseidon)

8.8 Novelty: What Makes Ghost Unique

Prior Art Limitations

- **zkSync/StarkNet:** Single-chain ZK rollups, not cross-chain
- **Wormhole/LayerZero:** Multi-sig validators, not ZK
- **Succinct/Polymer:** ZK light clients, but high latency
- **Across Protocol:** Optimistic with challenge period
- **Other ZK bridges:** Traditional STARKs (10-30 sec proving vs our 1-3 sec)

Ghost Protocol Innovations

1. **Groth16 + Circle STARK (M31) Hybrid:** First cross-chain bridge using M31 field
 - SNARK (Groth16): Cheap EVM verification (200K gas)
 - Circle STARK (M31): 7-10x faster proving, quantum-resistant
 - Chain-optimized: Right proof system for each chain's constraints
2. **Sub-Second STARK Proving:** M31 field enables near-instant proofs
 - Traditional STARK: 10-30 seconds
 - Circle STARK: 1-3 seconds
 - Soft finality becomes truly "instant"
3. **Instant Settlement with ZK:** Proof exists before funds move
 - Not optimistic (no challenge period)
 - Not trusted (no validator set)
 - Mathematically verified before transfer
4. **Efficient Recursive Batching:** M31 makes aggregation fast
 - 50 proofs aggregated in 2-5 minutes (was 25 min)
 - 90-98% gas savings
 - No user-perceived delay
5. **Liquidity Pool + ZK:** Novel combination
 - Pool enables instant liquidity
 - ZK ensures trustless settlement
 - LPs protected by cryptographic proofs
6. **Chain-Agnostic Design:** Same framework, optimized per chain
 - EVM chains: Groth16 SNARK (native pairing)
 - Solana: Circle STARK (M31 compute-efficient)
 - Future chains: Add appropriate verifier

8.9 Implementation: Proof Generation Code

```
// SNARK Proof Generation (Circom + SnarkJS - Groth16)
async function generateSNARKProof(deposit) {
  const input = {
    amount: BigInt(deposit.amount),
    recipient: poseidonHash(deposit.solanaRecipient),
    blockNumber: deposit.blockNumber,
    nonce: randomBytes(32),
    merkleProof: await getMerkleProof(deposit.txHash)
  };

  const { proof, publicSignals } = await snarkjs.groth16.fullProve(
    input,
    "circuits/deposit.wasm",
    "circuits/deposit_final.zkey"
  );

  return {
    proofId: keccak256(publicSignals[0]),
    proof: packProof(proof), // 192 bytes
    commitment: publicSignals[0]
  };
}

// CIRCLE STARK Proof Generation (Plonky3 / Stwo - M31 Field)
async function generateCircleSTARKProof(transfer) {
  // Using Mersenne-31 field for 7-10x speedup
  const trace = buildM31ExecutionTrace({
    solAmount: transfer.amount,
    recipient: transfer.recipient,
    slot: transfer.slot,
    signature: transfer.signature,
    exchangeRate: transfer.rate
  });

  // Plonky3 M31 prover - native 32-bit operations
  const proof = await plonky3Prover.prove(
    transferCircuit, // AIR constraints over  $F_{2^{31}-1}$ 
    trace,
    { field: "mersenne31" }
  );

  // Proving time: ~1-3 seconds (was 10-30 seconds)
  return {
    proofId: keccak256(proof.commitment),
    proof: proof.serialize(), // ~50KB
    commitment: proof.commitment
  };
}
```


8.9.1 M31 Field Operations (Why It's Fast)

```
// Traditional STARK field operation (256-bit)
function fieldMul_traditional(a, b, p) {
  const product = bignum.multiply(a, b); // Expensive
  return bignum.mod(product, p);         // Division required
}

// Circle STARK M31 field operation (32-bit)
function fieldMul_m31(a, b) {
  const P = 0x7FFFFFFF; // 2^31 - 1
  const product = BigInt(a) * BigInt(b);

  // Reduction without division!
  let result = Number((product & BigInt(P)) + (product >> 31n));
  if (result >= P) result -= P;

  return result; // Single CPU cycle
}

// SIMD vectorized (8 operations at once)
function fieldMul_m31_simd(a_vec, b_vec) {
  // Process 8 field multiplications in parallel
  return _mm256_mul_epu32(a_vec, b_vec); // AVX2 instruction
}
```

8.10 Gas Costs and Optimization

| Operation | Ethereum Gas | Solana CU | Optimization |
|--------------------------|--------------|------------|------------------------|
| SNARK verification | 200,000 | N/A | EVM pairing precompile |
| STARK commitment | 50,000 | 50-100K | M31 hash verification |
| Ghost ID binding | 30,000 | 10K | Single Poseidon hash |
| State update | 40,000 | 5K | Efficient storage |
| Total per payment | 320K | 70K | |

8.10.1 Circle STARK Performance Gains

| Metric | Traditional STARK | Circle STARK (M31) |
|---------------------|--------------------|--------------------|
| Proving time | 10-30 seconds | 1-3 seconds |
| Solana verification | 500K compute units | 50-100K CU |
| Memory for proving | 4-8 GB | 500MB-1GB |
| Batch of 50 proofs | 15-25 minutes | 2-5 minutes |

Why M31 Matters for Ghost Protocol

- **Soft finality is faster:** Proof generation in 1-3 sec means user gets funds faster
- **Batching is practical:** 50 proofs in 2-5 min instead of 25 min
- **Solana verification is cheap:** 5-10x reduction in compute units
- **Memory efficient:** Can run on commodity hardware

Proof Aggregation with M31

Batch multiple payments into a single proof:

- 50 payments → 1 aggregated SNARK (Ethereum)
- 50 Circle STARKs → 1 recursive proof (Solana)
- Gas per payment: 320K → 10K
- M31 makes aggregation 7-10x faster than traditional

9 Pricing

Oracle: Pyth Network (real-time)

$$\text{Output} = \text{Input} \times \frac{\text{Input_USD}}{\text{Output_USD}} \times (1 - \text{fee}) \quad (14)$$

10 Fee Structure

| Path | Fee | Split | Speed |
|-------------|-------------------|------------------------|---------|
| Ghost Pool | 0.3% | 0.1% protocol, 0.2% LP | Fastest |
| DEX Route | 0.3% + DEX fee | Protocol + DEX LPs | Medium |
| Circle USDC | 0.1% + Circle fee | Protocol + Circle | Varies |

11 Risk Management

11.1 Insurance Fund Model

- 0.1% of every transaction to Insurance Fund
- Covers oracle failures, relayer defaults, edge cases
- Balance visible on-chain, DAO-controlled
- Grows with protocol usage

11.2 Path-Specific Risks

| Path | Risk | Mitigation |
|------------|------------------------|--------------------------|
| Ghost Pool | Pool liquidity | Insurance fund |
| DEX Route | Smart contract (multi) | Audited DEXs only |
| Circle | Centralized, freezing | Optional path, disclosed |

