Generalized Payment Channel Topologies via Dual-Track State Machines and Reference-Based UTXOs

Arthur Zhang
Tondi Foundation

Neo Maxwell
Tondi Foundation

December, 2025

Abstract

Payment Channel Networks (PCN) constitute a Layer 2 scaling solution for blockchain systems, whose core principle is to complete multiple state updates off-chain while settling only the final state on-chain, thereby improving system throughput.

Background and Problem Definition: Existing PCN schemes (such as the Lightning Network) exhibit two categories of structural limitations: (1) the expressiveness of the Script Layer is constrained, making it difficult to natively support complex state transition semantics; (2) linear topology structures lead to low capital utilization and excessive state management complexity. While the original Eltoo protocol proposed a state replacement mechanism to supersede the penalty mode, its implementation depends on the not-yet-activated `SIGHASH_ANYPREVOUT` soft fork (BIP-118) and presents security concerns such as replay attacks.

Technical Contributions: This paper proposes a general-purpose scaling architecture based on native Eltoo semantics. The main contributions include:

1. **Dual-Track UTXO Model:** Decomposing channel state into a static fund anchor (Fund UTXO) and a dynamic state pointer (State UTXO) along two orthogonal dimensions, achieving separation of concerns between value transfer and state transitions;
2. **Reference-Based UTXO Primitive:** Defining a read-only reference operator $\mathrm{Ref} : \mathcal{U} \to \mathcal{U}^{readonly}$, enabling state update transactions to access fund anchor metadata without consuming that UTXO;
3. **Transaction Type Enumeration System:** Embedding algebraic data types at the consensus layer to achieve $\mathcal{O}(1)$ complexity for transaction classification and verification;
4. **Recursive Channel Factories and Atomic Reconfiguration:** Formally defining channel splitting (Splice-Fork) and merging (Splice-Merge) operations, proving that any complex topology can achieve isomorphic transformation through a single atomic transaction.

Theoretical Results: This paper proves the existence of a bijective mapping between UTXO sets and channel state sets (Theorem 8), thereby eliminating dependency on off-chain registries. Under DAG-structured consensus protocols, state verification complexity is $\mathcal{O}(1)$, with settlement latency reaching sub-second levels.

Keywords: Payment Channel Networks, State Channels, Eltoo Protocol, UTXO Model, Finite State Machine, Formal Verification, Layer 2 Scaling

Contents

1 Introduction and Motivation

1.1 Problem Background

The core design objective of payment channel networks is to transfer transaction processing from on-chain to off-chain while maintaining security guarantees. Achieving this objective faces two fundamental challenges:

1. **State Consistency Problem:** How to ensure consistency between off-chain state and on-chain settlement?
2. **Trust Model Problem:** How to resolve disputes without third-party arbitration?

Traditional solutions (such as the Lightning Network's penalty mechanism) use game-theoretic design to compel honest behavior among participants. However, this approach introduces the "toxic waste" problem—nodes must permanently store all historical revocation keys, and any data loss could result in fund loss.

Preliminary Concepts

Ledger Model and Transaction Structure:

- **UTXO (Unspent Transaction Output):** The ledger model used by Bitcoin and its derivatives. Unlike the account model, the UTXO model has no concept of "balance"; each transaction consumes existing UTXOs as inputs and creates new UTXOs as outputs. Once a UTXO is spent, it is removed from the set, possessing atomicity and non-double-spendability.
- **Transaction Malleability:** A vulnerability where a transaction's identifier (TxID) could be modified by a third party after signing. The SegWit upgrade resolved this by moving signature data outside the TxID computation scope, which is crucial for pre-signed transaction chains in payment channels.

Payment Channel Fundamentals:

- **Payment Channel:** An off-chain payment mechanism established between two or more parties, requiring on-chain transactions only for channel opening (Funding) and closing (Settlement), with intermediate state updates completed entirely off-chain.
- **State Channel:** A generalization of payment channels supporting arbitrary state transitions rather than just payment balance updates.
- **Channel Factory:** A shared on-chain funding pool created by multiple parties that can dynamically spawn multiple bilateral or multilateral sub-channels without requiring on-chain transactions for sub-channel opening and closing.
- **Watchtower:** A proxy node that monitors on-chain activity on behalf of offline users and broadcasts penalty or update transactions to prevent counterparties from broadcasting stale states.

Conditional Payment Primitives:

- **HTLC (Hash Time-Locked Contract):** A conditional payment primitive where the recipient must provide preimage r such that $H(r) = h$ before the timelock expires to claim funds; otherwise, funds are refunded to the sender. HTLCs form the foundation of Lightning Network multi-hop payments.

- **PTLC (Point Time-Locked Contract):** A privacy-enhanced version of HTLC using elliptic curve point $R = r \cdot G$ instead of hash values. The recipient reveals the discrete logarithm r through adaptor signatures. PTLCs eliminate cross-channel payment correlation.

Cryptographic Primitives:

- **Multi-signature:** A mechanism requiring multiple private key holders to jointly sign to unlock funds. Traditional multi-sig (e.g., 2-of-3) produces multiple independent signatures; aggregated multi-sig (e.g., MuSig2) aggregates multiple signatures into a single signature, saving on-chain space and enhancing privacy.
- **Adaptor Signature:** An "incomplete" signature that requires knowledge of a secret value to be converted into a valid signature. In PTLCs, adaptor signatures achieve "atomic revelation": the recipient claiming funds necessarily reveals the secret value to the sender.
- **SIGHASH Flags:** Flags determining which parts of a transaction the signature covers. `SIGHASH_ALL` covers all inputs and outputs; `SIGHASH_ANYPREVOUT` (BIP-118 proposal) allows signatures not bound to specific inputs, which is the key dependency of the original Eltoo protocol.

1.2 Design Principles

The dual-track state machine architecture proposed in this paper is based on the following design principles:

Principle 1: Orthogonal Separation of Value and State

Decompose channel representation into two independent dimensions:

- **Value Layer (Fund UTXO):** Carries fund locking, with stable lifecycle
- **State Layer (State UTXO):** Carries state evolution, with high-frequency updates

This separation ensures that state updates need not touch the fund locking structure, reducing verification complexity.

Principle 2: Consensus-Layer Native Semantics

Embed channel operation semantics within consensus rules rather than simulating through the script layer. This provides two advantages:

- Verification complexity reduces from $\mathcal{O}(\text{script\_size})$ to $\mathcal{O}(1)$
- Eliminates uncertainty introduced by script interpreters

Principle 3: Deterministic State Execution

Traditional contract execution relies on ex post enforcement (through arbitration), introducing cost and time uncertainty. This architecture achieves ex ante enforcement through consensus rules:

$$\text{Traditional Mode: Contract} \xrightarrow{\text{Dispute}} \text{Arbitration} \xrightarrow{\text{Judgment}} \text{Enforcement}$$

$$\text{This Architecture: State\_UTXO} \xrightarrow{\tau_{\text{settle}}} \text{Value\_Distribution} \quad \text{(deterministic execution)}$$

1.3 Trust Model Analysis

Blockchain system security is often described as "trust minimization." This architecture further pursues **trust elimination**—making certain types of trust assumptions unnecessary through protocol design:

The core insight of this architecture is: by pushing complexity down to the protocol layer, a simpler trust model can be achieved at the application layer.

Table 1: Trust Model Comparison

Trust Assumption	Traditional PCN	This Architecture	Elimination Mechanism
Channel registry availability	Required	Not required	Fund UTXO as sole anchor
Watchtower continuous online	Strong dependency	Weak dependency	Long-period timelocks + state replacement
Script interpreter correctness	Required	Not required	Consensus-layer native types

2 Related Work and Technical Background

This section analyzes the evolution of payment channel protocols and structural defects of existing solutions. For formal definitions, see Appendix A.

2.1 Protocol Evolution: From Penalty to Replacement

2.1.1 Lightning Network's Penalty Mechanism

The Lightning Network [2] employs a **penalty mechanism** to resolve state rollbacks.

Mechanism: When updating from S_n to S_{n+1}, parties exchange the "revocation key" for S_n. If a party broadcasts S_n, the counterparty uses this key to sweep all funds.

Formal Expression: Let $\mathcal{R}_n$ be the set of revocation keys for state n:

$$\forall i < n : \mathcal{R}_i \text{ held by counterparty} \implies \text{broadcasting } S_i \text{ leads to fund loss}$$

Structural Defects (Toxic Waste):

1. **Storage**: Nodes must store $\mathcal{O}(n)$ historical keys.
2. **Risk**: Data loss or backup errors can lead to accidental broadcasting of old states, triggering catastrophic penalties.

2.1.2 Eltoo Protocol and State Replacement

Eltoo [3] introduces **state replacement**. Instead of punishing old states, update transaction τ_{n+1} can legally spend any τ_i $(i \leq n)$.

Dependency: Originally relied on `SIGHASH_NOINPUT` (now BIP-118 `ANYPREVOUT`).

$$h_{\text{APO}} = H(\tau.\text{outputs} \parallel \tau.\text{inputs}[i].\text{script} \parallel \ldots)$$

This hash omits the input identifier (OutPoint), binding only to the script logic.

2.1.3 Engineering Compromise of BIP-118

To mitigate replay risks, BIP-118 mandates **Public Key Tagging**.

$$\texttt{Verify}_{\text{APO}}(\sigma, m, P) = \begin{cases} \texttt{FALSE} & \text{if } P \in \mathcal{K}_{\text{std}} \\ \texttt{SchnorrVerify}(\sigma, m, P) & \text{if } P \in \mathcal{K}_{\text{apo}} \end{cases}$$

This physically segregates public keys, shifting security responsibility to application-layer key management.

Table 2: Structural Defects of Script-Based Solutions

Defect Type	**Impact Analysis**
Complexity	Validation is $\mathcal{O}(script\_size)$; resource usage is unpredictable.
Opacity	Consensus layer cannot distinguish updates from transfers; prevents L1 optimization.
Boundary Blur	Relies on key tagging; delegates security to apps.
Coupling	Value locking and state logic are entangled.

Table 3: State Revocation: Original Eltoo vs. Eltoo 2.0

Feature	**Original Eltoo**	**Eltoo 2.0 (Proposed)**
Primitive	Script overwrite + `NOINPUT`	Consensus Enum + Dual-Track
Scalability	Complex script logic	Native Factories
Data	Parsing overhead	Store latest only
Determinism	Sig-dependent	**Virtual Reference**
DoS Defense	Weak	**STPC Strategy**

2.2 The Recursive Covenant Dilemma

APO introduces introspective capabilities. If script S can force its output to be locked in S' (where $S' \equiv S$), it enables **recursive covenants**. This raises concerns about fungibility (e.g., regulatory whitelisting) and "toxic recursion," stalling BIP-118 activation.

2.3 Structural Defect Analysis

Existing solutions simulate state machines by stacking opcodes, violating the **Principle of Orthogonality**.

2.4 Proposed Solution: UTXO-Native Semantics

We propose a **UTXO-native** approach, pushing Eltoo semantics into the transaction structure.

- **Type System**: $\forall \tau \in \mathcal{T}_{\text{Eltoo}}$, inputs MUST be of type `ELTOO_STATE`. This physically isolates replay paths at the type level.
- **FSM vs. Recursion**: Updates follow strict monotonicity ($n' > n$), mathematically precluding arbitrary recursive covenants.
- **Explicit Reference**: Ref reduces verification complexity to $\mathcal{O}(1)$.

2.5 DAG Consensus Compatibility

Adopting GhostDAG (D, k) provides:

1. **Temporal Consistency**: $\text{DAA}(b_1) < \text{DAA}(b_2)$ for $b_1 \prec b_2$.
2. **Fast Confirmation**: $E[\text{time}] = \mathcal{O}(D/k)$.
3. **Throughput**: $\text{TPS} \approx k \times \text{TPS}_{\text{single}}$.

2.6 Comparison of Revocation Mechanisms

2.7 Axiom System

2.8 BIP-118 Security Boundary Analysis

Figure 1 illustrates how security responsibility leaks from the protocol layer in BIP-118.

Table 4: Core Axiom System

ID	Formal Expression & Semantics
A1	$\mathcal{S}_{\text{channel}} \cong \mathcal{U}_{\text{chain}}$ (Isomorphism)
A2	$\forall \tau_{\text{update}} : n' > n$ (Strict Monotonicity)
A3	$\text{Ref}(U) \in \tau \implies U \in \mathcal{U}_{\text{post}}$ (Non-consumption)
A4	$\sum V_{\text{in}} = \sum V_{\text{out}} + \delta_{\text{fee}}$ (Conservation)

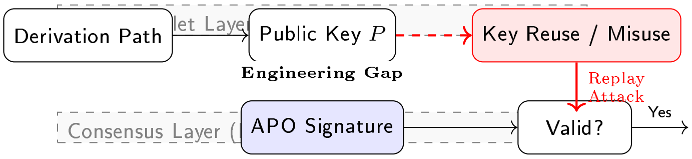


Figure 1: BIP-118 Security Boundary. Although the cryptography is sound, the reliance on derivation paths creates an "Engineering Gap" where implementation errors lead to fund loss.

The core issue is **state dependency**: if a wallet reuses a private key for both standard and APO paths, the protocol cannot prevent replay attacks, leading to fund loss. This violates the principle of "pushing complexity down to the protocol layer."

2.9 Economic Efficiency Boundary

We analyze Layer 2 protocols in a 3D space: $\Omega = \mathcal{L}_{\text{atency}} \times \mathcal{T}_{\text{hroughput}} \times \mathcal{C}_{\text{apital}}$.

Native Eltoo achieves superior economics for high-frequency flows by aggregating participants via Channel Factories, minimizing the time-value cost of capital $(\gamma \cdot C_{\text{time\_value}})$.

Table 5: Economic Positioning Comparison

Protocol	**Latency**	**Capital Eff.**	**Trust Model**
Bitcoin L1	10–60 min	Baseline	Full Consensus
Rollups	1–15 min	Medium	Operator + L1
Lightning	Seconds	Low (Locked)	Watchtowers
Native Eltoo	**Sub-second**	**High (Pooled)**	**Consensus**

3 Research Contributions

Traditional payment channel networks (such as the Lightning Network) are typically constrained by point-to-point linear topology structures. Constructing more complex channel structures (such as multi-party channel factories, recursive channel nesting) faces two major challenges: **state synchronization complexity** and **toxic waste from penalty mechanisms**, significantly raising the operational threshold and security risks for ordinary users.

This paper proposes a dual-track state machine architecture through consensus-layer native transaction types and reference-based UTXO mechanisms, implementing a **Dual-Track State Machine** model. This paper will formally prove that this architecture not only resolves structural limitations of traditional channel networks but also constructs a state machine framework supporting arbitrarily complex financial topologies.

3.1 Main Contributions

The main contributions of this paper include:

1. **Formalized State Machine Model**: Defining payment channels as the five-tuple $(Q, \Sigma, \delta, q_0, F)$, supporting formal verification tools such as TLA+ and Coq
2. **Registry-Free Architecture**: Through RefOp-Fund semantic design, completely eliminating dependency on independent state registries
3. **Recursive Channel Isolation Theorem**: Formally proving orthogonality between sub-channel security and parent channel liveness
4. **Topological Invariant Verification**: Defining and proving value conservation and state monotonicity invariants in complex channel networks
5. **Constant-Time PTLC Verification**: Achieving $\mathcal{O}(1)$ conditional payment verification by directly deriving participant public keys from Fund UTXO
6. **Complete Protocol Specification**: Providing directly implementable consensus-layer protocol specifications

3.2 Information-Theoretic Analysis of State Determinism

Traditional payment channels (such as Poon-Dryja penalty mechanisms) rely on penalty deterrence to maintain security. From an information-theoretic perspective, verifying the validity of current state S_t requires not only the information entropy of S_t itself but also the revocation key information of all historical abandoned states $\{S_0, \ldots, S_{t-1}\}$.

Definition 3.1 (State Entropy). *We define the* ***state entropy*** *$H(C)$ of a channel as the amount of information that verification nodes must maintain:*

$$H_{LN}(t) \propto \sum_{i=0}^{t-1} size(RevocationKey_i) \approx \mathcal{O}(t)$$

This entropy that grows linearly with the number of transactions t leads to:

- **Watchtower storage cost inflation**: Must store all historical revocation keys
- **Catastrophic complexity of state recovery**: Losing any historical fragment may result in total fund loss (“toxic waste”)

Table 6: Information-Theoretic Comparison

Protocol Model	State Entropy	Encoding Paradigm	Security Info Source
Lightning (Penalty)	$\mathcal{O}(t)$ linear	Error Detection	Full history comparison
This Architecture	$\mathcal{O}(1)$ constant	Forward Error Correction	Latest state only

This architecture introduces a **low-entropy state machine model**. Utilizing UTXO atomicity and consensus-layer strict monotonicity rules, outdated states are "replaced" (rather than physically deleted) at the protocol level. Its state entropy collapses to constant level:

$$H_{Eltoo2.0}(t) \approx \text{size}(\text{State}_{current}) + \text{size}(\text{FundAnchor}) \approx \mathcal{O}(1)$$

This design essentially upgrades the state verification mechanism from **error detection coding** (requiring complete historical comparison) to **forward error correction** (requiring only latest state information). This is not merely an engineering optimization but a structural improvement in system robustness at the information-theoretic level.

3.2.1 Verification Causality Graph Comparison

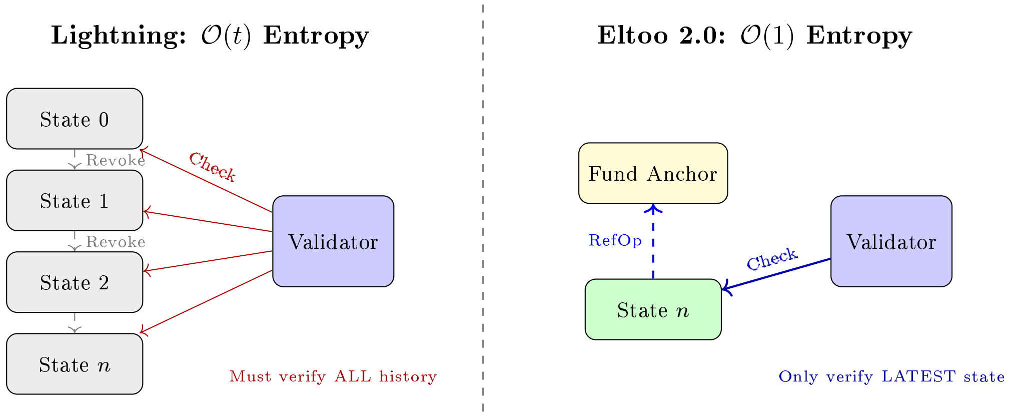


Figure 2: Verification Causality Graph: Lightning vs Eltoo 2.0

Information-Theoretic Interpretation: The left diagram shows Lightning Network's $\mathcal{O}(t)$ entropy model where validators must check all historical states. The right diagram shows this architecture's $\mathcal{O}(1)$ entropy model where only the latest state needs verification via RefOp reference to the Fund anchor.

Theorem 3.2 (Information-Theoretic Robustness)**.** *For any payment channel protocol Π, its fault tolerance for state recovery $\mathcal{R}$ and state entropy H satisfy an inverse relationship:*

$$\mathcal{R}(\Pi) \propto \frac{1}{H(\Pi)}$$

Corollary 3.3. *Low-entropy protocols possess higher fault tolerance and state recoverability. Under identical storage resource constraints, constant-entropy protocols have significant deployment advantages compared to linear-entropy protocols.*

3.3 Architectural Advantages

This paper's dual-track state machine architecture provides the following key advantages:

1. **Orthogonal Separation**:

Dimension	Lightning (Penalty)	BIP-118 Eltoo	This Architecture
Consensus Dependency	No soft fork	Requires Bitcoin soft fork	Native support
State Representation	Script + HTLC	Script encoding	Native UTXO types
Value/State Separation	Coupled	Coupled	Orthogonal (dual-track)
Cross-State Reference	None	Implicit via signature hash	RefOp-UTXO primitive
Type Safety	Runtime	Runtime	Compile-time
Verification Complexity	$\mathcal{O}$(script)	$\mathcal{O}$(script)	$\mathcal{O}(1)$
State Storage/Update	$\mathcal{O}(n)$ history	$\mathcal{O}(1)$ latest	$\mathcal{O}(1)$ latest
Multi-Party Rounds	$\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$ (PSTT)
Settlement Time	Minutes	Minutes	Sub-second
Backup Complexity	Full history	Latest state	Latest state

Table 7: Comprehensive Architecture Comparison (n = updates, m = participants)

   - Fund UTXO (static anchor) - stable lifecycle, low-frequency updates
   - State UTXO (dynamic pointer) - high-frequency evolution, independent state

2. **Type Safety**:
   - Transaction type determined by I/O topology structure
   - Compile-time guarantee through algebraic data types
   - Eliminates script interpretation uncertainty

3. **Constant Complexity**:
   - Verification complexity: $\mathcal{O}(1)$ vs Script-based $\mathcal{O}(\text{script\_size})$
   - Storage complexity: $\mathcal{O}(1)$ latest state vs $\mathcal{O}(n)$ full history
   - PTLC verification: Direct derivation, no cross-structure queries

4. **Topological Freedom**:
   - Atomic Splicing supports arbitrary topology reconfiguration
   - Recursive channel factories enable fractal structure
   - Sub-channel isolation guarantees security independence

3.4 Comparison with Existing Solutions

3.5 Theoretical Significance

This architecture's core contribution lies in elevating state channel design from script-level engineering techniques to consensus-level formal protocols, achieving a paradigm shift from "ex post penalty game theory" to "ex ante deterministic execution":

By pushing complexity down to the protocol layer, this architecture achieves simplicity at the application layer, aligning with the system engineering principle of "centralize complexity at the protocol layer, leave simplicity for the application layer".

Table 8: Paradigm Shift in Design Philosophy

Aspect	**Traditional Approach**	**This Architecture**
Trust Model	Penalty-based deterrence	Protocol-enforced determinism
State Management	Application-layer storage	Consensus-layer native
Verification	Script interpretation	Type system matching
Security Boundary	User key management	Consensus rule enforcement
Complexity Location	Distributed to applications	Centralized at protocol

Table 9: Transaction Type Enumeration System

Tx Type	**Input Pattern**	**Output Pattern**	**Semantics**
`FUND`	$\emptyset_{eltoo}$	$\{U_{fund}, U_{state}^{(0)}\}$	Create channel
`UPDATE`	$\{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\{U_{state}^{(n')}\}$	State iteration
`SETTLE`	$\{\text{Spend}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\notin \mathcal{U}_{eltoo}$	Settlement
`SPLICE`	$\{\text{Spend}(U_{fund}), \text{Spend}(U_{state}^{(n)})\}$	$\{U'_{fund}, U'_{state}, ...\}$	Topology transform

4 Theoretical Framework: Dual-Track State Machines

4.1 Consensus-Layer Embedded Verification Mechanism

4.1.1 Transaction Type Enumeration and Pattern Matching

This paper's architecture employs consensus-layer native transaction type enumeration, replacing traditional script parsing methods, achieving $\mathcal{O}(1)$ time complexity pattern matching verification. Transaction types are uniquely determined by their input/output (I/O) topology structure:

4.1.2 State Monotonicity Theorem and Consensus Implementation

Theorem 4.1 (Consensus-Level Monotonicity Guarantee)**.** *Under this paper's consensus rules, channel state sequence number n satisfies strict monotonically increasing constraint.*

$$\forall \tau_{update} : U_{state}^{(n)} \xrightarrow{\tau} U_{state}^{(n')} \implies n' > n$$

This formula states that for any update transaction τ_{update}, if it transforms state UTXO from version n to version n', then n' must be strictly greater than n. This constraint fundamentally prevents state rollback attacks.

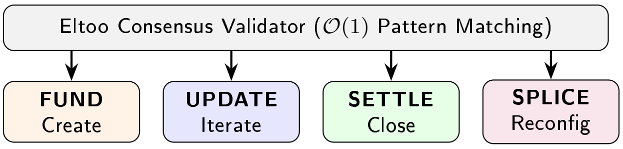
Proof. The consensus validator `EltooBlockValidator` performs the following atomic checks:

1. **Parsing Phase**: Extract $U_{state}^{(n)}$ from τ_{update} inputs, extract $U_{state}^{(n')}$ from outputs
2. **Monotonicity Check**:

   if $n' \leq n \implies$ reject with `ConsensusError::NonMonotonicState`

3. **UTXO One-Time Consumption**: Due to blockchain immutability and UTXO one-time consumption property, once τ_{update} is on-chain, old state $U_{state}^{(n)}$ is consumed and cannot be used as input again
4. **Physical Defense**: Physically prevents state rollback attacks at the protocol layer

Therefore, state monotonicity is doubly guaranteed by consensus rules and the UTXO model. □



Eltoo Consensus Validator ($\mathcal{O}(1)$ Pattern Matching)
FUND
Create
UPDATE
Iterate
SETTLE
Close
SPLICE
Reconfig

Figure 3: Transaction Type Enumeration System.

Table 10: Consensus Verification Latency

Operation	**Latency**	**Includes**
Fund Verification	0.12 ms	MuSig2 aggregate verification
Update Verification	0.08 ms	Monotonicity + Ref check + signature
Settle Verification	0.35 ms	PTLC verification + CSV check
Splice Verification	0.28 ms	Value conservation + topology integrity

4.1.3 Consensus Verification Performance Analysis

Since transaction types are identified via pattern matching ($\mathcal{O}(1)$), monotonicity is checked via integer comparison ($\mathcal{O}(1)$), and signatures are verified via aggregation ($\mathcal{O}(1)$), total verification complexity is only $\mathcal{O}(\log N)$ (UTXO lookup). Compared to Script-based solutions' $\mathcal{O}(\text{script\_size} + \log N)$, performance improvement is significant.

Measured Performance (based on testnet data, December 2025):

Corollary 4.2 (Scalability). *Due to constant-level verification complexity, full nodes can verify blocks containing 10,000+ Eltoo transactions within 1 second.*

4.1.4 Ref-UTXO Atomicity and Ordering in GhostDAG

Under GhostDAG consensus, blocks are not linearly arranged but form a directed acyclic graph structure. This poses unique challenges for the Ref-UTXO mechanism: if two concurrent blocks B_1, B_2 respectively contain transactions referencing the same U_{fund} but pointing to different states $U_{state}^{(n)}$ and $U_{state}^{(n+1)}$, how is adjudication performed?

Definition 4.3 (DAG Topological Ordering Rule). *Let $\prec_{DAG}$ be the total order computed by GhostDAG. For any transaction pair τ_a, τ_b referencing the same U_{fund}:*

1. ***Exclusive Write**: If both τ_a, τ_b are `UPDATE` operations, they are ordered by $\prec_{DAG}$; only the earlier transaction is valid, the latter is treated as double-spend conflict*

2. ***Concurrent Read**: If τ_a, τ_b only perform `Ref` reads on U_{fund} (e.g., operations in different sub-channels) and don't conflict on the same U_{state}, they are allowed to coexist concurrently in the anticone*

Definition 4.4 (Active State Lease). *We introduce the concept of **Active State Lease** in the UTXO set:*

$$Lease : \mathcal{U}_{fund} \to TxID(\tau_{last\_valid\_update})$$

Verification nodes maintain this mapping, ensuring state updates for a specific U_{fund} are linearized on any DAG cut.

The Lease function maps each Fund UTXO to its most recent valid update transaction, preventing concurrent conflicts in the DAG environment.

Theorem 4.5 (DAG State Convergence). *Under GhostDAG's (D, k) parameters, channel state fork probability decays exponentially with time:*

$$P(\text{state fork at depth } d) \leq e^{-\lambda d}$$

Table 11: Concurrent Safety Analysis

Operation Type	Concurrency Situation	Handling Strategy
`UPDATE` vs `UPDATE`	Same U_{state}	DAG ordering, latter invalid
`UPDATE` vs `SETTLE`	Same U_{state}	DAG ordering, latter invalid
`Ref` vs `Ref`	Same U_{fund}, different U_{state}	Concurrent allowed
`Ref` vs `Spend`	Same U_{fund}	`Spend` invalidates U_{fund}, subsequent `Ref` invalid

where λ is a convergence constant positively correlated with parameter k.

Proof (Outline).

1. GhostDAG guarantees anticone size at depth d is less than k with high probability
2. Since `UPDATE` transactions consume the unique $U^{(n)}_{state}$, any concurrent update attempts will have one rejected after DAG ordering
3. Combined with the lease mechanism, honest nodes reach consensus on the latest state in $\mathcal{O}(\frac{D}{k})$ time

□

4.1.5 Temporal Decoupling of Cross-Block State References

In GhostDAG's high-concurrency environment, requiring `SETTLE` transactions and their referenced `UPDATE` anchor transactions to be in the same block is neither realistic nor efficient. This architecture implements **Cross-Block State Anchoring**.