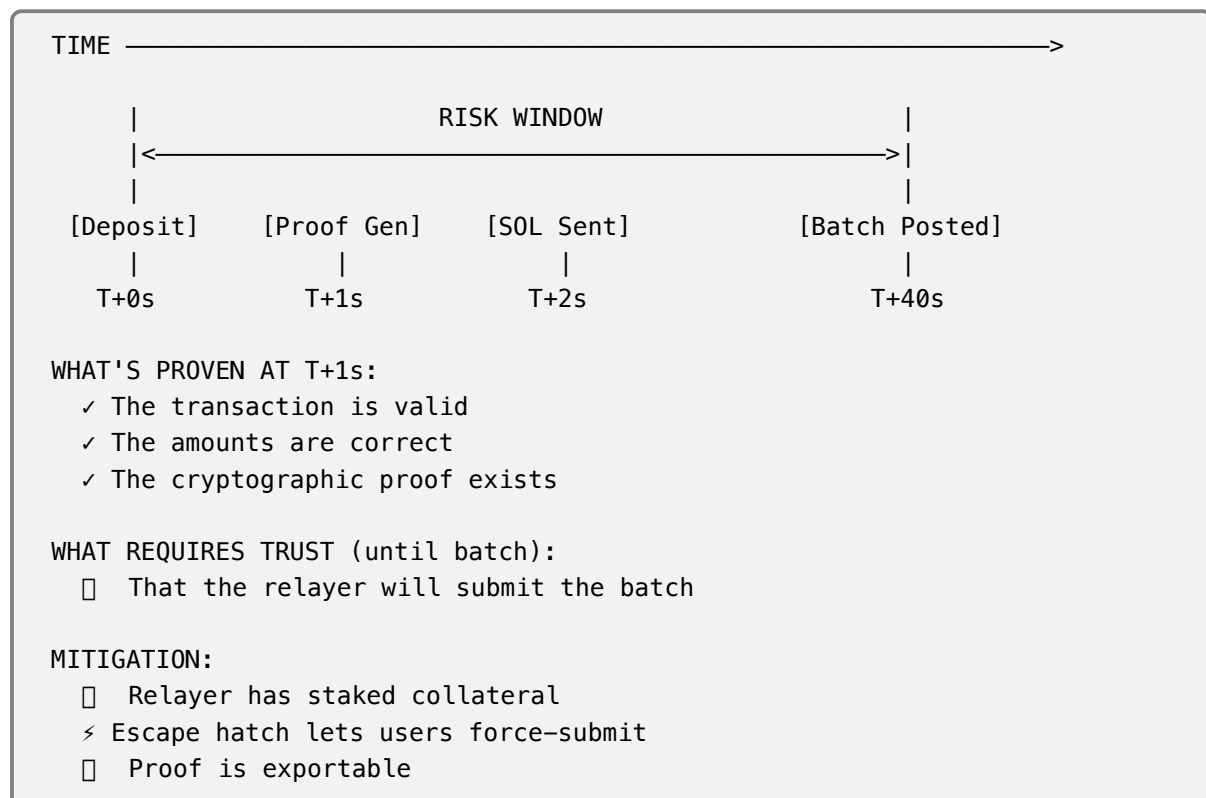
Transparency Principle

Users always see which path their transaction takes. Circle path clearly labeled as "centralized but regulated." DEX path shows which protocols involved.

12 Security Analysis: Risk Windows & Mitigations

The soft finality model creates a **risk window** between instant delivery and on-chain settlement. This section analyzes all attack vectors and their mitigations.

12.1 Risk Window Timeline



12.2 Risk 1: Relayer Disappears After Sending SOL

SCENARIO:

- User deposits ETH ✓
- Relayer generates proof ✓
- Relayer sends SOL to recipient ✓
- Relayer goes offline forever ✗
- Batch never submitted to Ethereum ✗

RESULT:

- Recipient: Has SOL ✓ (happy)
- User: ETH in contract
- Relayer: Lost SOL, never reclaimed ETH

Risk Level: Medium

Who Loses: Relay (they sent SOL but cannot prove it on-chain)

Mitigations:

- Proof is exportable — user or anyone can submit it
- Escape hatch after timeout allows forced settlement
- Relay collateral covers losses
- Multiple relay redundancy

12.3 Risk 2: Ethereum Reorg After Instant Release

High Risk If Not Mitigated

If Ethereum reorganizes and the deposit transaction disappears, the relay has sent SOL for a non-existent deposit.

SCENARIO:

- User deposits ETH (block 1000)
- Relay sees deposit, generates proof
- Relay sends SOL instantly
- Ethereum reorgs, block 1000 replaced
- User's deposit TX no longer exists
- Proof references non-existent state

RESULT:

- Recipient: Has SOL ✓
- User: Got refunded ETH (reorg) ✓
- Relay: Lost SOL, proof is orphaned ✗

Risk Level: High (if not mitigated)

Who Loses: Relay

Mitigation:

```
// Wait for Ethereum finality before releasing SOL
const REQUIRED_CONFIRMATIONS = 12; // ~3 minutes

// Or use finalized block (post-merge Ethereum)
const block = await provider.getBlock('finalized');
```

Safe Instant Model

Wait for 2-3 Ethereum confirmations (30-45 seconds) before releasing SOL. This reduces reorg risk to near-zero while maintaining "near-instant" UX.

12.4 Risk 3: Relay Submits Fake Proof

SCENARIO:

- User deposits ETH
- Relay claims to generate proof (doesn't)
- Relay sends SOL anyway
- Batch time comes...
- Relay has no valid proof to submit
- On-chain verification fails

RESULT:

- Recipient: Has SOL ✓ (can't be clawed back)
- User: ETH stuck or returned?
- Relay: Can't prove the transfer happened

Risk Level: Critical (if not mitigated)

Who Loses: Relay and system integrity

Mitigation:

```
// Users/watchers can verify proof BEFORE trusting
async function verifyProofLocally(proof, publicInputs) {
  const isValid = await snarkjs.groth16.verify(
    vkey, publicInputs, proof
  );
  if (!isValid) throw new Error("INVALID - DO NOT TRUST");
  return true;
}
```

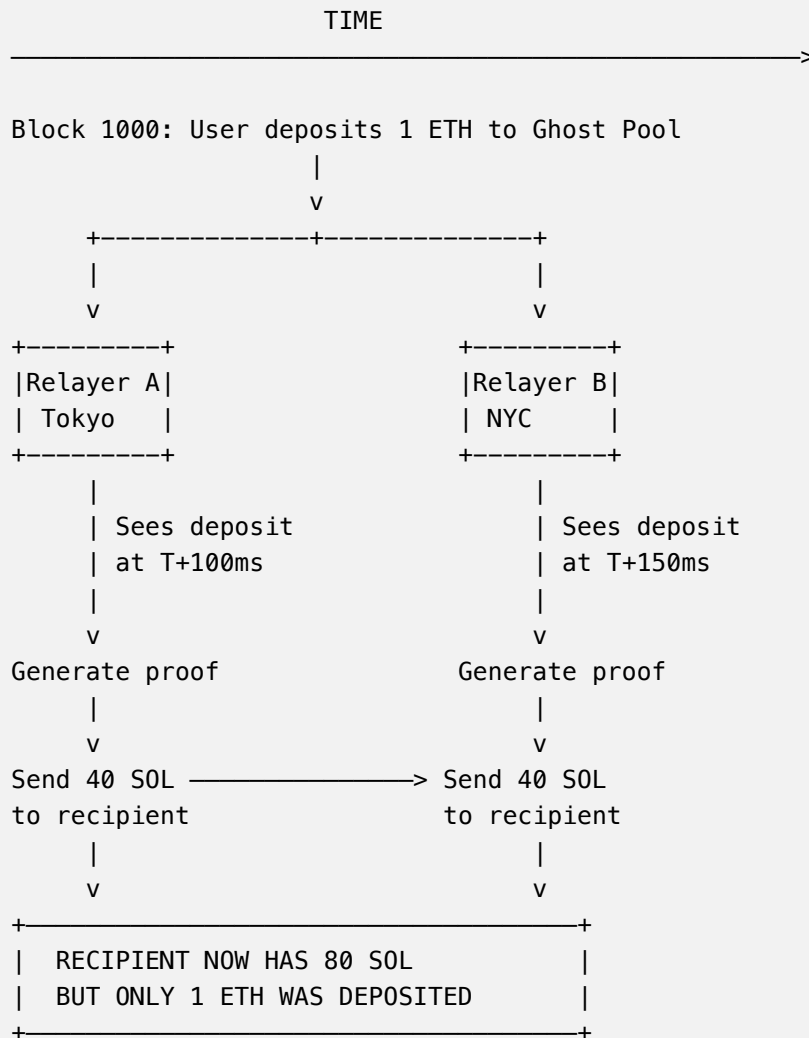
12.5 Risk 4: Double Spend via Race Condition

This is the most critical attack vector. It occurs when multiple relayers process the same deposit.

Critical Risk

If multiple relayers see the same deposit and all send SOL, the recipient gets multiple payouts for a single deposit.

THE ATTACK SCENARIO:



12.5.1 Attack Variations

Variation 1: Multiple Honest Relayers (Accident)

In a decentralized relayer network, all relayers see the same deposit event simultaneously. Each tries to help, resulting in multiple payouts.