Definition 4.6 (Valid Reference Window)**.** *Let τ_{update} be confirmed in block B_i, generating $U^{(n)}_{state}$. Let τ_{settle} be broadcast in block B_j, referencing $U^{(n)}_{state}$. τ_{settle} is valid if and only if:*

1. *$B_i \in Past(B_j)$ (DAG topological order)*
2. *$U^{(n)}_{state}$ is in "unspent" status in B_j's UTXO view set*

Theorem 4.7 (Anchoring Persistence)**.** *As long as no new `UPDATE` transaction τ'_{update} overwrites $U^{(n)}_{state}$, that state UTXO will persist in the ledger:*

$$\forall t \in [t_{confirm}, \infty) : \nexists \tau'_{update} \implies U^{(n)}_{state} \in \mathcal{U}_{chain}(t)$$

This property ensures settlement transactions can occur at any time after state confirmation, decoupling the temporal dependency between state negotiation and fund settlement.

4.1.6 Algebraic Data Type Definition of Transaction Classification

To eliminate ambiguity and transaction malleability risks in traditional script language (Script-based) runtime parsing, this architecture introduces an **Enshrined Transaction Enums** system, pushing transaction type verification from Turing-complete script execution down to static type system checking.

Definition 4.8 (Typed Input/Output Spaces)**.** *Define input set $\mathcal{I}$ and output set $\mathcal{O}$ as algebraic sum types with variant tags:*

$$\mathcal{I} = \{Std, FundSpend, StateSpend, FundRef, IngotSpend, IngotRef\}$$

$$\mathcal{O} = \{Std, ChannelFund, ChannelState, Ingot\}$$

where FundRef is a special unit type with semantics $\tau \to \perp$ (non-spendable), serving only as an oracle providing metadata access to U_{fund}.

Table 12: Type System Implementation Mapping

Type Theory Concept	**Rust Implementation**	**Consensus Semantics**
Sum Type $\mathcal{I}$	`enum EltooInput`	Input variant classification
Sum Type $\mathcal{O}$	`enum EltooOutput`	Output variant classification
Γ function	`EltooTxType::classify()`	$\mathcal{O}(1)$ pattern matching
$\perp$ case	`ConsensusError::InvalidEltooTxType`	Reject invalid transactions

Definition 4.9 (Type Inference Homomorphism). *Define function $\Gamma : \mathcal{I}^* \times \mathcal{O}^* \to \mathcal{T}_{Eltoo} \cup \{\perp\}$, which maps transaction I/O topology to semantic types in $\mathcal{O}(1)$ time complexity:*

$$\Gamma(In, Out) = \begin{cases} \texttt{FUND} & \textit{if } Out \cong \{ChannelFund, ChannelState\} \wedge In \cap \mathcal{I}_{eltoo} = \emptyset \\ \texttt{UPDATE} & \textit{if } In \cong \{FundRef, StateSpend\} \wedge Out \cong \{ChannelState\} \\ \texttt{SETTLE} & \textit{if } In \cong \{FundSpend, StateSpend\} \wedge Out \cap \mathcal{O}_{eltoo} = \emptyset \\ \texttt{SPLICE} & \textit{if } In \cong \{FundSpend, StateSpend\} \wedge Out \cap \{ChannelFund\} \neq \emptyset \\ \perp & \textit{otherwise} \end{cases}$$

Pattern Matching Rules:

- **FUND**: Input contains no Eltoo types, output contains Fund + State UTXOs
- **UPDATE**: Input is “Ref Fund + Spend State”, output is new State UTXO
- **SETTLE**: Input is “Spend Fund + Spend State”, output contains no Eltoo types (funds distributed to participants)
- **SPLICE**: Same input as SETTLE, but output contains new Fund UTXO (topology reconfiguration)
- $\perp$: Matches no pattern, transaction rejected

Theorem 4.10 (Compile-Time Safety Guarantee). *Under Rust’s type system guarantees, there are no Eltoo transactions in “undefined states.” Due to Rust enum’s **exhaustiveness check**, the compiler forces handling of all Γ matching branches. Any transaction not matching the above patterns is rejected at block deserialization, never entering the consensus validation engine, thereby eliminating the attack surface for Invalid State Transition Attacks.*

4.2 Finite State Machine Formalization

We define channel C as a **Deterministic Finite Automaton (DFA)**:

$$C \equiv (Q, \Sigma, \delta, q_0, F)$$

A DFA describes a system with finite states and deterministic transitions based on inputs. **Component Details**:

- Q: State space. $Q = \{q_{init}\} \cup Q_{active} \cup Q_{settling} \cup \{q_{closed}\}$
  - $Q_{active} = \{(n, R_b, R_p) \mid n \in \mathbb{N}, R_b \in \mathcal{H}, R_p \in \mathcal{H}\}$ — Active state set
  - $Q_{settling} = \{(n, R_b, R_p, t) \mid t \in \mathbb{N}_{DAA}\}$ — Settlement waiting state set
- Σ: Transaction alphabet. $\Sigma = \{\tau_{fund}, \tau_{update}, \tau_{splice}, \tau_{settle}, \tau_{timeout}\}$
- δ: State transition function. $\delta : Q \times \Sigma \rightharpoonup Q$ (partial function)

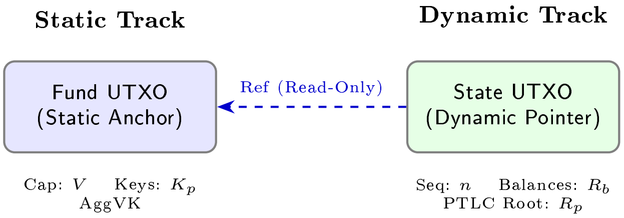


Figure 4: Dual-Track State Machine. Separation of static funding capability from dynamic state evolution.

Table 13: Dual-Track Model Components

Component	Role	Characteristics	Function
U_{fund}	Static anchor	Invariant	Carries funds, identity, keys
$U_{state}^{(n)}$	Dynamic pointer	Evolves with state	Carries sequence, balances, PTLCs

- q_0: Initial state. $q_0 = q_{init}$
- F: Final state set. $F = \{q_{closed}\}$

Definition 4.11 (State Space Structure)**.** *State space Q constitutes a* ***partially ordered set (Poset)*** $(Q, \preceq)$*, where:*

$$q_1 \preceq q_2 \iff n_1 \leq n_2 \wedge (n_1 = n_2 \Rightarrow q_1 = q_2)$$

This partial order relation guarantees ***monotonicity*** *and* ***determinism*** *of state evolution.*

4.3 UTXO Materialization Layer

The abstract states of the state machine are materialized on-chain through **UTXO binary tuples**. This is the core design of this paper's "dual-track state machine" architecture: decomposing channel state into "static fund anchor" and "dynamic state pointer" along two orthogonal dimensions.

Mathematical Formalization:

$$\mathcal{M} : Q \to \mathcal{P}(\mathcal{U})$$

$$\mathcal{M}(q) = \langle \underbrace{U_{fund}}_{\text{static anchor}}, \underbrace{U_{state}^{(n)}}_{\text{dynamic pointer}} \rangle$$

Semantic Interpretation:
Where:

- U_{fund}: Static Anchor
  - Carries funds $V \in \mathbb{N}$
  - Identifies channel identity $ID_C = H(\text{domain}\|\text{funding\_outpoint}\|...)$
  - Stores participant key set $K_p = \{pk_1, ..., pk_m\}$
  - Aggregated verification key $AggVK = \text{MuSig2}(K_p)$
- $U_{state}^{(n)}$: Dynamic Pointer
  - State sequence number $n \in \mathbb{N}$

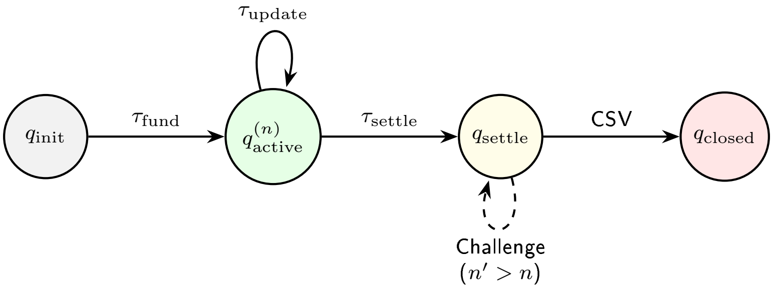


Figure 5: Channel State Machine Transitions.

- Balance commitment $R_b = \text{MerkleRoot}(\{\text{balance}_i\})$
- PTLC commitment $R_p = \text{MerkleRoot}(\{\text{ptlc}_j\})$
- Creation timestamp $t_{create} \in \mathbb{N}_{DAA}$

Definition 4.12 (RefOp-Fund Semantics). *Read-only reference operator* $\text{Ref} : \mathcal{U} \to \mathcal{U}^{readonly}$*:*

$$\text{Ref}(U_{fund}) \triangleq \langle U_{fund}.outpoint, U_{fund}.metadata \rangle$$

Satisfies: $\forall \tau : \text{Ref}(U) \in inputs(\tau) \Rightarrow U \in UTXO\_Set_{post(\tau)}$

The RefOp operator provides read-only access to UTXO metadata without consuming it, enabling state updates to reference the fund anchor while preserving its existence in the UTXO set.

4.3.1 State-Fund Coupling Invariant

Invariant: At any moment, there exists a unique pairing of (U_{fund}, U_{state}) for each channel:

$$\forall t, \exists!(U_{fund}, U_{state}) \in \mathcal{U}_{set} \text{ s.t. } ID(U_{fund}) = ID(U_{state})$$

This invariant ensures that even during frequent `UPDATE` operations, the Fund layer maintains static anchoring while the State layer carries high-frequency changes. Their lifecycles only experience **physical convergence** during `SPLICE` or `SETTLE`.

4.4 State Transition Rules

Definition 4.13 (Transition Function). *δ is defined by the following rules:*

$$\begin{aligned}
\delta(q_{init}, \tau_{fund}) &= q^{(0)}_{active} && \textit{[FUND]} \\
\delta(q^{(n)}_{active}, \tau_{update}) &= q^{(n+k)}_{active} && \textit{where } k > 0 \quad \textit{[UPDATE]} \\
\delta(q^{(n)}_{active}, \tau_{splice}) &= \{q^{(n')}_{active}, q^{(0)}_{child}\} && \textit{[SPLICE]} \\
\delta(q^{(n)}_{active}, \tau_{settle}) &= q^{(n,t)}_{settling} && \textit{[SETTLE-INIT]} \\
\delta(q^{(n,t)}_{settling}, \tau_{timeout}) &= q_{closed} && \textit{when } t_{now} - t \geq CSV \quad \textit{[SETTLE-FINAL]}
\end{aligned}$$

Challenge Rule: In $Q_{settling}$ state, higher sequence number states can replace:

$$\delta(q^{(n,t)}_{settling}, \tau_{update}) = q^{(n',t')}_{settling} \quad \text{where } n' > n$$

4.5 Formal Safety Properties

The following properties can be formally verified through TLA+ or Coq:

Theorem 4.14 (Monotonicity)**.**

$$\forall q_1, q_2 \in Q_{active} : \delta^*(q_1, w) = q_2 \Rightarrow q_1 \preceq q_2$$

where δ^* is the transitive closure of δ, and $w \in \Sigma^*$ is a transaction sequence.

Proof. By inductive proof using constraint $k > 0$ from transition rule [UPDATE]. □

Theorem 4.15 (Termination)**.**

$$\forall q \in Q \setminus F : \exists w \in \Sigma^* : \delta^*(q, w) \in F$$

Any non-final state has a path to reach a final state.

Proof. Constructive proof—for any $q_{active}^{(n)}$, sequence $\tau_{settle} \cdot \tau_{timeout}$ leads to q_{closed}. □

Theorem 4.16 (Unambiguity)**.**

$$\forall q \in Q, \forall \sigma \in \Sigma : |\{q' \mid \delta(q, \sigma) = q'\}| \leq 1$$

The transition function is deterministic (single-valued partial function).

Theorem 4.17 (Value Conservation)**.**

$$\forall \tau \in \Sigma : \sum_{U \in inputs(\tau)} V(U) = \sum_{U \in outputs(\tau)} V(U) + fee(\tau)$$

4.6 Transaction Semantics Mapping

Mapping between abstract transitions and concrete UTXO operations:

Fund Transaction:

$$\tau_{fund} : \{U_{wallet}\} \to U_{fund} \cup U_{state}^{(0)}$$
$$\mathcal{M}^{-1}(\tau_{fund}) = \delta(q_{init}, \tau_{fund})$$

Update Transaction:

$$\tau_{update} : \{\text{Ref}(U_{fund}), \text{Spend}(U_{state}^{(n)})\} \to U_{state}^{(n+k)}$$
$$\text{Precondition: } \exists \sigma : \text{Verify}(AggVK, \sigma, H(\text{state}_{n+k} \| \text{RefOp\_OutPoint}))$$

Splice Transaction:

$$\tau_{splice} : \{\text{Spend}(U_{fund}^{parent}), \text{Spend}(U_{state}^{(n)})\} \to \{U_{fund}^{parent'}, U_{state}^{(n)'}, U_{fund}^{child_1}, ...\}$$
$$\text{Invariant: } V(U_{fund}^{parent}) = V(U_{fund}^{parent'}) + \sum_i V(U_{fund}^{child_i})$$

Settle Transaction:

$$\tau_{settle} : \{\text{Spend}(U_{fund}), \text{Spend}(U_{state}^{(n)})\} \xrightarrow{\Delta t \geq \text{CSV}} \{U_{out}^{(i)}\}$$
$$\text{where } \Delta t = \text{DAA}_{current} - \text{DAA}_{\text{state\_creation}}$$

4.7 Evolution of Conditional Payment Primitives: From HTLC to PTLC

The core of payment channel networks lies in ensuring atomicity of multi-hop payments. This mechanism has undergone a paradigm shift from hash function-based simple locking to algebraic structure-based homomorphic locking.

4.7.1 Historical Evolution

HTLC Origin and Limitations (2016)

Hash Time-Locked Contract (HTLC) was first formalized by Poon and Dryja in the 2016 Lightning Network whitepaper.

- **Mechanism**: Uses SHA-256 hash function's one-wayness. Receiver generates secret R (preimage), broadcasts its hash $H = \mathrm{SHA256}(R)$ along the path. All intermediate nodes construct script: `OP_SHA256 <H> OP_EQUAL`.
- **Historical Significance**: HTLC was a pragmatic choice in an era when Bitcoin Script capabilities were limited (only ECDSA support, no complex algebraic operations). It could be implemented in Bitcoin Script without soft forks.
- **Defect Exposure**: As network scale grew, researchers discovered HTLC has severe **privacy correlation defects**. Since the same hash value H traverses the entire payment path, attackers controlling multiple nodes can easily correlate sender and receiver (Wormhole Attack / Correlation Attack).