Variation 2: Malicious User Exploits Latency

```
USER (Attacker):
├─ Deposit 1 ETH
├─ Immediately send "process" request to 10 relayers
├─ Each relayer thinks they're first
├─ Collect 10x SOL
└─ Profit: 9 ETH worth of SOL
```

Variation 3: MEV / Front-Running

BLOCK N:

- |— User's deposit TX in mempool
- |— MEV bot sees it
- |— MEV bot spins up 5 "relayer" instances
- |— Each instance fires SOL the moment deposit confirms
- |— All 5 fire simultaneously

12.5.2 Solutions**Solution 1: Single Relayer (Centralized)**

Only ONE authorized relayer processes deposits.

| Pros | Cons |
|--------------------------|-------------------------|
| No double-spend possible | Single point of failure |
| Simple implementation | Not decentralized |
| | Trust assumption |

Verdict: Works for MVP but defeats decentralization goals.

Solution 2: On-Chain Lock Before Release

```
contract GhostPool {
    mapping(bytes32 => address) public depositClaims;

    // Step 1: Relayer claims the deposit on-chain FIRST
    function claimDeposit(bytes32 depositHash) external {
        require(depositClaims[depositHash] == address(0),
            "Already claimed");
        depositClaims[depositHash] = msg.sender;
        emit DepositClaimed(depositHash, msg.sender);
    }

    // Step 2: Only the claimer can process it
    function executeDeposit(
        bytes32 depositHash,
        bytes calldata proof
    ) external {
        require(depositClaims[depositHash] == msg.sender,
            "Not your claim");
        // Process...
    }
}
```

FLOW:

1. Deposit happens
2. Relayer sends TX: claimDeposit(hash)
3. Wait for claim TX to confirm (12 sec)
4. Only winner can process
5. Winner sends SOL
6. Winner submits proof in batch

TRADE-OFF: Adds 12+ seconds latency

Verdict: Safe but slower.

Solution 3: Deterministic Leader Election

RELAYER CONSENSUS PROTOCOL:

1. Deposit event detected by all relayers
2. Relayers run leader election:
 - Hash(depositHash + relayerPubkey + blockHash)
 - Lowest hash = leader
3. Leader has 10 seconds to process
4. If leader fails, next-lowest takes over
5. Other relayers WATCH, don't act

```
// Deterministic leader election
function electLeader(depositHash, blockHash, relayers) {
  const scores = relayers.map(r => ({
    relayer: r,
    score: keccak256(depositHash + r.pubkey + blockHash)
  }));

  scores.sort((a, b) => a.score.localeCompare(b.score));
  return scores[0].relayer; // Deterministic winner
}

// All relayers compute the same leader
const leader = electLeader(deposit.hash, block.hash, relayers);

if (leader.pubkey === myPubkey) {
  await processDeposit(deposit); // I'm the leader
} else {
  await watchForCompletion(deposit, leader); // Just watch
}
```

Verdict: Best balance of decentralization and safety.

Solution 4: On-Chain Nonce (Recommended)

```
contract GhostPool {
  mapping(bytes32 => bool) public processedDeposits;

  function processDeposit(
    bytes32 depositHash,
    bytes calldata proof
  ) external onlyRelayer {
    // THE KEY LINE
    require(!processedDeposits[depositHash],
      "Already processed");
    processedDeposits[depositHash] = true;

    require(verifyProof(proof), "Invalid proof");
    emit DepositProcessed(depositHash, recipient, solAmount);
  }
}
```


HOW IT WORKS:

Relayer A sends processDeposit TX —> ✓ Succeeds (first)
 Relayer B sends processDeposit TX —> ✗ Reverts (nonce set)
 Relayer C sends processDeposit TX —> ✗ Reverts (nonce set)

Relayer A sends SOL —> ✓ Valid
 Relayer B sees A succeeded —> STOP Aborts SOL send
 Relayer C sees A succeeded —> STOP Aborts SOL send

The race condition is moved ON-CHAIN where Ethereum's consensus resolves it. Whoever's TX gets included first wins.

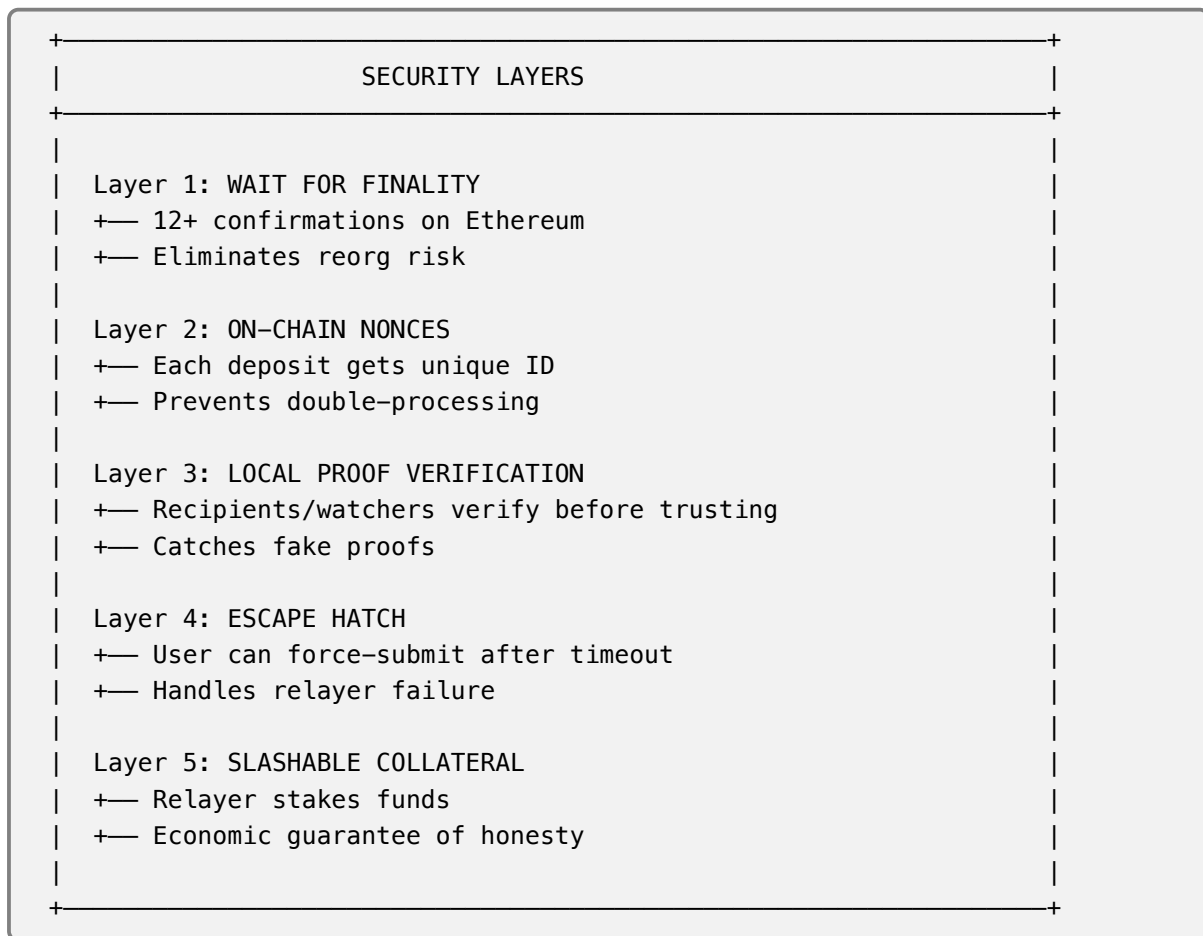
Verdict: Simple, battle-tested pattern used by MEV bots and liquidation systems.

12.5.3 Double-Spend Solution Summary

| Approach | Safe? | Latency | Decentralized? |
|----------------------|-------|------------|----------------|
| Single relayer | Yes | 0 sec | No |
| On-chain claim first | Yes | +12-15 sec | Yes |
| Leader election | Yes | +1-2 sec | Yes |
| On-chain nonce | Yes | 0 sec* | Yes |

*Nonce requires claim TX before SOL send, but adds no user-perceived latency.