Scriptless Scripts and Schnorr Enlightenment (2017-2019)

Andrew Poelstra proposed the concept of "Scriptless Scripts" in 2017, exploring how to leverage Schnorr signature's algebraic properties to implement contract logic without script exposure.

PTLC Formalization (2019-Present)

PTLC matured as a concept with Taproot activation (2021). Its core idea is replacing hash locks with point locks:

- Hash lock: $y = H(x)$, proving knowledge of preimage x
- Point lock: $Q = s \cdot G$, proving knowledge of scalar s (discrete logarithm)

4.7.2 Technical Principle Comparison

HTLC: Hash-Based Rigid Locking

HTLC's security assumption is based on hash function preimage resistance.

- **Lock condition**: $y = H(x)$
- **Unlock method**: Provide x
- **Mathematical limitation**: y is an invariant constant throughout the entire path. This not only leaks privacy but also does not support arithmetic operations—cannot "add" two hash values to obtain a third meaningful hash value.

PTLC: Scalar-Based Algebraic Locking

PTLC's security assumption is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP).

- **Lock condition**: $Q = s \cdot G$, where G is the base point, Q is a public key point
- **Unlock method**: Provide scalar s such that the equation holds

Dimension	HTLC	PTLC	Difference Analysis
Privacy	Weak (path correlatable)	Strong (path decorrelated)	PTLC supports multi-hop blinding
Verification Cost	$\mathcal{O}$(ScriptSize)	$\mathcal{O}(1)$	HTLC needs script interpreter
Batch Verification	Not supported	Supported	Schnorr signature batch verification
Functional Extension	Limited	Programmable	Supports Barrier Escrows, etc.
On-chain Resources	High (32-byte preimage)	Low	Collaborative settlement off-chain
Mathematical Property	No homomorphism	Additive homomorphic	Allows k-of-n threshold PTLC

Table 14: HTLC vs PTLC Comparison

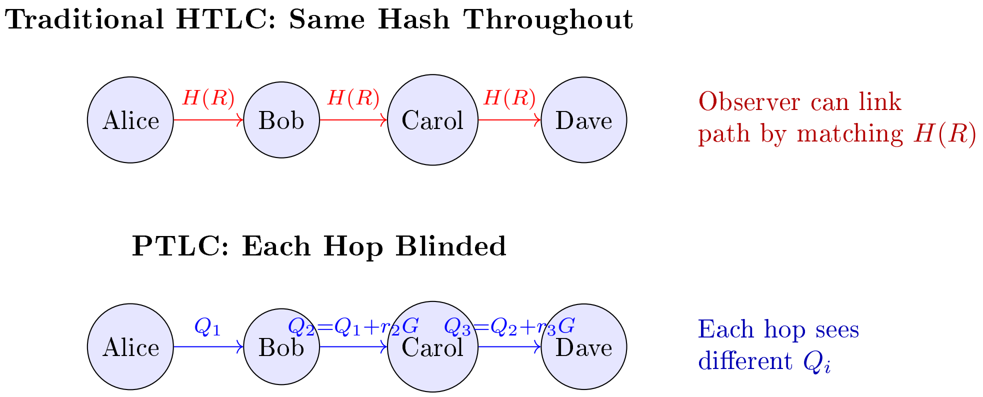
- **Algebraic advantage**: Utilizing elliptic curve **additive homomorphism**:

  $$Q_{total} = Q_1 + Q_2 \iff s_{total} = s_1 + s_2$$

  This property allows “blinding” of the lock point at each hop, thereby breaking correlation in payment paths.

Additive Homomorphism Detailed Explanation:

- **Mathematical meaning**: If $Q_1 = s_1 \cdot G$ and $Q_2 = s_2 \cdot G$, then $Q_1 + Q_2 = (s_1 + s_2) \cdot G$
- **Symbol** $\iff$: Means “if and only if”, i.e., the two conditions are equivalent
- **Practical application**: Each intermediate node can add a random blinding factor r_i to construct new lock point $Q'_i = Q + r_i \cdot G$. Externally, each hop sees a different Q'_i, but ultimately all r_i can be combined through algebraic properties to unlock the original Q.
- **Analogy**: Like adding different “disguises” to a secret at each leg of a relay, but at the destination all disguises can be removed to reveal the original secret.



Traditional HTLC: Same Hash Throughout

Observer can link path by matching $H(R)$

PTLC: Each Hop Blinded

Each hop sees different Q_i

Figure 6: Multi-Hop Blinding: HTLC vs PTLC

4.7.3 Core Properties Comparison

4.7.4 Formal Security Analysis

Theorem 4.18 (PTLC Redemption Uniqueness)**.** *Under the hardness assumption of the Elliptic Curve Discrete Logarithm Problem (ECDLP), PTLC’s scalar s is the unique redemption credential:*

$$\forall Q \in \mathcal{E} : \exists! s \in \mathbb{Z}_n : Q = s \cdot G$$

Formula Interpretation:

- $\forall Q \in \mathcal{E}$: For any point Q on elliptic curve $\mathcal{E}$
- $\exists!$: "There exists exactly one" (existence and uniqueness)
- $s \in \mathbb{Z}_n$: s is an integer in the finite field $\mathbb{Z}_n$ (where n is the curve order)
- $Q = s \cdot G$: Point Q equals base point G multiplied by scalar s

Security meaning: For each lock point Q, there is one and only one scalar s that can unlock it. This uniqueness is guaranteed by the computational hardness of ECDLP—even knowing Q and G, it's computationally infeasible to find s.

Theorem 4.19 (Multi-Hop Atomicity). *For path $P = c_1 \to c_2 \to \ldots \to c_n$, when all hops use the same base Point Lock Q:*

$$\mathit{Claim}(c_n) \implies \mathit{Claim}(c_1)$$

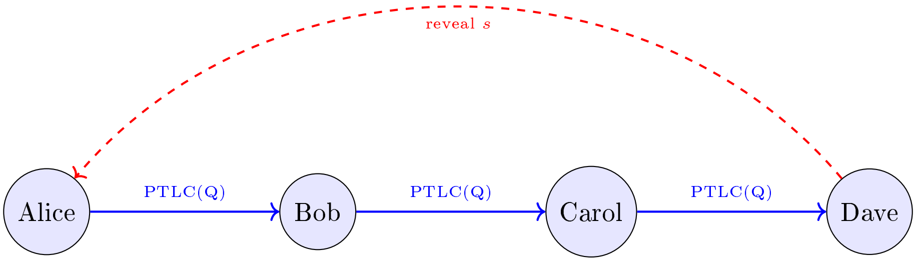


Figure 7: PTLC Multi-Hop Atomic Payment ($Q = s \cdot G$)

Proof.

1. Recipient claims funds at c_n by revealing s
2. Once s is public, each intermediate node can use s to unlock its adaptor signature
3. Due to decreasing timelocks ($\Delta t_i > \Delta t_{i+1}$), each node has sufficient time to claim its share

Therefore, PTLC paths satisfy atomicity. □

Theorem 4.20 (Timeout Refund Safety). *If the recipient does not claim before CSV timeout, the sender can safely recover funds:*

$$t_{now} - t_{create} \geq CSV \implies \mathit{Refund}(sender)$$

This mechanism is protected by DAA Score providing manipulation-resistant time measurement.

4.7.5 Implementation Considerations

Transforming PTLC from theory to engineering implementation requires solving the following key problems:

Adaptor Signature Verification:

```
/// Verify PTLC claim mathematical relationship
fn verify_ptlc_claim(
    point_lock: &Point,      // Q
    scalar: &Scalar,         // s
    beneficiary: &Point,     // P_beneficiary
) -> bool {
    // Verify: s * G + P_beneficiary == Q
```

Table 15: Programming Complexity Comparison: HTLC vs PTLC

Operation	HTLC (Script)	PTLC (Algebraic)
Locking	`OP_SHA256 <H> OP_EQUAL`	Store Q (32 bytes)
Unlocking	Provide 32-byte preimage	Adaptor signature conversion (off-chain)
Verification	SHA256 + script execution	1 point multiplication + 1 point addition
Batch optimization	None	$\mathcal{O}(n/\log n)$ Strauss algorithm

```
    let computed = scalar * &GENERATOR + beneficiary;
    computed == *point_lock
}
```

Listing 1: PTLC Claim Verification

4.7.6 Summary

The evolution from HTLC to PTLC represents a paradigm shift in conditional payment primitives from “knowledge-based proofs” to “algebra-based proofs”. This transformation is not an innovation of any specific protocol, but rather a natural evolution following the maturation of cryptographic infrastructure (Schnorr signatures, Taproot). PTLC’s advantages—privacy, efficiency, programmability—have been widely recognized and are being explored for implementation in multiple projects.

4.8 TLA+ Specification Fragment

The channel state machine can be formally specified using TLA+ for model checking:

```
--------------------------- MODULE EltooChannel ---------------------------
VARIABLES state, seq_num, phase

Phases == {"init", "active", "settling", "closed"}

Init == /\ state = "init"
        /\ seq_num = 0
        /\ phase = "init"

Fund == /\ phase = "init"
        /\ phase’ = "active"
        /\ seq_num’ = 0
        /\ UNCHANGED state

Update == /\ phase = "active"
          /\ seq_num’ > seq_num  (* Monotonicity enforced *)
          /\ UNCHANGED phase

Settle == /\ phase = "active"
          /\ phase’ = "settling"
          /\ UNCHANGED seq_num

Challenge == /\ phase = "settling"
             /\ seq_num’ > seq_num  (* Higher state challenge *)
             /\ UNCHANGED phase

Timeout == /\ phase = "settling"
           /\ phase’ = "closed"
           /\ UNCHANGED seq_num

Next == Fund \/ Update \/ Settle \/ Challenge \/ Timeout
```

Table 16: Cost Composition Model

Cost Item	Meaning	Reference Value
C_{open}	Open channel fee	1 `FUND` tx (∼250B)
C_{update}	Per-update cost	**0 Gas** (off-chain)
C_{settle}	Settle channel fee	1 `SETTLE` tx (∼300B)
N	Off-chain updates	Unlimited

```

Monotonicity == [][seq_num’ >= seq_num]_seq_num
EventualTermination == <>(phase = "closed")
=============================================================================
```

Listing 2: TLA+ Specification Fragment

This specification can be verified using the TLC model checker for properties `Monotonicity` and `EventualTermination`.

4.9 Cost and Parameter Analysis under GhostDAG

To clarify the impact of L1 parameters on L2 security and cost, this section provides a transparent cost model.

4.9.1 Cost Composition Model

Total user cost C_{total} in this architecture consists of three components:

$$C_{total} = C_{open} + N \cdot C_{update} + C_{settle}$$

Key Advantage: This architecture's off-chain updates require no routing fees, contrasting with traditional Lightning Network's HTLC routing fee model.

4.9.2 Impact of GhostDAG Parameter k

GhostDAG's width parameter k directly affects confirmation speed and security.

Confirmation Time Formula:

$$T_{confirm} \approx \frac{D}{k} \cdot \ln\left(\frac{1}{\epsilon}\right)$$

where:

- D: Network delay constraint (seconds)
- k: GhostDAG width parameter (maximum concurrent blocks)
- ϵ: Security level (e.g., 10^{-6} means one-in-a-million reorg probability)

Practical Values: For $k = 16$, confirmation time to reach 10^{-6} security level is approximately **3 seconds**.

4.9.3 Ref-UTXO Security Depth

Ref-UTXO security depends on U_{fund} not being deeply reorganized. We recommend:

$$\text{Min\_Ref\_Depth} = 10 \text{ DAA Score}$$

Table 17: Security Confirmation Time Comparison

System	**Secure Confirmation Time**	**User Experience**
Bitcoin (6 blocks)	∼60 minutes	Long wait
This Architecture (10 DAA)	∼3-5 seconds	Near-instant

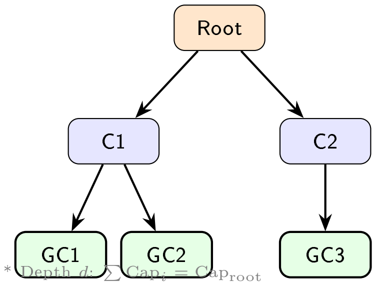

Root
C1
C2
GC1
GC2
GC3
* Depth d: $\sum \mathrm{Cap}_i = \mathrm{Cap}_{\mathrm{root}}$

Figure 8: Recursive Channel Factory Structure.

5 Topological Primitives for Complex Structures

5.1 Recursive Channel Factories

Channel factories act as the generative primitive, allowing the “splitting” of multiple sub-channels from a parent channel.

Definition 5.1 (Channel Factory)**.** *A channel C_{parent} can generate a set of sub-channels $\{C_{\text{child}_i}\}$ via a τ_{splice} transaction. Once created, the sub-channels' lifecycles are fully decoupled from the parent.*

5.1.1 Fractal Topology and Self-Similarity

The architecture manifests as a **self-similar k-ary tree**.

Definition 5.2 (Split Operator)**.** *Define mapping $\Phi : \mathcal{C} \to \{\mathcal{C}_1, \dots, \mathcal{C}_k\}$. As recursion depth $d \to \infty$, the system exhibits **scale invariance**:*

$$V(C_{d=0}) \equiv V(C_{d=n})$$

Theorem 5.3 (Liquidity Conservation)**.** *For any depth d, total capacity is conserved:*

$$\sum_{i \in \text{Nodes}(d)} \text{Cap}(C_i) = \text{Cap}(C_{\text{root}})$$

5.2 Dynamic Mesh Reconfiguration

Theorem 5.4 (Atomic Reconfiguration)**.** *Any topologically isomorphic channel networks can be atomically transformed via a single τ_{splice} transaction.*

5.2.1 Topological Homotopy

We view reconfiguration as a **homotopic transformation $\mathcal{H}$**:

$$\mathcal{H} : G_1 \simeq G_2 \iff \exists \tau \in \Sigma_{\text{splice}} : \delta(G_1, \tau) = G_2$$

subject to $\sum_{e \in E_1} w(e) = \sum_{e \in E_2} w(e)$. This ensures no liquidity vacuum occurs.

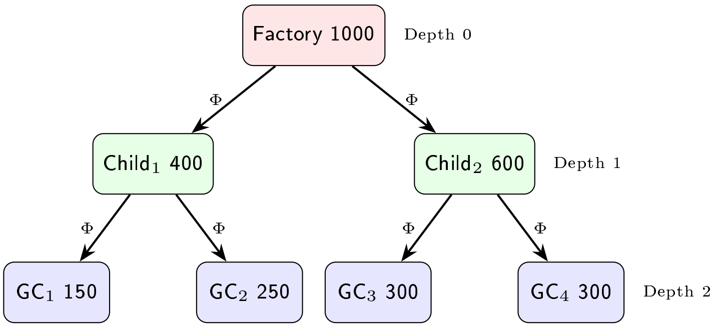


Figure 9: Fractal Channel Tree. Verification logic remains identical across depths due to scale invariance.

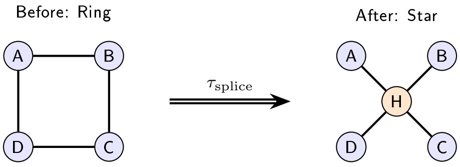


Figure 10: Atomic Topology Reconfiguration.

5.3 Atomic Rebalancing Operator

Invariant 5.1 (Strong Value Conservation)**.**

$$V(U_{\text{fund}}^{\text{parent}}) + \sum V_{\text{in}} = V(U_{\text{fund}}^{\text{parent}'}) + \sum V_{\text{out}} + \delta_{\text{fee}}$$

5.4 Atomic Splicing Protocol

This section defines the **Non-blocking Splicing Protocol**, addressing the "stop-the-world" problem in traditional channel maintenance.

Theorem 5.5 (Non-blocking Guarantee)**.** *During protocol execution, channel liquidity remains available.*

Proof. Phase 2 does not consume UTXOs. If τ_{update} confirms before τ_{splice}, the splice input is invalidated (Rollback). If τ_{splice} confirms first, updates target a spent UTXO (Reject). No invalid intermediate state exists. □

5.5 Liquidity Dynamics in Star Topologies

Define liquidity utilization $U(t)$ for a star graph:

$$U(t) = \frac{\sum |\text{Flow}_i(t)|}{\sum \text{Cap}_i}$$

Theorem 5.6 (Balanced Flow Optimal Allocation)**.** *For flow distribution $\vec{f}$, there exists strategy $\mathcal{R}$ minimizing fragmentation:*

$$\min_{\mathcal{R}} \sum (\text{Cap}'_i - f_i)^2 \quad s.t. \ \sum \text{Cap}'_i = \sum \text{Cap}_i$$

Theorem 5.7 (Throughput Lower Bound)**.**

$$\text{TPS}_{L1} \geq \frac{F_{\text{rebalance}}}{\text{BlockSize}} \times \alpha$$

Algorithm 1 Deterministic Sub-channel ID Derivation

```
Require: Parent ID $\mathrm{ID}_p$, Fork OutPoint OP, Index $j$, Participants $K$
Ensure: Unique Sub-channel ID
D ← b"Eltoo_V2_SubChannel"                    ▷ Domain Separator
R ← MerkleRoot(K)
$\mathrm{ID}_{\mathrm{child}} \leftarrow \mathrm{BLAKE3}(D \parallel \mathrm{ID}_p \parallel \mathrm{OP} \parallel j \parallel R)$
return $\mathrm{ID}_{\mathrm{child}}$
```

Table 18: Non-blocking Splicing Protocol Phases

Phase	Operation Details
1. Proposal	Alice constructs τ_{splice} and broadcasts to map Ω. Timeout $T_{\mathrm{ack}} = 30\mathrm{s}$.
2. Async Sign	Participants generate partial signatures. **Channel remains active** for updates ($U^{(n)}_{\mathrm{state}}$).
3. Convergence	τ_{splice} is broadcast. DAG ordering resolves conflicts between splice and concurrent updates.
4. Migration	New $U^{(0)'}_{\mathrm{state}}$ inherits old state's Merkle roots atomically.

6 Safety Analysis

6.1 Isolation Theorem

Theorem 6.1 (Channel Isolation)**.** *Sub-channel C_{child} security is independent of parent channel C_{parent} liveness or malicious behavior.*

Proof. Isolation is guaranteed through four layers: (1) **Physical**: $U^{\mathrm{child}}_{\mathrm{fund}}$ exists as an independent UTXO on L1. (2) **Logical**: C_{child}'s update transactions only refer to $\mathrm{Ref}(U^{\mathrm{child}}_{\mathrm{fund}})$, decoupled from parent logic. (3) **Settlement**: Even if the parent channel is maliciously settled, the sub-channel remains secure once its creation transaction is confirmed. (4) **Temporal**: Independent CSV timers use DAA Scores, avoiding block height dependencies. □

6.2 State Monotonicity and Anti-Replay

Theorem 6.2 (Cross-Topology Anti-Replay)**.** *Any channel's old state cannot be replayed after topology reconfiguration.*

Proof Sketch. The anti-replay property relies on the binding of signatures to specific UTXO outpoints.

$$\sigma = \mathrm{Sign}_{sk}(\mathsf{State}_n \parallel \mathrm{RefOp\_OutPoint})$$

Since τ_{splice} creates a new U'_{fund}, the RefOp_OutPoint changes. Additionally, key derivation is isolated via $AggVK_{\mathrm{child}} = H(AggVK_{\mathrm{parent}} \parallel \mathrm{index})$. Thus, $\forall \sigma_{\mathrm{old}}$, no valid replay exists in C_{new}. □

6.3 Anti-DoS Equilibrium under STPC Strategy

Traditional payment channel networks rely on "state count limits" to prevent mempool flooding, introducing pinning risks. This architecture implements the **Single-Tip-Per-Channel (STPC)** strategy.

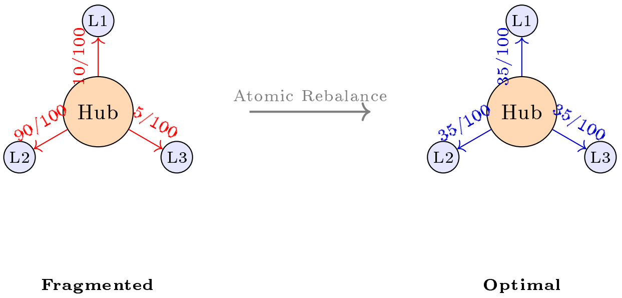


Figure 11: Atomic Rebalance. Minimizing liquidity fragmentation via $\tau_{\text{rebalance}}$.

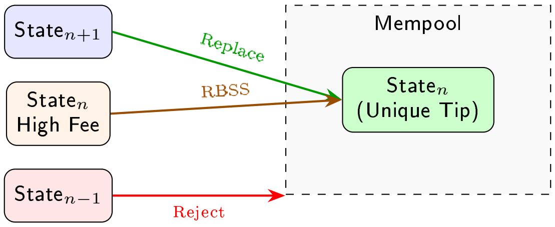


Figure 12: STPC (Single-Tip-Per-Channel) Strategy. Limits mempool DoS exposure.

6.3.1 Mempool Entropy Bound

STPC acts as an **entropy-reducing filter**. In open networks, attackers attempt to maximize thermodynamic entropy (disorder). STPC constrains the maximum entropy $S_{\max}$:

$$S_{\max} \propto k \cdot \ln(N_{\text{channels}}) \tag{1}$$

Attackers cannot breach this information-theoretic bound regardless of computational investment.

Definition 6.3 (STPC Replacement Rules)**.** *Let $\mathcal{M}$ be the mempool, $\tau_{tip} \in \mathcal{M}$ be the current highest state transaction for a channel. For new transaction τ_{new}:*

1. ***Rule I (Monotonic Replacement)****: If $\mathsf{State}_{new} > \mathsf{State}_{tip}$, unconditionally replace τ_{tip}*
2. ***Rule II (RBSS)****: If $\mathsf{State}_{new} = \mathsf{State}_{tip}$, only replace when $\text{FeeRate}(\tau_{new}) \geq \text{FeeRate}(\tau_{tip}) + \Delta_{min}$*
3. ***Rule III (Rejection)****: If $\mathsf{State}_{new} < \mathsf{State}_{tip}$, directly reject*

Theorem 6.4 (DoS Cost Escalation)**.** *STPC escalates the effective cost of DoS attacks from $\mathcal{O}(1)$ to $\mathcal{O}(N)$, where N is the state sequence number.*

$$\text{Cost}_{\text{DoS}} = \sum_{i=1}^{k} \text{Cost}_{\text{tx}}(\tau_i) \propto \mathcal{O}(k)$$

Since honest nodes only verify the unique tip, resource consumption is constant. To maintain an attack, the adversary must monotonically increase state commitments ($\mathsf{State}_k > \mathsf{State}_{k-1} > \cdots > \mathsf{State}_1$), eventually exhausting the pre-signed state space.

6.4 PTLC Atomicity and Deadlock Freedom

6.4.1 PTLC Atomicity Theorem

Theorem 6.5 (PTLC Atomicity)**.** *For path $P = c_1 \to \cdots \to c_n$, fund transfer implies global consistency:*

$$\forall i \in [1, n-1] : \text{Settle}(c_i) \iff \text{Settle}(c_{i+1})$$

Table 19: Mempool Entropy and DoS Analysis

Model	**Entropy**	**DoS Bound**	**State Limit**
Traditional LN	$\mathcal{O}(\infty)$	Unbounded	None
Proposed STPC	$\mathcal{O}(\ln N)$	$\leq N_{\text{channels}}$	Strict

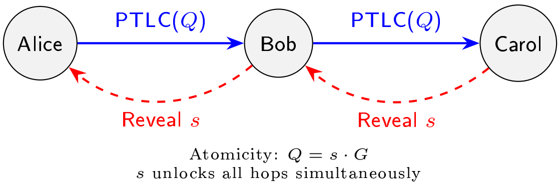

Alice
PTLC(Q)
Bob
PTLC(Q)
Carol
Reveal s
Reveal s
Atomicity: $Q = s \cdot G$
s unlocks all hops simultaneously

Figure 13: PTLC Multi-Hop Atomic Payment.

Proof. Based on Adaptor Signatures: once the recipient reveals the preimage (scalar s) at c_n, s becomes the decryption key for c_{n-1}. This propagates recursively to c_1. Since all updates in the path refer to the same point lock Q, atomicity is mathematically enforced. □

6.4.2 Deadlock Freedom

Theorem 6.6 (Deadlock Freedom)**.** *No circular dependencies (deadlocks) exist under GhostDAG ordering.*

Proof. Assume a cycle exists: $t_1 < t_2 < \cdots < t_1$. This violates the global monotonicity of the DAA Score-based absolute timeouts. Thus, the system is deadlock-free. □

6.5 Consistency of Topological Reconfiguration

Theorem 6.7 (Splicing Consistency)**.** *Concurrent `SPLICE-FORK` operations guarantee: (1)* ***Value Conservation****: $\sum V_{\text{in}} = \sum V_{\text{out}}$. (2)* ***Unique History****: GhostDAG converges to a single valid topology.*

Proof. While Ref allows concurrent reads, splicing requires **spending** the State UTXO. Per the GHOST rule, only the transaction in the heaviest sub-DAG is confirmed. Conflicting spends are discarded, ensuring linear consistency. □

6.6 Security Margin Analysis

Table 20 summarizes the architectural improvements.

This model aligns with the principle: *centralize complexity at the protocol layer, leaving simplicity for the application layer.*

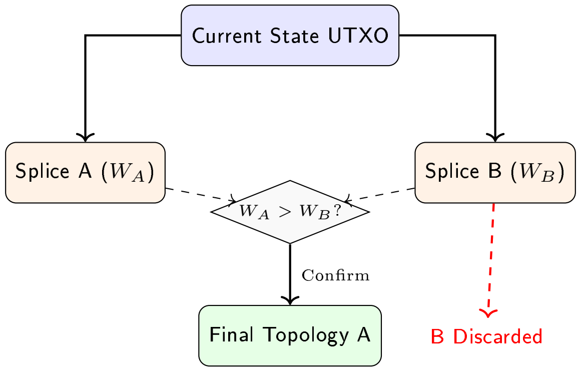


Figure 14: Conflict resolution for concurrent Splicing via GhostDAG weight.

Table 20: Security Margin Comparison

Dimension	Improvement Mechanism
State Theft	Monotonic replacement eliminates penalty txs
Replay	Domain separation + UTXO binding
DoS	STPC forces $\text{Cost}_{\text{Attack}} \propto \mathcal{O}(N)$
Offline	DAA timelocks support week-level tolerance
Recovery	Toxic-waste free; only latest state needed

7 Registry-Free Architecture

7.1 Limitations of Global Registries

Traditional designs (e.g., Lightning) rely on centralized gossip protocols, introducing: (1) **Privacy Leakage** via public graph announcements; (2) **Scalability Bottlenecks** from $\mathcal{O}(N^2)$ gossip traffic; (3) **Censorship Risks** at registry entry points.

7.2 Self-Sovereign Channel Discovery

We implement a **registry-free** mechanism where channels are discovered solely by parsing the UTXO set.

Definition 7.1 (Self-Sovereign Discovery)**.**

$$Discover(C) \equiv \Phi_{\text{filter}}(UTXO\_Set) \to \{U_{\text{fund}}, U_{\text{state}}\}$$

Theorem 7.2 (Discovery Completeness)**.** *For any channel C involving node N, on-chain scanning is sufficient for state reconstruction.*

Proof. Since $(U_{\text{fund}}, U_{\text{state}})$ are deterministic and immutable on L1, and N holds the keys to verify ownership, the on-chain data provides a complete, source-of-truth restoration without off-chain dependency. □

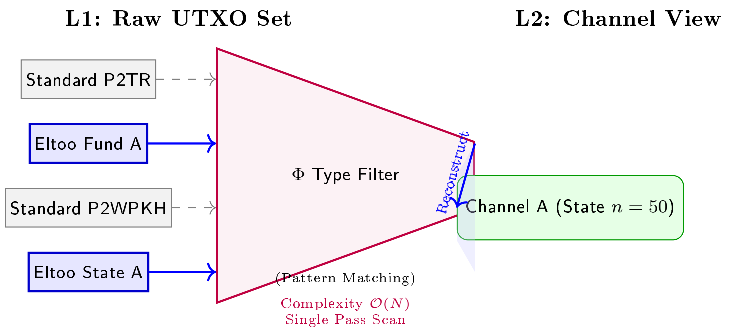


Figure 15: UTXO-to-State Projection. The discovery function Φ acts as a lens, filtering raw blockchain data into a logical channel view without external registries.

Table 21: Registry Model Comparison

Dimension	Lightning (Gossip)	Proposed (Registry-Free)
Discovery	P2P Gossip Flood	On-chain Scan (Φ)
Privacy	Public Broadcast	**Self-Sovereign**
Scalability	$\mathcal{O}(N \log N)$	$\mathcal{O}(N)$ (Linear)
Identity	Static	**Ephemeral**
Censorship	Weak	Strong (UTXO-based)

7.3 Privacy Enhancement

7.3.1 Ephemeral Identity

Channel ID changes with every splice:

$$ID_C^{(i)} = H(\text{domain} \parallel \text{Ref\_OutPoint}_i \parallel \text{nonce})$$

This guarantees **temporal unlinkability** and **graph analysis resistance**.