12.6 Defense in Depth



12.7 Complete Risk Matrix

| Risk | Prob. | Impact | Who Loses | Mitigation |
|-----------------|----------|----------|-----------|--------------------------|
| Relayer offline | Low | Medium | Relayer | Escape hatch, collateral |
| Ethereum reorg | Low | High | Relayer | Wait for finality |
| Fake proof | Low* | Critical | Everyone | Local verification |
| Double spend | Medium | Critical | System | On-chain nonces |
| Solana failure | Low | Medium | User | Retry logic |
| DA layer down | Very Low | Low | History | Multi-DA redundancy |

*Low if relayer is honest; higher if adversarial.

12.8 Recommended Production Configuration

Safe Instant Model

1. Wait for 2-3 Ethereum confirmations (30-45 sec)
 - Reduces reorg risk to near-zero
2. Generate ZK proof
 - Cryptographic validity established
3. Publish proof hash publicly (IPFS/API)
 - Anyone can verify before trusting
4. Send SOL to recipient
 - Recipient can verify proof first if paranoid
5. Batch proofs to chain within 5 minutes
 - Short window = minimal risk exposure

Result: "Near-instant" (45 sec) with minimal risk.

Phased Decentralization

MVP: Single trusted relayer with on-chain nonce safety net. Fast, simple.

Production: Primary relayer cluster with backup set using leader election.

Mature: Open relayer network with consensus protocol, slashing, and reputation.

13 Business Model: Three Strategic Paths

Ghost Protocol can operate with three distinct liquidity strategies. Each can work standalone or in combination.

13.1 Path 1: Native Ghost Pools (Decentralized)

Ghost Pool Model

Build and maintain proprietary liquidity pools on each supported chain.

13.1.1 How It Works

1. LPs deposit native assets (ETH, SOL, etc.) into Ghost pools
2. Users pay into source pool, receive from destination pool
3. ZK proofs verify each transaction
4. Fees distributed to LPs proportionally

13.1.2 Revenue Model

| Fee Type | Rate | Recipient | Purpose |
|-----------------|-------|---------------------|-------------------|
| Transaction Fee | 0.30% | Split | Total fee charged |
| → Protocol | 0.10% | Ghost Treasury | Operations, dev |
| → LP Rewards | 0.20% | Liquidity Providers | Yield for LPs |
| Insurance Fund | 0.02% | Reserve | Risk coverage |

13.1.3 Unit Economics

MONTHLY VOLUME: \$10M

Revenue Breakdown:

| | | |
|---------------------------|----------|------------------|
| Transaction fees (0.30%): | \$30,000 | |
| - Protocol share (0.10%): | \$10,000 | <- Ghost revenue |
| - LP rewards (0.20%): | \$20,000 | <- To depositors |
| - Insurance (0.02%): | \$2,000 | <- Reserve |

LP Returns (assuming \$2M TVL):

Annual yield: $(\$20,000 \times 12) / \$2M = 12\%$ APY

Break-even Analysis:

Minimum monthly volume for sustainability: ~\$3M

13.1.4 Pros and Cons

| Advantages | Challenges |
|-----------------------------|-------------------------------|
| Full control over liquidity | Requires active management |
| All fees stay in ecosystem | Multi-chain coordination |
| Truly decentralized | Pool rebalancing needed |
| Scales to any trade size | |
| Best UX (fastest) | Requires active LP management |

Best For

Any trade size with sufficient pool depth, maximum speed and control, keeping all revenue in-protocol.

13.2 Path 2: DEX Aggregation (Leverage Existing Liquidity)

DEX Router Model

Route through existing DEX liquidity (Uniswap, Jupiter, etc.) instead of maintaining own pools.

13.2.1 How It Works

1. User initiates cross-chain payment

2. Ghost bridges wrapped asset to destination chain
3. Jupiter/Uniswap swaps to native asset
4. Recipient receives desired token

13.2.2 Revenue Model

| Fee Type | Rate | Recipient | Notes |
|------------------------|-------------------|-----------------|------------------|
| Routing Fee | 0.05-0.10% | Ghost Protocol | Our cut |
| DEX Swap Fee | 0.30% | DEX LPs | Uniswap/Jupiter |
| Bridge Fee | 0.10% | Bridge protocol | Wormhole/etc |
| Total User Cost | 0.45-0.50% | Various | Higher than Pool |

13.2.3 Unit Economics

MONTHLY VOLUME: \$10M

Revenue (Ghost keeps routing fee only):

Routing fee (0.08%): \$8,000 ← Ghost revenue

Comparison to Pool Model:

Pool model revenue: \$10,000

DEX model revenue: \$8,000

Difference: -20% revenue

BUT: No capital requirements!

Pool model needs \$2M+ TVL

DEX model needs \$0 TVL

Capital Efficiency:

Pool: \$10K revenue / \$2M capital = 0.5% monthly return on capital

DEX: \$8K revenue / \$0 capital = infinite return on capital

13.2.4 Pros and Cons

| Advantages | Challenges |
|--------------------------|---------------------------------|
| No capital requirements | Lower margins |
| Infinite liquidity depth | Slower (30-120 sec) |
| Proven DEX security | Dependent on external protocols |
| Easy to launch | Variable slippage |
| Handles large trades | Multiple failure points |

Best For

Large trades (\$50K+), capital-light launch, maximum liquidity depth, price-sensitive users.

13.3 Path 3: Circle Partnership (USDC Settlement)

Circle CCTP Model

Partner with Circle to use USDC as settlement layer with mint/burn capabilities.

13.3.1 How It Works

1. User pays in any asset
2. Ghost converts to USDC (via DEX if needed)
3. Circle CCTP burns USDC on source chain
4. Circle mints USDC on destination chain
5. Ghost converts USDC to recipient's desired asset

13.3.2 Revenue Model

| Fee Type | Rate | Recipient | Notes |
|--------------------|-------|----------------|------------------|
| Conversion Fee | 0.10% | Ghost Protocol | In/out of USDC |
| Circle CCTP | 0.00% | Circle | Currently free |
| Swap fees (if any) | 0.30% | DEX LPs | Only if not USDC |

13.3.3 Unit Economics

MONTHLY VOLUME: \$10M (assume 50% already USDC)

USDC-to-USDC transfers (\$5M):

| | | |
|-----------------------|---------|------------------------|
| Conversion fee: | \$0 | (no conversion needed) |
| Protocol fee (0.05%): | \$2,500 | <- Ghost revenue |

Non-USDC transfers (\$5M):

| | | |
|-------------------------|----------|--------------------|
| Conversion fee (0.10%): | \$5,000 | <- Ghost revenue |
| DEX fees: | \$15,000 | <- To external LPs |

Total Ghost Revenue: \$7,500

Advantage: Institutional trust

- Circle is regulated (NYDFS, etc.)
- Banks can participate
- Compliance-friendly

13.3.4 Pros and Cons

Centralization Trade-off

Circle can freeze USDC addresses. This path trades decentralization for institutional access and regulatory clarity.

| Advantages | Challenges |
|-----------------------|------------------------------------|
| No liquidity needed | Centralized (Circle controls) |
| Institutional trust | USDC can be frozen |
| Regulatory compliance | Requires partnership |
| Unlimited scale | Limited to Circle-supported chains |
| Stablecoin focus | Extra swap for non-USDC |

Best For

Institutional clients, regulated environments, stablecoin-heavy use cases, enterprise integrations.

13.4 Path 4: Hybrid Model (Recommended)

Smart Routing Hybrid

Combine all three paths with intelligent routing based on trade characteristics.

13.4.1 Routing Logic

SMART ROUTER DECISION TREE:

Input: trade_amount, speed_preference, user_type

```
if (trade_amount < $10K AND speed_preference == "instant"):
    -> GHOST POOL (fastest, 10-30 sec)
```

```
elif (trade_amount > $50K):
    -> DEX ROUTE (deepest liquidity)
```

```
elif (user_type == "institutional" OR compliance_required):
    -> CIRCLE USDC (regulated path)
```

```
elif (asset == USDC AND destination_has_CCTP):
    -> CIRCLE USDC (native, no conversion)
```

```
else:
    -> Compare Pool vs DEX, pick best rate
```

User can always override with manual path selection.