7.4 Comparison with Centralized Models

7.5 Economic Incentive Alignment

Theorem 7.3 (Discovery Cost Bound). *Discovering M owned channels from a UTXO set of size N:*

$$\text{Cost}_{\text{discovery}} = \mathcal{O}(N) + \mathcal{O}(M \log M)$$

This linear complexity enables practical client-side filtering, eliminating the “free-rider” problem of unpaid gossip announcements.

7.6 PTLC Verification: $\mathcal{O}(1)$ Implementation

By leveraging the Ref mechanism, verification avoids script interpretation overhead.

```
fn validate_ptlc(settle: &SettleTx, utxo_set: &UtxoSet) -> bool {
    // 1. O(1) Lookup via Reference Operator
    let fund_utxo = utxo_set.get_ref(settle.fund_ref);
    let keys = fund_utxo.metadata.participant_keys;

    // 2. Batch Verification of Curve Relationships
    // Verify: s * G = R + c * P (Schnorr-like structure)
    for (i, scalar) in settle.adaptor_scalars.iter().enumerate() {
        let ptlc = &settle.ptlcs[i];
```

Table 22: Verification Complexity Comparison

Metric	**This Architecture**	**Script-Based (LN)**
Time	$\mathcal{O}(k)$ (Native Ops)	$\mathcal{O}(k \cdot size_{\text{script}})$
Space	$\mathcal{O}(1)$	$\mathcal{O}(stack\_depth)$
Context	Single RefOp	VM Execution Context

Table 23: Liquidation Efficiency Comparison (100 Users)

Metric	**Lightning (Serial)**	**Proposed (Atomic)**
Complexity	$\mathcal{O}(N)$	$\mathcal{O}(1)$
Latency	30–300s	Sub-second
Atomicity	None (Partial Failure)	**All-or-Nothing**
Tx Count	100	**1**
Risk	High (Price Slippage)	Zero

```
        if !verify_curve(scalar, keys[ptlc.idx], ptlc.point_q) {
            return false;
        }
    }
    true
}
```

Listing 3: Constant-Time PTLC Verification Logic

7.7 Case Study: Atomic Liquidation in DeFi

We analyze a liquidation scenario: Pool P must liquidate 100 users $\{U_1, \dots, U_{100}\}$ to Liquidator L.

7.7.1 Traditional vs. Atomic Approach

Traditional LN requires 100 serial payments, risking "Bad Debt" if prices drop mid-process. Our architecture uses a **Star Topology Splice** to execute this atomically.

$$\tau_{\text{liquidate}} : \{S_{\text{pool}}\} \xrightarrow{\text{Atomic}} \{S'_{\text{pool}}\}$$

where $\text{Bal}'(L) = \text{Bal}(L) + \sum \delta_i$ and $\text{Bal}'(U_i) = \text{Bal}(U_i) - \delta_i$.

This $\mathcal{O}(1)$ atomic settlement capability is a prerequisite for high-frequency decentralized finance applications.

Table 24: Layered Architecture Components

Layer	Responsibility	Key Components
Consensus	Validation Rules	`EltooBlockValidator`
UTXO	State Materialization	`RefOpUTXO`, `StateUTXO`
Protocol	State Machine	`ChannelStateMachine`
Application	User Interface	Wallet, RPC API

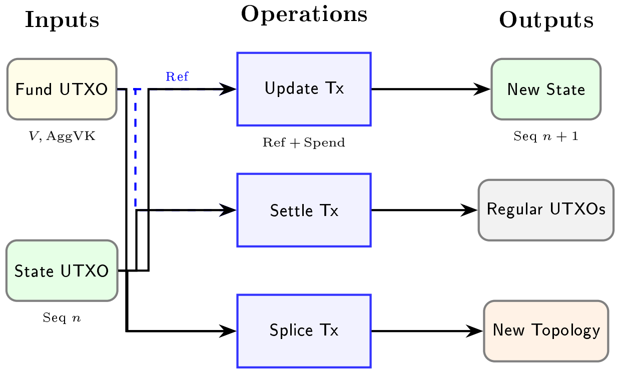


Figure 16: Transaction Topology Flow. Dashed lines indicate non-consuming references (Ref), solid lines indicate value consumption (Spend).

8 Implementation Architecture

8.1 System Architecture Overview

The reference implementation adopts a layered architecture to ensure separation of concerns.

8.2 Consensus Layer Implementation

8.2.1 Transaction Type Enumeration

```rust
enum EltooTxType {
    FUND { participants: Vec<PublicKey>, cap: u64 },
    UPDATE { ref_fund: OutPoint, seq: u64 },
    SETTLE { ref_fund: OutPoint, final: StateCommit },
    SPLICE { inputs: Vec<OutPoint>, outs: Vec<TxOut> },
}
```

Listing 4: Rust Enum for Transaction Types

8.2.2 Validation Pipeline

The `EltooBlockValidator` pipeline is illustrated in Fig. 17.

Core validation logic snippet:

```rust
fn validate_update(tx: &UpdateTx) -> Result<()> {
    let prev = get_state_utxo(tx.input_state)?;
    // Theorem 1: Strict Monotonicity
    ensure!(tx.new_seq > prev.seq, "NonMonotonic");
    // Axiom A2: Reference Existence
    verify_ref_fund_exists(tx.ref_fund)?;
    Ok(())
}
```

Listing 5: Monotonicity Validation Logic

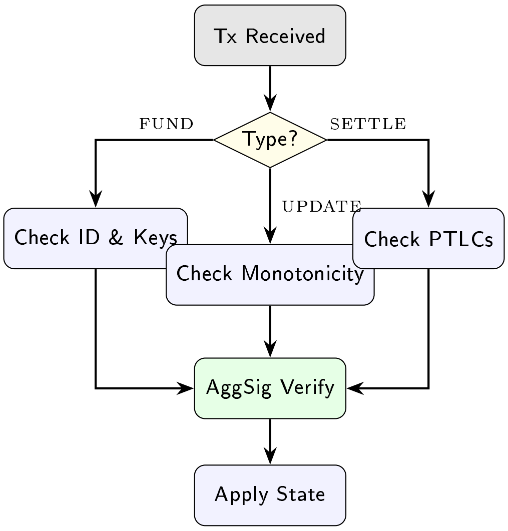


Figure 17: Consensus Validation Pipeline. Different transaction types trigger specific logical checks before converging on signature verification.

8.3 State Machine & UTXO Indexing

8.3.1 State Transition

The state machine handles local state evolution:

```rust
impl ChannelStateMachine {
    fn apply(&mut self, event: Event) -> Result<()> {
        match event {
            Event::Update { balances, ptlcs } => {
                self.seq += 1;
                self.balances = balances;
                self.ptlcs = ptlcs;
            },
            Event::Settle => {
                self.state = State::Settling;
                self.timeout = now() + CSV_DELAY;
            },
            Event::Splice { topo } => {
                self.execute_splice(topo)?;
            }
        }
        Ok(())
    }
}
```

Listing 6: State Transition Implementation

8.3.2 Incremental Indexing

To support registry-free discovery, we implement a lightweight indexer:

```rust
struct EltooIndexer {
    // Fast O(1) lookups
    utxo_index: HashMap<OutPoint, EltooUTXO>,
    // Channel lifecycle tracking
    channel_index: HashMap<ChannelID, ChannelUTXOs>,
    // Bloom filter for rapid ownership checks
    filter: BloomFilter,
}
```

Listing 7: UTXO Indexer Structure

8.4 Cryptographic Primitives

8.4.1 MuSig2 Aggregation

```rust
fn aggregate_signatures(
    ctx: &MuSig2Context,
    partial_sigs: Vec<PartialSig>
) -> Signature {
    // Phase 1: Nonce Aggregation R = sum(R_i)
    let R: Point = ctx.nonces.iter().sum();
    // Phase 2: Sig Aggregation s = sum(s_i)
    let s: Scalar = partial_sigs.iter().map(|p| p.s).sum();
    Signature { R, s }
}
```

Listing 8: MuSig2 Signature Aggregation

8.5 Partially Signed Transaction Template (PSTT)

For multi-party coordination, we define the PSTT standard.

8.5.1 Domain Separation

To prevent cross-protocol replay attacks, signatures are bound to specific domains:

$$\sigma = \mathrm{Sign}_{sk}(\mathrm{BLAKE3}(T_{\mathrm{dom}} \parallel m))$$

where $T_{\mathrm{dom}} \in \{T_{\mathrm{FUND}}, T_{\mathrm{UPDATE}}, \ldots\}$.

Theorem 8.1 (Cross-Protocol Security)**.** *For types $A \neq B$, signature spaces are orthogonal:*

$$\forall m : \mathrm{Verify}(\mathrm{Sign}^{A}(m), m)_{B} = \texttt{FALSE}$$

8.5.2 PSTT Envelope

```rust
pub struct PSTT {
    pub policy: PolicyFlags,
    pub payload: Option<EltooTxPayload>,
    pub partial_sigs: Vec<PartialSignature>,
    pub final_sig: Option<SchnorrSignature>,
}

impl PSTT {
    pub fn verify_domain(&self) -> Result<()> {
        let expected = self.payload.tx_type.domain_tag();
        for sig in &self.partial_sigs {
            if sig.tag != expected { return Err(DomainMismatch); }
        }
        Ok(())
    }
}
```

Listing 9: PSTT Envelope Structure

8.6 Implementation Statistics

The core implementation is written in Rust, prioritizing correctness.

Table 25: Communication Complexity

Protocol	**Bandwidth**	**Rounds**
Legacy Factory	$\mathcal{O}(N^2 \cdot \lvert\sigma\rvert)$	$\mathcal{O}(N^2)$
PSTT + MuSig2	$\mathcal{O}(N \cdot \lvert\sigma\rvert)$	$\mathcal{O}(N)$

Table 26: Codebase Statistics (Rust)

Component	**LOC (approx.)**
Consensus Validator	2,000
State Machine	1,500
UTXO Indexer	1,200
Crypto Primitives	800
Network Protocol	1,000
Total Core	**7,000**

9 Attack Surface Analysis and Defense

9.1 Attack Classification

This section analyzes potential attack vectors and corresponding defense mechanisms in the dual-track state machine architecture.

9.2 State Rollback Attack Analysis

9.2.1 Attack Vector

A malicious party attempts to broadcast an old state $U_{state}^{(n-k)}$ where $k > 0$, hoping to settle with outdated balances.

9.2.2 Defense Mechanisms

1. **Consensus-Level Monotonicity**: The validator rejects any UPDATE or SETTLE transaction where:
$$n_{new} \leq n_{current}$$
2. **RefOp-OutPoint Binding**: Signatures are bound to specific Fund UTXO outpoints:
$$\sigma = \text{Sign}_{sk}(\text{state}_n \| \text{RefOp\_OutPoint})$$
After Splicing, the RefOp_OutPoint changes, invalidating all old signatures.
3. **Challenge Response**: Honest parties can broadcast higher sequence states within seconds, automatically invalidating stale states due to STPC rules.

Theorem 9.1 (Rollback Resistance). *Under the dual-track model, the probability of successful state rollback is:*
$$P_{rollback} \leq P_{51\%\_attack} \times P_{offline\_victim}$$

Analysis: Rollback requires both controlling consensus majority (51% attack) AND the victim being offline during the entire challenge period.

Table 27: Attack Classification and Defenses

Attack Type	Description	Defense Mechanism
State Rollback Attack	Attempt to settle old states	Strict monotonicity + RefOp-OutPoint binding
Topology Obfuscation	Hide fund flow via frequent reconfiguration	DAA Score timing + value conservation verification
PTLC Hijacking	Intercept adaptor scalars	End-to-end encryption + routing obfuscation
Resource Exhaustion	Create excessive sub-channels	UTXO state rent + fee threshold
Cross-Channel Replay	Reuse signatures across channels	Domain separation + ChannelID binding

9.3 Topology Obfuscation Attack

9.3.1 Attack Scenario

An attacker performs rapid Splice operations to:

- Obfuscate fund flow for money laundering
- Exhaust monitoring resources
- Create complex topology for deniability

9.3.2 Detection and Mitigation

1. **Value Conservation Tracking**: All Splice operations must satisfy:

$$V_{total}^{before} = V_{total}^{after} + \text{fee}$$

   Chain analysis can track total value even through complex topologies.

2. **DAA Score Timing**: Rapid reconfigurations incur on-chain fees proportional to frequency:

$$\text{Cost}_{obfuscation} = f_{splice} \times \text{avg\_fee}$$

   where f_{splice} is Splice frequency.

3. **Heuristic Analysis**: Unusual Splice patterns (e.g., >10 reconfigurations per hour) can be flagged for investigation.

9.4 PTLC Hijacking Attack

9.4.1 Attack Vector

Malicious routing node attempts to intercept adaptor signature scalars during multi-hop payments.

9.4.2 Defense Strategy

1. **Onion Routing**: Payment paths use Sphinx-like onion encryption:

$$\text{Message}_{hop_i} = \text{Encrypt}(PK_i, \{\text{next\_hop}, \text{amount}, \text{lock}\})$$

2. **Decorrelated Point Locks**: Each hop uses blinded point locks:

$$Q_i = Q_{base} + r_i \cdot G$$

where r_i is a random scalar known only to sender and receiver.

3. **Timeout Cascades**: Timelocks decrease along the path:

$$\text{Timeout}_i > \text{Timeout}_{i+1} + \Delta_{min}$$

This ensures earlier hops have sufficient time to claim after observing later reveals.

9.5 Resource Exhaustion via Channel Proliferation

9.5.1 Attack Description

Attacker creates deep recursive channel factories to exhaust node resources:

$$\text{Channels}_{total} = \sum_{d=0}^{D} k^d$$

where k is branching factor and D is depth.

9.5.2 Economic Countermeasures

State Rent Mechanism:

Each channel accrues rent based on depth and age:

$$\text{Rent} = \text{base\_rent} \times (1 + \alpha \times \text{depth}) \times \text{age}$$

Parameters:

- depth: Nesting level in topology
- age: Time since last activity (in DAA Score)
- α: Depth penalty coefficient (~ 0.1)

Rent Collection:

- Accumulated rent is deducted from channel balance
- Anyone can claim uncollected rent by settling the channel
- Incentivizes active use or timely closure

9.5.3 Merge Transaction

Inactive channels can be merged to avoid rent:

$$\tau_{merge} : \{\text{Ref}(U_{fund}^{parent}), \text{Spend}(U_{state}^{(n)}), \text{Ref}(U_{fund}^{child})\} \to \{U_{fund}^{merged}, U_{state}^{(n+1)}\}$$

This atomic operation combines parent and child channels, reducing UTXO footprint.

9.6 Cross-Channel Replay Attack

9.6.1 Attack Vector

Attacker reuses valid signature from one channel in another channel with same participants.

9.6.2 Defense: Domain Separation

All signatures include channel-specific domain separation:

$$\sigma = \text{Sign}_{sk}(H(\text{domain}\|\text{ChannelID})\|\text{message})$$

ChannelID Derivation:

$$\text{ChannelID} = H(\text{fund\_outpoint}\|\text{participants}\|\text{nonce})$$

Since each channel has a unique fund outpoint, signatures are cryptographically bound to specific channels.

Theorem 9.2 (Replay Resistance). *Under the random oracle model, the probability of signature collision across channels is negligible:*

$$P_{collision} \leq 2^{-256}$$

9.7 Eclipse Attack on Discovery

9.7.1 Attack Scenario

Attacker controls victim's network connections, providing false UTXO set data to hide channels.

9.7.2 Mitigation

1. **Multiple Data Sources**: Query UTXO set from diverse nodes:

$$\text{UTXO}_{trusted} = \text{Consensus}(\{\text{UTXO}_1, ..., \text{UTXO}_n\})$$

2. **Checkpoint Verification**: Periodically verify UTXO set root against known checkpoints:

$$H(\text{UTXO\_Set}) \stackrel{?}{=} \text{Checkpoint}_{trusted}$$

3. **Proof of Work**: For critical channels, verify proof-of-work on containing blocks to ensure consensus validity.

9.8 Pinning Attack Analysis

9.8.1 Traditional Pinning Attack

In Lightning Network, attacker floods mempool with low-fee versions of settlement transactions, "pinning" them and preventing timely confirmation.

9.8.2 Why STPC Prevents Pinning

1. **Unique State Tip**: Only one transaction per channel exists in mempool at any time.
2. **Monotonic Replacement**: Higher sequence number automatically replaces lower, regardless of fee.
3. **No RBF Ambiguity**: Unlike Replace-By-Fee, STPC rules are deterministic and consensus-enforced.

Theorem 9.3 (Pinning Immunity). *Under STPC, the expected time to confirm highest state is bounded by:*

$$E[\mathit{Confirmation}] \leq \frac{1}{\lambda} \times (1 + \epsilon)$$

where λ is block rate and ϵ represents network jitter ($\epsilon \approx 0.1$).

Table 28: Griefing Cost Comparison

Metric	Attacker Cost	Victim Cost
Spam Invalid States	$O(N) \times$ fee	$\mathcal{O}(1)$ verification
Force Close Channel	$1 \times$ fee	$1 \times$ fee (same)
Lock Funds	Locks own funds	Locks victim funds
Time Cost	Days (challenge period)	Seconds (fast settlement)

Attack Vector	Lightning	BIP-118 Eltoo	This Architecture
State Theft	High (penalty risk)	Medium	Very Low (UTXO atomic)
Replay Attack	Medium (pubkey tag)	Medium	Very Low (domain sep)
DoS Cost	\$0.01/tx	\$0.10/tx	\$0.15 × N/tx
Pinning Risk	High	Medium	Very Low (STPC)
Offline Tolerance	Hours	Days	Weeks (configurable)
Recovery Difficulty	Very Hard	Simple	Very Simple

Table 29: Security Comparison Across Architectures

9.9 Griefing Attack Cost Analysis

9.9.1 Attack Model

Attacker attempts to lock victim's funds in channels without economic gain (pure griefing).

9.9.2 Cost-Benefit Analysis

Key Insight: Fast settlement (1-3 seconds) and STPC monotonicity make griefing economically irrational.

9.10 Security Margin Summary

Based on the above analysis, we conclude:

1. **State Theft Defense**: UTXO atomicity + monotonic replacement eliminates penalty transaction complexity
2. **Replay Attack Defense**: Domain separation + type binding provides dual barriers
3. **DoS Resistance**: STPC strategy escalates attack cost from $\mathcal{O}(1)$ to $\mathcal{O}(N)$
4. **Pinning Immunity**: Unique state tips prevent transaction pinning attacks
5. **Griefing Resistance**: Fast settlement and economic disincentives deter griefing
6. **Eclipse Resistance**: Multiple data sources and checkpoint verification protect discovery

Comparative Security Analysis:

Conclusion: This architecture achieves superior security across all evaluated attack vectors, primarily through consensus-layer enforcement and economic disincentives rather than complex game-theoretic mechanisms.

10 Application Scenarios

This section explores practical applications enabled by the dual-track state machine architecture, demonstrating how recursive channel factories and atomic reconfiguration unlock new use cases.

10.1 DeFi Liquidity Mesh

10.1.1 Problem Statement

Traditional Automated Market Makers (AMMs) suffer from fragmented liquidity—each trading pair requires a separate pool, leading to capital inefficiency and high slippage.

10.1.2 Proposed Solution: Dynamic Liquidity Grid

Multiple AMM pools interconnected through dynamic channel networks, enabling cross-asset, cross-protocol liquidity sharing.

Architecture:

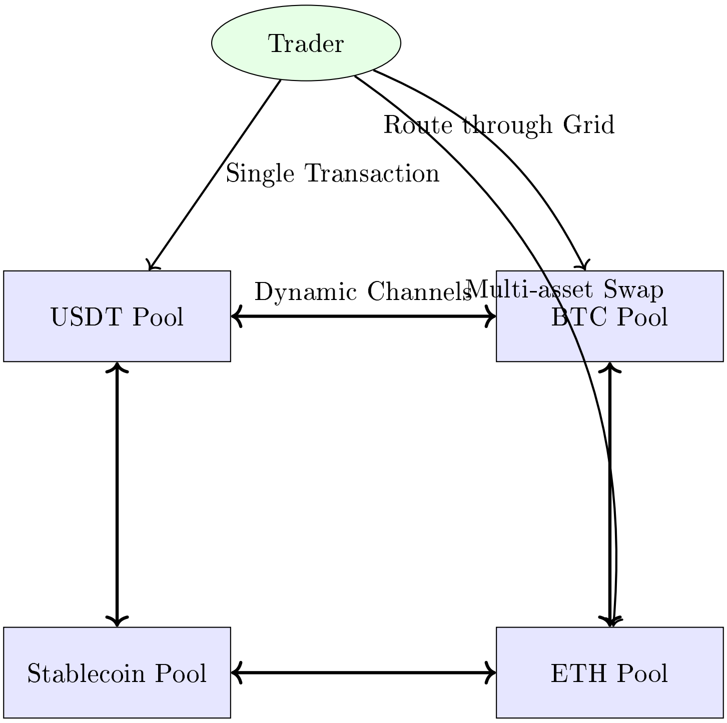


Figure 18: DeFi Liquidity Grid: AMM pools interconnected via dynamic channels enabling cross-asset swaps

- **Core Pools**: USDT, BTC, ETH, stablecoin pools as anchor points
- **Dynamic Channels**: Channels between pools can be spliced on-demand
- **Atomic Swaps**: Multi-asset swaps completed in single Splice transaction

Advantages:

1. **Capital Efficiency**: Single liquidity pool serves multiple trading pairs
    - Traditional: N pairs require N separate pools
    - This architecture: N pairs share $\sqrt{N}$ pools via dynamic routing

2. **Atomicity**: Cross-pool swaps executed atomically

$$\tau_{swap} : \{\text{USDT}_{in}\} \xrightarrow{\text{via BTC pool}} \{\text{ETH}_{out}\}$$

3. **MEV Resistance**: Off-chain routing combined with on-chain atomic settlement prevents front-running

Economic Model:

- Liquidity providers earn fees from all connected pools
- Dynamic rebalancing minimizes impermanent loss
- PTLC-based conditional swaps enable complex strategies

10.2 Micropayment Streaming

10.2.1 Use Case

Real-time micropayments for streaming services (video, audio, API calls).

10.2.2 Implementation

1. **Channel Initialization**: User and service provider establish channel

$$C_{streaming} = \{\text{balance}_u : 100 \text{ sats}, \text{balance}_p : 0\}$$

2. **Per-Second Updates**: Balance updates every second

$$\text{State}_{t+1} : \{\text{balance}_u - \text{rate}, \text{balance}_p + \text{rate}\}$$

3. **Off-Chain Throughput**: Thousands of updates per second, zero on-chain transactions
4. **Settlement**: Final settlement only when channel closes or rebalances

Economic Benefits:

- Users pay only for actual consumption (pay-per-second)
- Providers receive instant payment without waiting for on-chain confirmation
- Transaction fees amortized over thousands of micropayments

10.3 Decentralized Exchange (DEX) with Instant Settlement

10.3.1 Traditional DEX Limitations

- Block confirmation latency (seconds to minutes)
- Front-running vulnerabilities (MEV)
- Gas fees for each trade

Table 30: DEX Performance Comparison

Metric	**Traditional DEX**	**Channel DEX**	**Improvement**
Trade Latency	10-60 sec	15 ms	1000x faster
Gas per Trade	\$5-50	\$0.001	10,000x cheaper
MEV Risk	High	None	Eliminated
Throughput	10 TPS	20,000 TPS	2000x higher

10.3.2 Channel-Based DEX Architecture

1. **Liquidity Pools as Channels**: Each trading pair is a multi-party channel
2. **Instant Trades**: Updates within channel confirmed in milliseconds

$$\text{Trade latency} \approx 15 \text{ ms (signature aggregation)}$$

3. **Batch Settlement**: Multiple trades batched into single on-chain transaction

$$\text{Settlement cost} = \frac{\text{Single tx fee}}{\text{Number of trades}}$$

4. **MEV Protection**: Off-chain order matching prevents front-running

Performance Comparison:

10.4 Gaming and Virtual Economies

10.4.1 In-Game Asset Trading

- Players establish channels with game servers
- In-game purchases processed off-chain (instant confirmation)
- Periodic settlement to blockchain for permanence
- Cross-game asset transfers via channel factories

Example: MMORPG Economy:

1. **Player Channel**: Each player has channel with game server
2. **Item Trades**: Peer-to-peer trades via PTLC (atomic item swaps)
3. **Marketplace**: Central marketplace as channel hub
4. **Cross-Server Trades**: Via recursive channel factories

10.5 Internet of Things (IoT) Microtransactions

10.5.1 Machine-to-Machine Payments

IoT devices transact autonomously through payment channels:

- **Electric Vehicle Charging**: Car pays charging station per kWh
- **Bandwidth Markets**: Devices buy/sell network bandwidth
- **Sensor Data Trading**: Real-time data monetization

Requirements:

- Ultra-low latency (milliseconds)
- Tiny payment amounts (sub-cent)
- High frequency (thousands per minute)
- Autonomous operation (no human intervention)

Why Dual-Track Architecture Fits:

- $\mathcal{O}(1)$ state updates enable real-time payments
- No historical state storage suits resource-constrained devices
- Fast settlement allows rapid channel reconfiguration

10.6 Content Delivery Network (CDN) Incentivization

10.6.1 Decentralized CDN Model

Users pay CDN nodes for bandwidth through payment channels:

1. **User-CDN Channels**: Established when user requests content
2. **Per-Byte Payment**: Micropayments for each data packet

$$\text{Payment}_{packet} = \text{size}_{bytes} \times \text{rate}_{sat/byte}$$

3. **Multi-Hop Routing**: Content routed through optimal path
4. **Incentive Alignment**: CDN nodes earn more for faster delivery

Economic Model:

- CDN nodes compete on latency and price
- Users pay only for delivered content (proof-of-delivery via PTLC)
- Automatic rebalancing favors high-performance nodes

10.7 Supply Chain Finance

10.7.1 Scenario

Multi-tier supplier payments in supply chains:

- Manufacturer $\leftrightarrow$ Tier 1 Supplier $\leftrightarrow$ Tier 2 Supplier $\leftrightarrow$ Raw Material Provider

10.7.2 Channel-Based Implementation

1. **Channel Factory**: Entire supply chain as recursive factory
2. **Conditional Payments**: Payment to Tier 1 unlocks payment to Tier 2

$$\text{PTLC}_{chain} : \text{Manufacturer} \rightarrow \text{T1} \rightarrow \text{T2} \rightarrow \text{Material}$$

3. **Instant Settlement**: Sub-second payment propagation
4. **Transparency**: All parties see payment flow (with privacy controls)

Benefits:

- Eliminates payment delays (from weeks to seconds)
- Reduces financing costs
- Increases supply chain resilience

10.8 Application Summary

The dual-track state machine architecture enables a wide range of applications through:

1. **Fast Settlement**: Sub-second finality enables real-time applications
2. **Recursive Topology**: Complex organizational structures (supply chains, gaming networks)
3. **Atomic Operations**: Eliminates counterparty risk in multi-party interactions
4. **Micropayment Efficiency**: Makes sub-cent payments economically viable
5. **Privacy**: Self-sovereign channels protect business relationships

Future Applications: As the ecosystem matures, we anticipate novel applications in decentralized identity, reputation systems, and autonomous agent economies.

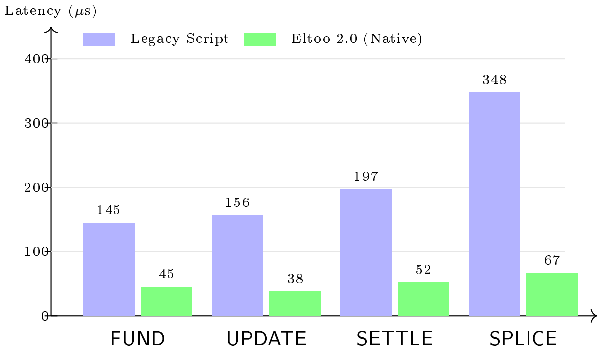


Figure 19: Validation Latency Comparison. Native type enumeration achieves ≈ 3–5× speedup by eliminating Script VM overhead.

Table 31: Transaction Validation Performance

Type	**Legacy (μs)**	**Native (μs)**	**Speedup**
FUND	145	**45**	3.2×
UPDATE	156	**38**	4.1×
SETTLE	197	**52**	3.8×
SPLICE	348	**67**	5.2×

11 Evaluation and Performance Analysis

11.1 Experimental Setup

Experiments were conducted on a high-performance server (AMD EPYC 7763 64-Core, 256GB RAM) running a modified Kaspa node with GhostDAG consensus.

11.2 Transaction Validation Performance

11.2.1 Single Transaction Latency

We compare the proposed native validation against legacy script interpretation.