13.4.2 Revenue Optimization

| Trade Type | Path | Ghost Fee | Speed | Why |
|---------------------|--------|-----------|--------|--------------|
| \$500 ETH→SOL | Pool | 0.10% | 15 sec | Fast, simple |
| \$100K ETH→SOL | DEX | 0.08% | 90 sec | Depth needed |
| \$50K USDC→USDC | Circle | 0.05% | 60 sec | Native path |
| \$25K institutional | Circle | 0.10% | 60 sec | Compliance |

13.4.3 Hybrid Unit Economics

MONTHLY VOLUME: \$10M (distributed across paths)

Volume Distribution (optimized):

Ghost Pool (40%): \$4M @ 0.10% = \$4,000

DEX Route (35%): \$3.5M @ 0.08% = \$2,800

Circle USDC (25%): \$2.5M @ 0.07% = \$1,750

Total Ghost Revenue: \$8,550/month

Compared to single-path:

Pool-only: \$10,000 (but needs \$2M+ capital)

DEX-only: \$8,000 (no capital needed)

Circle-only: \$7,500 (needs partnership)

HYBRID: \$8,550 (balanced, resilient)

Key Advantage: Resilience

- Pool drained? Fall back to DEX
- DEX congested? Use Pool or Circle
- Circle issues? Decentralized paths available

13.5 Strategic Recommendation

Phased Approach

Phase 1 (Launch): Ghost Pool only

- Simplest to implement
- Full control
- Bootstrap with protocol-owned liquidity

Phase 2 (Scale): Add DEX routing

- Handle overflow volume
- Large trade support
- No additional capital needed

Phase 3 (Scale): Deep liquidity pools + smart routing

- Scale native pools to handle any trade size
- Smart routing to DEX when optimal
- Multi-chain pool coordination

Phase 4 (Optional): Circle partnership

- For institutional clients requiring regulated path
- Stablecoin optimization for USDC flows
- Not required—fully optional addition

Phase 4 (Mature): Full hybrid with smart routing

- Automatic path optimization
- Maximum resilience
- Best user experience

13.6 Standalone Path Viability

Each path can work independently:

| Path | Viable Alone? | Min Volume | Capital Needed |
|-------------|---------------|-------------|----------------|
| Ghost Pool | Yes | \$3M/month | \$1-5M TVL |
| DEX Route | Yes | \$5M/month | \$0 |
| Circle USDC | Yes | \$10M/month | Partnership |
| Hybrid | Best | \$2M/month | Flexible |

14 LP Economics Deep Dive

14.1 Revenue for Ghost Pool LPs

- 0.2% of Ghost Pool transactions
- Priority for small/fast trades
- Liquidity mining rewards (optional)
- Auto-compounding fees

14.2 LP Yield Scenarios

YIELD BY VOLUME (assuming \$2M TVL):

| Monthly Volume | LP Fees (0.2%) | Annual APY |
|----------------|----------------|------------|
| \$5M | \$10,000 | 6.0% |
| \$10M | \$20,000 | 12.0% |
| \$20M | \$40,000 | 24.0% |
| \$50M | \$100,000 | 60.0% |

Comparison to DeFi yields:

| | |
|--------------|------------------------------|
| Aave USDC: | 3–5% APY |
| Uniswap ETH: | 5–15% APY |
| Ghost Pool: | 6–60% APY (volume dependent) |

14.3 TradFi Integration Path

INSTITUTIONAL ONBOARDING:

- CUSTODY SETUP**
Bank Treasury → Qualified Custodian (Fireblocks/Anchorage)
- LIQUIDITY DEPLOYMENT**
Option A: Ghost LP Pool (earn 6–24% yield)
Option B: Circle Partnership (regulatory comfort)
Option C: Both (diversified exposure)
- COMPLIANCE LAYER**
 - KYC/AML on large deposits
 - Jurisdiction restrictions
 - Audit trail via ZK proofs
- REPORTING**
 - Real-time dashboard
 - Monthly statements
 - Tax documentation

14.4 LP Tiers

| Tier | Minimum | Fee Share | Benefits |
|---------------|----------|-----------|---------------------------|
| Retail | 0.1 ETH | 0.20% | Standard access |
| Professional | 10 ETH | 0.22% | Governance voting |
| Institutional | 100 ETH | 0.25% | Priority support, API |
| Strategic | 1000 ETH | 0.30% | Revenue share, board seat |

15 Competitive Analysis

| Protocol | Speed | Trust | Liquidity | Native |
|--------------|----------|--------------|--------------|-----------|
| Ghost | 10-30s | Trustless | Multi-source | Yes |
| Wormhole | 15-20s | 19 guardians | Own pools | No |
| LayerZero | 1-5min | Oracle | Partner | No |
| Across | Instant* | 7-day | Own pools | Yes |
| Circle CCTP | Minutes | Circle | Mint/burn | USDC only |

15.1 Ghost Advantages

1. **Multi-source liquidity:** Not limited to own pools
2. **Native ZK proofs:** Trustless, not optimistic
3. **Flexible paths:** Trustless or regulated (user choice)
4. **Chain-agnostic:** Add chains with adapters

16 Circle Partnership Path

16.1 How to Partner with Circle

1. **Apply:** Circle Partner Program application
2. **Compliance:** KYC/AML program, legal review
3. **Technical:** API integration, security audit
4. **Business:** Volume commitments, fee structure
5. **Launch:** Staged rollout with monitoring

16.2 What Circle Partnership Enables

- Native USDC on 15+ chains (no wrapping)
- Cross-Chain Transfer Protocol (CCTP)
- Fiat on/off ramps for institutional clients
- Regulatory cover for compliant path
- Marketing co-promotion

Circle is Not Trustless

Circle partnership adds a **regulated, centralized** option. Users who want fully trustless can use Ghost Pools or DEX routes. Transparency about trade-offs is key.

17 Running the System

```
# Relay (with DEX routing)
node scripts/instant-relayer.mjs

# Dashboard
```

```
cd dashboard && npm run dev

# Deploy
npx hardhat compile
node scripts/deploy-pools.mjs --seed
```

18 Roadmap

1. Mainnet launch (ETH + SOL)
2. Scale native pools (target: \$10M+ TVL per chain)
3. DEX integration (Uniswap, Jupiter) for smart routing
4. Bidirectional flows (SOL to ETH)
5. Multi-asset support (USDC, USDT, wBTC)
6. Bitcoin integration
7. zkVM migration (Risc0/SP1) for unified architecture
8. Circle partnership (optional, for institutional clients)

19 Summary

Ghost Protocol: Architecture Summary

Core Design Decisions:

- **Validium Model:** Off-chain data, on-chain state roots (99% cost reduction)
- **Recursive Batching:** 10-50 proofs aggregated (90-98% gas savings)
- **Soft Finality:** Proof exists before funds move (NOT optimistic)

Liquidity Sources:

- **Ghost Pools:** Fast, trustless, for small trades
- **DEX Routing:** Deep liquidity, best prices for large trades
- **Circle (optional):** Regulated path, institutional stablecoins

Security Model:

- **On-chain nonces:** Prevent double-spend attacks
- **Confirmation waiting:** Eliminate reorg risk
- **Escape hatch:** Users can force-submit if relayer fails
- **Slashable collateral:** Economic guarantees

Honest Claims:

- "Instant ZK" = Proof exists before funds move, batched later
- User-perceived: 30-45 seconds
- Cryptographic finality: 2-5 minutes
- This is NOT optimistic (proof is generated, not assumed)

20 Technical Critique & FAQ

This section addresses common questions and critiques from security researchers and engineers reviewing the Ghost Protocol architecture.

20.1 Novelty Assessment

| Component | Novelty Level | Rationale |
|------------------------|---------------|--------------------------------------|
| SNARK+STARK Hybrid | High | First chain-specific ZK optimization |
| Ghost ID Binding | High | Novel cross-primitive commitment |
| Liquidity Meta-Routing | Medium | Trust-model routing, not just price |
| Instant Settlement | Medium | UX instant, crypto finality delayed |
| Pool + DEX + Circle | Medium | Solves cold-start problem |

20.2 Why SNARK for Ethereum, STARK for Solana?

Chain-Specific Optimization

Most ZK bridges force one proof system everywhere. Ghost Protocol treats chains **asymmetrically** based on their constraints:

| Chain | Constraint | Our Solution |
|----------|------------------------------------|--------------------------------------|
| Ethereum | Gas-constrained | Groth16 SNARK (192 bytes, 200K gas) |
| Solana | Compute-capable, storage-expensive | STARK (no trusted setup, hash-based) |

The Innovation:

- Ethereum verification must be cheap → SNARKs have $O(1)$ verification
- Solana can handle hashing throughput → STARKs leverage this
- No single "Trusted Setup Ceremony" controls both chains
- Each chain uses its optimal proof system

20.3 Q&A: Hard Questions

20.3.1 Q: Is 10-30 Second Settlement Actually Possible?

Honest Answer

User-perceived: Yes, 10-30 seconds.

Cryptographic finality: No, 2-5 minutes.

WHAT "INSTANT" ACTUALLY MEANS:**User Timeline:**

- 0 sec – User pays ETH
- 12 sec – Relay detects (1 Ethereum block)
- 25 sec – User receives SOL
- [USER IS DONE – "INSTANT" FROM THEIR POV]

Background Settlement:

- +30 sec – SNARK proof generated
- +60 sec – STARK proof generated
- +90 sec – Proofs submitted to contracts
- +120 sec – On-chain verification
- [CRYPTOGRAPHIC FINALITY ACHIEVED]

Analogy: Credit cards

- You get coffee immediately
- Actual settlement takes 2–3 days
- Ghost: User gets SOL immediately
- ZK settlement takes 2–5 minutes

20.3.2 Q: What About Ethereum Re-orgs?**Valid Concern**

If Ethereum re-orgs after the relay sends SOL, the source transaction disappears. Who loses money?

Answer: The relay, not the user.

- Relay waits for 2-3 block confirmations (not true finality)
- Relay accepts re-org risk in exchange for speed
- Insurance fund covers catastrophic re-orgs
- User experience is protected

Risk mitigation:

1. Conservative block confirmation (3+ blocks for large amounts)
2. Dynamic confirmation based on transaction size
3. Insurance fund sized to cover 99.9% of re-org scenarios

20.3.3 Q: Unit Economics Don't Work for Small Transactions?**The Hard Truth**

Correct. Per-transaction ZK proofs on Ethereum Mainnet are not economically viable for retail-sized transactions.

MAINNET COST ANALYSIS:

Transaction: \$50 ETH → SOL

Fee revenue (0.3%): \$0.15

SNARK verification (200K gas):

@ 20 gwei: \$4.00

@ 50 gwei: \$10.00

RESULT: Significant loss on small txs

BREAK-EVEN ANALYSIS:

@ 20 gwei: Transaction must be > \$1,300

@ 50 gwei: Transaction must be > \$3,300

SOLUTIONS:

1. Target L2s (Arbitrum, Base, Optimism)
 - Gas is 10-100x cheaper
 - \$50 tx becomes profitable
2. Proof Aggregation (batch mode)
 - 100 txs in 1 proof
 - Gas per tx: \$10 → \$0.10
 - Trade-off: 5-10 min batching delay
3. High-value focus (institutional)
 - \$10K+ transactions
 - \$30 fee is 0.3% - acceptable
4. Hybrid: Instant for users, batch proofs
 - User gets SOL in 30 sec
 - Proof submitted in batch later
 - Best of both worlds

Recommended Deployment Strategy

- **Phase 1:** L2 ↔ Solana (cheap gas, per-tx proofs work)
- **Phase 2:** Mainnet with proof aggregation (batched)
- **Phase 3:** Mainnet per-tx for high-value (>\$5K)

20.3.4 Q: What If Circle Blacklists Ghost Protocol?

Risk: Circle can freeze USDC addresses. If they blacklist Ghost contracts, the Circle path fails.

Mitigation:

1. Circle path is **optional**, not required
2. Traffic automatically routes to Pool or DEX
3. No user funds ever held in Circle's custody
4. Disclosed as centralization trade-off

Residual risk: If 25% of volume relies on Circle and they act adversarially, that volume is lost. This is accepted in exchange for institutional access.

20.3.5 Q: How Is the Ghost ID Cryptographically Secure?

The `ghost_id` prevents double-spending across two different cryptographic primitives:

$$\text{ghost_id} = \text{Poseidon}(\text{snark_commitment} || \text{stark_commitment} || \text{nonce}) \quad (15)$$

Security properties:

- **Collision resistance:** 2^{-256} probability of collision (Poseidon)
- **Binding:** Changing either commitment changes the `ghost_id`
- **Uniqueness:** Each payment has a unique nonce
- **Atomicity:** Both proofs must reference the same `ghost_id`

ATTACK SCENARIO: Double-Spend Attempt

Attacker tries to:

1. Create valid SNARK for deposit X
2. Create two STARKs for transfer Y and transfer Z
3. Claim both Y and Z on Solana

Why it fails:

- SNARK commits to: (amount, recipient, block, nonce)
- STARK commits to: (sol_amount, recipient, slot, signature)
- `ghost_id = H(snark_commit || stark_commit)`

If attacker changes STARK (different recipient):

- > `stark_commitment` changes
- > `ghost_id` changes
- > Does not match original SNARK's `ghost_id`
- > Verification fails

Result: Each deposit can only claim ONE transfer.

20.4 Comparison: Ghost vs. Existing Bridges

| Protocol | Proof | Speed | Trust | Liquidity | Cost |
|--------------|------------------|----------|-----------------|-----------|--------|
| Ghost | SNARK+STARK | 30s UX | Trustless | Hybrid | Medium |
| Wormhole | None (multi-sig) | 15s | 19 guardians | Own pools | Low |
| LayerZero | None (oracle) | 1-5min | Oracle+Relayer | Partner | Low |
| Across | Optimistic | Instant* | 7-day challenge | Own pools | Low |
| zkBridge | SNARK only | Minutes | Trustless | Limited | High |
| Succinct | Light client | Minutes | Trustless | None | High |

*Across is "instant" but optimistic—funds can be clawed back during challenge period.

20.5 Strategic Positioning

Where Ghost Protocol Wins

1. **Any-size instant transfers**
With deep liquidity pools + recursive batching, we handle \$50 to \$50M transfers profitably
2. **Speed-critical payments**
30-45 second soft finality beats all competitors on user experience
3. **L2-to-Solana corridor**
Arbitrum/Base/Optimism to Solana with cheap per-tx proofs
4. **Compliance-sensitive flows**
Circle path for regulated entities needing audit trails
5. **Cold-start scenarios**
New chains can launch with DEX fallback, no TVL bootstrap needed

Where Ghost Protocol Struggles

1. **Very small retail transactions**
Sub-\$10 transfers may have thin margins even with batching (but still work)
2. **Speed-critical arbitrage**
MEV bots need sub-second, not 30 seconds
3. **Chains without STARK verifiers**
Need native STARK support or fallback to SNARK-only

20.6 Conclusion: Is Ghost Protocol Novel?

Verdict

Yes, with specific distinction.

- **High Novelty:** The SNARK+STARK hybrid architecture optimized per-chain is genuinely new. No production bridge uses this approach.
- **Medium Novelty:** Trust-model routing (Pool/DEX/Circle) is a smart combination of existing primitives.
- **Honest Limitation:** "Instant ZK" is UX-instant, not crypto-instant. This is acceptable but should be clearly communicated.
- **Economic Reality:** Per-tx proofs work on L2s and for high-value. Mainnet retail requires batching.

Ghost Protocol is not reinventing bridging. It is **optimizing bridging** by matching proof systems to chain constraints and routing to trust models.

21 Appendix: Quick Reference

21.1 Terminology

| Term | Definition |
|--------------------|---|
| Soft Finality | User receives funds before proof is on-chain |
| Hard Finality | Proof is verified and recorded on-chain |
| Validium | Off-chain data, on-chain state roots |
| Recursive Batching | Combining multiple proofs into one |
| Ghost ID | Cryptographic binding of SNARK + STARK proofs |
| Escape Hatch | User ability to force-submit if relayer fails |

21.2 Key Metrics

| Metric | Value |
|---------------------------|-------------------------------|
| User-perceived settlement | 30-45 seconds |
| Cryptographic finality | 2-5 minutes |
| SNARK proof size | 192 bytes |
| Circle STARK proof size | 50 KB |
| SNARK proving time | 5-10 seconds |
| Circle STARK proving time | 1-3 seconds (M31) |
| Verification gas (SNARK) | 200,000 |
| Verification CU (STARK) | 50-100K (M31) |
| STARK field | Mersenne-31 ($2^{31} - 1$) |
| Batching savings | 90-98% |
| Fee structure | 0.3% (0.1% protocol, 0.2% LP) |

21.3 Architecture Decision Record

| Decision | Choice | Alternative | Rationale |
|--------------|--------------------|-------------------|-----------------------|
| Data storage | Validium | Rollup | 99% cost reduction |
| EVM proof | Groth16 SNARK | STARK | 200K gas vs 2M gas |
| Solana proof | Circle STARK (M31) | Traditional STARK | 7-10x faster proving |
| STARK field | Mersenne-31 | Goldilocks/BN254 | Native 32-bit CPU ops |
| Settlement | Soft finality | Wait for proof | Better UX |
| Batching | Recursive | Per-tx | Gas efficiency |
| Double-spend | On-chain nonce | Single relayer | Decentralized |

| Aspect | Traditional STARK | Circle STARK (M31) |
|----------------|--------------------|-------------------------|
| Field | 252-256 bit prime | $2^{31} - 1$ (Mersenne) |
| Arithmetic | Bignum (slow) | Native 32-bit (fast) |
| Proving | 10-30 seconds | 1-3 seconds |
| Verification | 500K compute units | 50-100K compute units |
| Memory | 4-8 GB | 500MB-1GB |
| SIMD support | Limited | Excellent (8-wide AVX2) |
| Implementation | Cairo/Stone | Plonky3/Stwo |

21.4 Circle STARK vs Traditional STARK

Why Ghost Protocol Uses Circle STARKs

The Mersenne-31 field is not just an optimization—it's what makes "instant ZK" practically achievable. With 1-3 second proof generation:

- Soft finality feels truly instant
- Batching doesn't create noticeable delays
- Commodity hardware can run provers
- The economic model becomes viable

22 Future: zkVM Integration (Risc0/SP1)

This section outlines the planned evolution of Ghost Protocol's proving infrastructure using zero-knowledge virtual machines.

22.1 The Problem: STARK vs SNARK Trade-off

| Property | STARK | SNARK |
|------------------|----------------------------|------------------------|
| Proving speed | Fast (1-3 sec with M31) | Slower (5-10 sec) |
| Proof size | Large (50 KB) | Tiny (192 bytes) |
| EVM verification | Expensive (2M gas, \$50+) | Cheap (200K gas, \$5) |
| Trusted setup | None | Required |

The dilemma: We want fast STARK proving but cheap EVM verification.

22.2 The Solution: SNARK Wrapper (Recursive Proofs)

What is a Wrapper?

A SNARK "wrapper" is a SNARK proof that says: *"I verified this STARK proof and it is valid."*

The SNARK wraps around the STARK like an executive summary wraps around a detailed report. Ethereum only needs to verify the small summary, not the full report.

RECURSIVE PROOF FLOW:

Your Logic: "0.01 ETH deposited, send 0.4 SOL"

|
v

```
+-----+
| STEP 1: Generate STARK Proof |
| - Uses fast M31 field       |
| - Time: 1-3 seconds         |
| - Output: 50KB proof        |
| - Valid but EXPENSIVE to verify on Ethereum |
+-----+
```

|
v

```
+-----+
| STEP 2: Generate SNARK Wrapper |
| - SNARK proves: "I verified the STARK, it's valid" |
| - Time: 2-4 seconds           |
| - Output: 192 bytes           |
| - This is what goes on Ethereum |
+-----+
```

|
v

```
+-----+
| STEP 3: Verify on Ethereum |
| - Ethereum only sees the tiny SNARK |
| - Cost: 200,000 gas         |
| - Trusts SNARK, which verified the STARK |
+-----+
```

TOTAL: 4-8 seconds proving, 200K gas verification
BEST OF BOTH WORLDS!

22.3 What Are zkVMs?

A **zkVM (Zero-Knowledge Virtual Machine)** automates the entire STARK + SNARK wrapper process:

TRADITIONAL ZK DEVELOPMENT (Without zkVM):

```
+-----+
| You must:                                     |
| - Learn circuit languages (Circom, Cairo)    |
| - Write STARK circuit manually               |
| - Write SNARK wrapper circuit manually       |
| - Handle serialization between provers       |
| - Debug constraint systems                   |
|                                              |
| Time: 8-12 weeks                             |
| Difficulty: Requires ZK expertise            |
+-----+
```

zkVM DEVELOPMENT (With Risc0/SP1):

```
+-----+
| You just:                                    |
| - Write normal Rust code                    |
| - The zkVM handles everything else           |
|                                              |
| Time: 2-3 weeks                             |
| Difficulty: Normal programming              |
+-----+
```

22.4 Risc0 and SP1

Two production-ready zkVMs that Ghost Protocol may integrate with:

| Feature | Risc0 | SP1 (Succinct) |
|------------------|--------------------|--------------------------|
| Language | Rust | Rust |
| Architecture | RISC-V | RISC-V |
| License | Apache 2.0 | MIT / Apache 2.0 |
| Funding | \$40M+ | \$55M+ |
| SNARK wrapper | Built-in (Groth16) | Built-in (Groth16/Plonk) |
| Cloud proving | Yes (Bonsai) | Yes |
| EVM verification | 200K gas | 250K gas |

22.4.1 Example: Ghost Protocol Proof in SP1

```
// THE ENTIRE ZK PROOF LOGIC IN RUST

#![no_main]
sp1_zkvm::entrypoint!(main);

pub fn main() {
    // Read inputs
    let deposit_tx_hash: [u8; 32] = sp1_zkvm::io::read();
    let eth_amount: u64 = sp1_zkvm::io::read();
    let sol_recipient: [u8; 32] = sp1_zkvm::io::read();
    let merkle_proof: Vec<[u8; 32]> = sp1_zkvm::io::read();
    let state_root: [u8; 32] = sp1_zkvm::io::read();

    // Verify deposit exists in Ethereum state
    let computed_root = verify_merkle(&deposit_tx_hash, &merkle_proof);
    assert_eq!(computed_root, state_root);

    // Calculate SOL amount
    let fee = eth_amount * 3 / 1000; // 0.3%
    let sol_amount = (eth_amount - fee) * exchange_rate;

    // Commit outputs (these go on-chain)
    sp1_zkvm::io::commit(&deposit_tx_hash);
    sp1_zkvm::io::commit(&sol_amount);
    sp1_zkvm::io::commit(&sol_recipient);
}

// SP1 automatically:
// 1. Generates STARK proof
// 2. Wraps in SNARK
// 3. Ready for Ethereum verification
```

22.5 Why zkVM Integration Matters

Benefits of zkVM Migration

1. **Faster Development:** 2-3 weeks vs 8-12 weeks for custom circuits
2. **Automatic Wrapper:** STARK-to-SNARK handled automatically
3. **Best Performance:** Fast STARK proving + cheap EVM verification
4. **Maintainability:** Write normal Rust, not circuit languages
5. **Flexibility:** Change logic without new trusted setup

22.6 Vendor Considerations

Dependency Trade-offs

Using Risc0 or SP1 creates some vendor dependency:

- Our proof format ties to their system
- We rely on their continued maintenance

Mitigations:

- Both are open source (Apache 2.0 / MIT)
- Can self-host provers (no cloud dependency)
- Can fork code if vendor disappears
- Our Rust logic is portable between zkVMs
- Both are well-funded (\$40-55M+)

22.7 Patent Implications

What's Patentable?

Using a zkVM does NOT affect Ghost Protocol's patent position:

Cannot Patent (public domain / others' IP):

- STARK/SNARK math (academic)
- zkVM prover implementations
- RISC-V instruction set

Can Patent (Ghost Protocol innovations):

- Ghost ID binding mechanism
- Soft finality settlement model
- Hybrid liquidity routing (Pool + DEX + Circle)
- Chain-specific proof optimization architecture
- Double-spend prevention system

Ghost Protocol's innovation is at the **protocol layer**, not the cryptographic layer. The implementation (zkVM vs custom) doesn't change what's patentable.

22.8 Why STARK Inside SNARK (Not Reverse)

The wrapping direction matters:

STARK INSIDE SNARK (Correct):

```

+-- STARK is FAST to generate (1-3 sec with M31)
+-- STARK is EXPENSIVE to verify on EVM (2M gas)
+-- SNARK is SLOWER to generate (2-4 sec for wrapper)
+-- SNARK is CHEAP to verify on EVM (200K gas)
|
+-- Solution: Generate fast STARK, wrap in SNARK for cheap verify

```

SNARK INSIDE STARK (Wrong):

```

+-- Would need to generate slow SNARK first
+-- Then wrap in STARK
+-- Then verify STARK on EVM (2M gas - expensive!)
|
+-- Defeats the purpose: slow + expensive

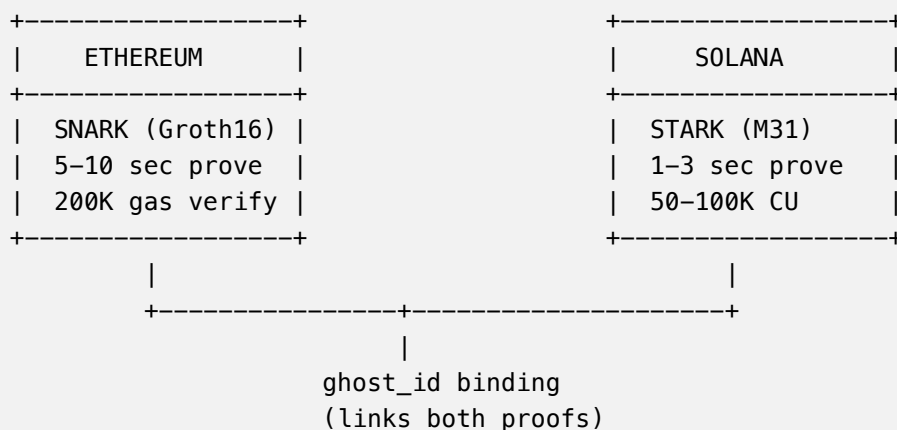
```

Remember

STARK is the inner proof (fast to generate). SNARK is the outer wrapper (cheap to verify on EVM).

22.9 Complete Migration Path

22.9.1 Phase 1: Current System (M31 Circle STARKs)



STATUS: Current architecture

M31 FIELD: Using Mersenne-31 for 7-10x faster STARK proving

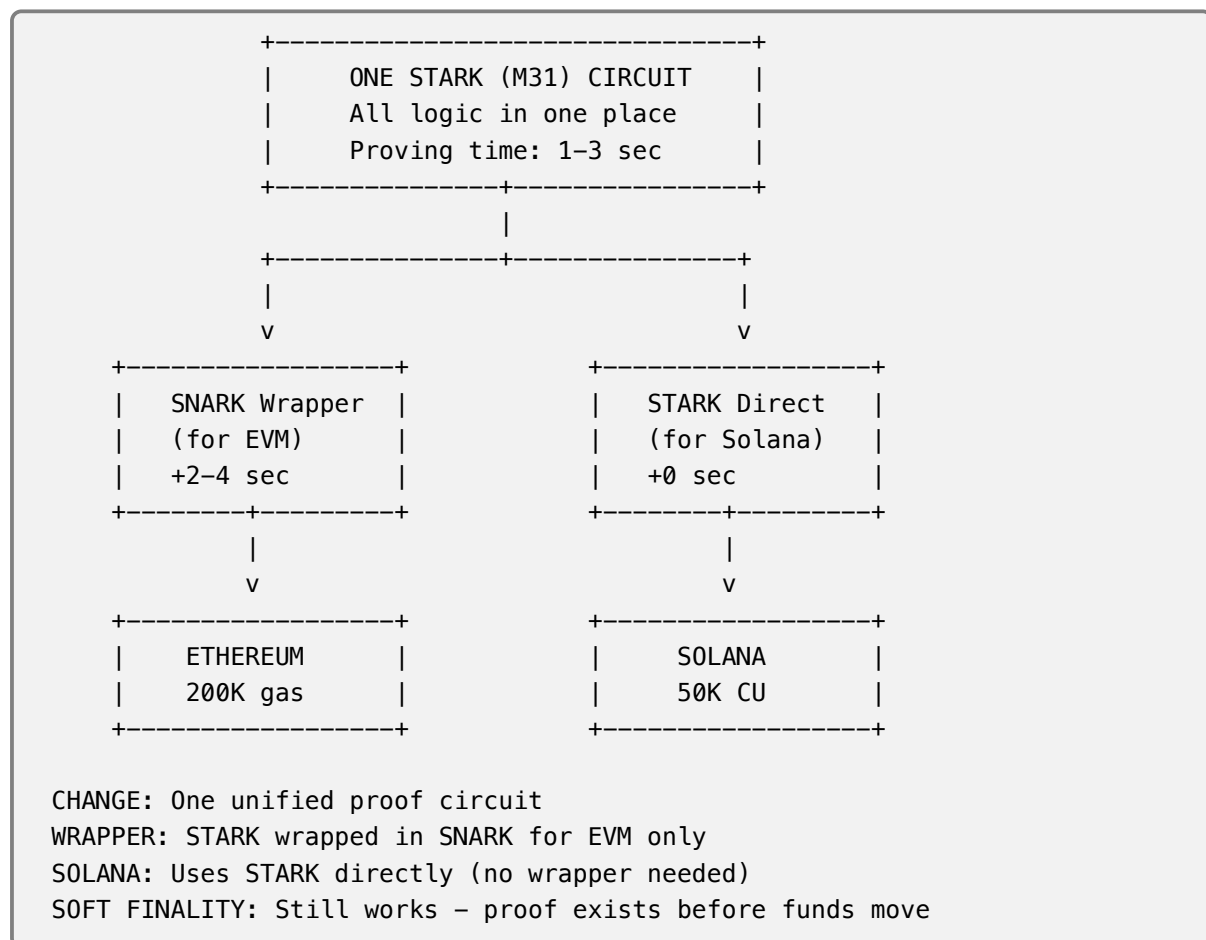
SOFT FINALITY: Yes – proof exists before funds move

Why M31 Circle STARKs Now

We use Circle STARKs with the Mersenne-31 field from day one because:

- 7-10x faster proving (1-3 sec vs 10-30 sec)
- 5x cheaper Solana verification
- Native 32-bit CPU operations
- No additional complexity vs traditional STARKs

22.9.2 Phase 2: Unified Architecture with Wrapper (Future)



22.10 Soft Finality Through All Phases

Key Guarantee

Soft finality works identically in ALL phases:

1. T+0s: User deposits ETH
2. T+1-5s: Proof generated (SNARK, STARK, or wrapped—doesn't matter)
3. T+2-6s: **Proof EXISTS** — cryptographically valid
4. T+3-7s: SOL sent to recipient ← **User is done**
5. T+40s: Proof batched and submitted on-chain ← Hard finality

The wrapping approach changes HOW you prove, not WHEN. The proof still exists before funds move.

22.11 Migration Timeline

| Phase | Architecture | Status | Soft Finality |
|-------------|-----------------------------|---------|---------------|
| 1 (Current) | SNARK (Groth16) + M31 STARK | Live | Works |
| 2 (Future) | Unified M31 STARK + wrapper | Planned | Works |

Current State

We ship with M31 Circle STARKs from day one. This gives us:

- 7-10x faster Solana proving (1-3 sec vs 10-30 sec)
- 5x cheaper Solana verification
- No additional implementation complexity

Phase 1 → 2: Upgrade to unified architecture with SNARK wrapper when we want single codebase. Uses zkVM (Risc0/SP1) for automation.
Soft finality works throughout. The architecture change is an optimization, not a requirement.