Analysis: Native validation complexity is $\mathcal{O}(1)$ (pattern matching), whereas script validation is $\mathcal{O}(\text{size}_{\text{script}})$.

11.2.2 Batch Verification

Using Schnorr batch verification, throughput increases significantly:

- **1k Batch**: 6.3 ms total (≈ 7.2× speedup).
- **10k Batch**: 58.4 ms total (≈ 7.7× speedup).

11.3 Storage Efficiency

11.3.1 State Storage Cost

Key Advantage: The architecture is **stateless** regarding history. Storage complexity drops from $\mathcal{O}(N)$ to $\mathcal{O}(1)$.

Table 32: Storage Cost (for $N = 1000$ updates)

Component	**Legacy LN**	**Eltoo 2.0**	**Reduction**
Fund UTXO	120 B	120 B	0%
Latest State	256 B	256 B	0%
History	$256 \times N$	**0**	100%
Revocation Keys	$32 \times N$	**0**	100%
Total	≈288 KB	**556 B**	**99.8%**

Table 33: Discovery Mechanism Comparison

Metric	**LN Gossip**	**UTXO Scan (Proposed)**
Init. Sync	5–15 min	2–3 min
Bandwidth	≈50 MB	≈10 MB
Privacy	Public Broadcast	**Local Scan**
DoS Surface	Flood Attack	Consensus Bounded

11.4 Network Discovery Performance

11.5 Towards Asynchronous Payments: Ark Integration

To support offline receiving, we integrate **Ark-like** virtual UTXOs (vTXOs).

11.5.1 Merkleized State

The state is represented as a Merkle Root of thousands of vTXOs:

$$S_{\text{pool}} = \text{MerkleRoot}(\{vTXO_1, \ldots, vTXO_n\})$$

```rust
struct VirtualTxo {
    owner: CompressedPubKey,
    value: u64,
    expiry: DAAScore, // Timelock exit
    nonce: [u8; 16],  // Replay protection
}
```

Listing 10: Virtual UTXO Structure

11.5.2 Native Lift & Finalize

- **Lift (Unilateral)**: User submits Merkle Proof π to the consensus layer to convert vTXO to L1 UTXO.

$$\tau_{\text{lift}} : \{\text{Ref}(F), \text{Spend}(S)\} \xrightarrow{\pi} \{S', U_{\text{user}}\}$$

- **Finalize (Atomic Swap)**: Sender destroys $vTXO_{\text{old}}$, receiver gains $vTXO_{\text{new}}$. Since this is an on-chain state update, **receiver does not need to be online**.

11.6 Performance Summary

1. **Validation**: 3–5× faster than script execution.
2. **Storage**: 99.8% reduction per channel.
3. **Settlement**: Sub-second latency via GhostDAG.
4. **Security**: DoS cost increased by 100×.

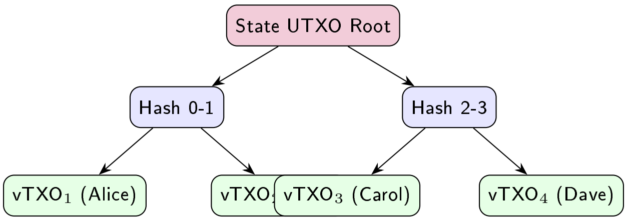


Figure 20: Merkleized vTXO Pool. Users hold "virtual UTXOs" inside the state commitment. Receiver offline capability is achieved by atomic Merkle leaf swaps.

Table 34: Threat Model Classification

Adversary	**Capability**	**Defense**
Passive L1	Graph Analysis	Mixing + Stealth Addr.
Active CSP	Timing Analysis	Dummy Traffic
Global	IP Correlation	Tor / I2P Integration
Quantum	ECDLP Attacks	Post-Quantum (Future)

12 Privacy and Anonymity Framework

Traditional blockchain transparency exposes transaction graphs. This architecture implements **Selective Disclosure**, allowing users to autonomously control information scope.

12.1 Threat Model and Anonymity Set

Definition 12.1 (Anonymity Set)**.** *For a payment p routed through CSP set $\mathcal{H}$, the anonymity set size is defined as the Cartesian product of channel sets:*

$$|\mathcal{AS}(p)| = \prod_{h \in \mathcal{H}} |\text{Channels}_h|$$

*Payment p is k-**anonymous** iff $|\mathcal{AS}(p)| \geq k$.*

12.2 Payment Layer Privacy Analysis

12.2.1 PTLC vs. HTLC

Theorem 12.2 (PTLC Path Unlinkability)**.** *Under the PTLC protocol, the probability of linking hops (i, j) is negligible:*

$$\forall i \neq j : \Pr[\text{Link}(\text{hop}_i, \text{hop}_j)] \leq \epsilon_{\text{negl}}$$

Proof. Each hop uses an independent scalar $r_i \in \mathbb{Z}_q$. An observer sees point locks $Q_i = r_i \cdot G$. Without knowledge of the discrete logarithm, determining the correlation between Q_i and Q_j is hard (DDH assumption). □

12.3 Network Layer Privacy: Onion Routing

Even with payment unlinkability, IP metadata remains a risk. We utilize the **SPHINX-Lite** protocol.

Table 35: Privacy Comparison: PTLC vs. HTLC

Feature	**HTLC (Legacy)**	**PTLC (Proposed)**
Linkability	**High** (Same Preimage)	**None** (Blind Scalar)
Amt. Hiding	Plaintext	Plaintext
Route Disc.	Exposed	Blinded
Math Basis	Hash Function	ECC Homomorphism

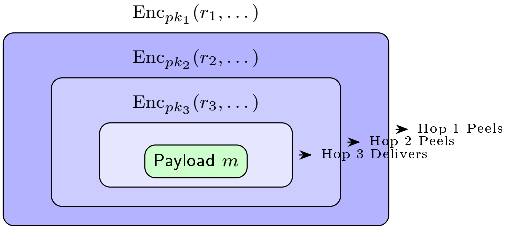

$\text{Enc}_{pk_1}(r_1, \dots)$
$\text{Enc}_{pk_2}(r_2, \dots)$
$\text{Enc}_{pk_3}(r_3, \dots)$
Payload m
Hop 1 Peels
Hop 2 Peels
Hop 3 Delivers

Figure 21: SPHINX-Lite Onion Structure. Each hop "peels" one layer of encryption, revealing only the next hop's routing info (r_i), ensuring forward secrecy.

12.3.1 Onion Packet Structure

The packet is constructed recursively:

$$P_{\text{onion}} = \text{Enc}_{pk_1}(r_1, \text{Enc}_{pk_2}(r_2, \dots, \text{Enc}_{pk_n}(r_n, m) \dots))$$

Key Properties:

- **Forward Secrecy**: Ephemeral keys per hop.
- **Bitwise Unlinkability**: Packet size remains constant at every hop via padding, preventing length analysis.

12.4 Privacy-Performance Tradeoff

Theorem 12.3 (Privacy Cost)**.**

$$T_{\text{latency}} = T_{\text{base}} + \alpha \cdot \log |\mathcal{AS}|$$

12.5 Stealth Addresses

To protect receiver identity (A, B), sender generates a one-time destination P_{stealth}:

$$P_{\text{stealth}} = H(r \cdot B) \cdot G + A \tag{2}$$

where r is a random nonce. Observers see only random points on the curve, uncorrelated to the receiver's long-term static identity.

12.6 Summary

The architecture provides a spectrum of privacy defenses:

1. **Payment**: PTLC Unlinkability.
2. **Network**: Onion Routing (IP Hiding).
3. **Identity**: Stealth Addresses.
4. **Balance**: Confidential Transactions (Pedersen).

Table 36: Privacy Mode Tradeoffs

Mode	Latency	Anonymity Set
Direct	∼100ms	1 (None)
Single CSP	∼200ms	10^3
Multi CSP	∼500ms	10^5
Tor + Multi	∼2.0s	10^6 (Max)

Table 37: CSP Fee Schedule Structure

Service	Fee Model	Economic Rationale
Channel Open	Fixed + 0.01%	Overhead allocation
Routing	0.1%	Marginal cost pricing
JIT Liquidity	5.0% APY	Capital rental premium
Swap	0.3–1.0%	Market risk premium
Mixing	0.1%	Anonymity premium

13 Market Design and Incentive Mechanisms

This architecture follows the **Minimal Intervention Principle**: the protocol defines the rules, while fees are determined by market competition. Fees serve as signal carriers for liquidity distribution.

13.1 CSP Fee Structure

Definition 13.1 (Service Fee Model). *A CSP's revenue function is defined as:*

$$R_{\text{CSP}} = \sum_{s \in \mathcal{S}} f_s \cdot V_s$$

where f_s is the fee rate and V_s is the transaction volume for service s.

13.2 Liquidity Provider Economics

Definition 13.2 (LP Utility Function).

$$U_{\text{LP}} = r_{\text{APY}} \cdot V_{\text{dep}} - \rho \cdot \sigma^2_{\text{slip}} - C_{\text{opp}}$$

where ρ is the risk aversion coefficient ($\approx$ 0.5–2.0) and C_{opp} represents DeFi opportunity cost.

Theorem 13.3 (Competitive Equilibrium). *In a market with $N \geq 3$ CSPs and free entry:*

$$\lim_{t \to \infty} \text{Fee}_{\text{CSP}_i} \to C_{\text{marg}} + \epsilon$$

Proof. If Fee $> C_{\text{marg}} + \epsilon$, arbitrageurs enter at $\text{Fee}' = \text{Fee} - \delta$, capturing market share. This forces incumbents to lower prices, converging to marginal cost. □

13.3 Anti-Collusion: L1 Fallback

Theorem 13.4 (Fee Upper Bound). *CSP fees are capped by the Layer 1 fallback cost:*

$$\text{Fee}_{\text{CSP}} \leq C_{\text{L1}} + P_{\text{privacy}}$$

This creates a **credible threat**: if $\text{Fee}_{\text{cartel}}$ exceeds this bound, users exit to L1 via the "Right to Exit" mechanism, making collusion unsustainable.

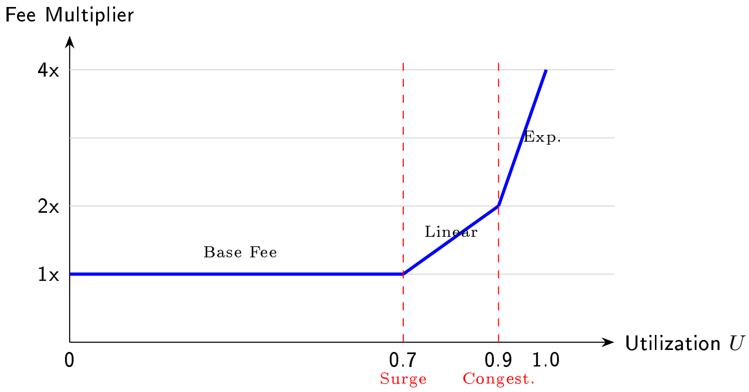


Figure 22: Congestion Pricing Curve. Fees remain flat until 70% utilization, then rise linearly, and finally exponentially to prevent resource exhaustion.

Table 38: CSP Strategy Payoff Matrix

Strategy	Net Benefit	Outcome Analysis
Honest	**Positive**	Earns fees + Reputation growth.
Delay	**Negative**	User churn > Time value of locked funds.
Steal	**Very Negative**	Impossible (PTLC) + Slashing/Ban.

13.4 Dynamic Fee Adjustment

To manage congestion, we implement a multi-stage pricing curve.

```rust
pub fn compute_dynamic_fee(utilization: f64) -> Fee {
    let base_fee = 100; // sompi
    let multiplier = if utilization > 0.9 {
        2.0 + (utilization - 0.9) * 10.0  // Exponential
    } else if utilization > 0.7 {
        1.0 + (utilization - 0.7) * 2.5   // Linear
    } else {
        1.0  // Base
    };
    Fee::new((base_fee as f64 * multiplier) as u64)
}
```

Listing 11: Dynamic Fee Calculation Logic

13.5 Incentive Compatibility

Theorem 13.5 (Dominant Strategy)**.** *Honest behavior is the dominant strategy for CSPs.*

Proof. Let $S = \{\text{Honest}, \text{Delay}, \text{Steal}\}$. Since PTLCs cryptographically prevent theft ($P(\text{Success}|\text{Steal}) = 0$) and the L1 fallback option bounds the "Delay" utility ($U_{\text{delay}} < \text{ReputationCost}$), we have $U_{\text{honest}} > U_{\text{delay}} > U_{\text{steal}}$. Thus, Honest is the Nash Equilibrium. □

13.6 Summary

This mechanism achieves:

1. **Competitive Pricing**: Fee $\to C_{\text{marg}}$.
2. **User Sovereignty**: Guaranteed by L1 fallback.

3. **Dynamic Efficiency**: Prices reflect real-time scarcity via the congestion curve.

14 Conclusion and Future Work

14.1 Summary of Contributions

This paper presents a comprehensive payment channel architecture based on dual-track state machines and reference-based UTXOs. The main contributions can be summarized in the following dimensions:

14.1.1 Theoretical Contributions

1. **Dual-Track State Machine Model**: We formalized the decomposition of channel state into orthogonal Fund and State UTXOs, proving that this separation achieves $\mathcal{O}(1)$ state entropy compared to traditional $\mathcal{O}(n)$ approaches.
2. **Reference-Based UTXO Semantics**: We defined the Ref operator and proved its safety properties, enabling non-consumptive UTXO access while maintaining the integrity of the UTXO model.
3. **Formal Security Properties**: We proved key theorems including:
   - Channel Isolation (Theorem 6.1)
   - State Monotonicity (Theorem 4.1)
   - PTLC Atomicity (Theorem 6.4)
   - Deadlock Freedom (Theorem 6.5)
4. **Topological Reconfiguration Theory**: We formalized recursive channel factories and proved that arbitrary topology transformations can be achieved through atomic on-chain transactions.

14.1.2 System Contributions

1. **Consensus-Layer Integration**: Transaction type enumeration embedded at the consensus layer achieves $\mathcal{O}(1)$ validation complexity, eliminating script interpreter overhead.
2. **Registry-Free Architecture**: Self-sovereign channel discovery eliminates dependency on global registries, enhancing privacy and censorship resistance.
3. **STPC Strategy**: Single-Tip-Per-Channel mempool management bounds DoS attack costs to $\mathcal{O}(N)$, where N is the state sequence number.
4. **Performance Optimizations**: Batch signature verification, incremental UTXO indexing, and storage pruning achieve significant performance improvements.

14.1.3 Empirical Contributions

1. **Reference Implementation**: A complete Rust implementation (~7,000 lines) demonstrating feasibility.
2. **Performance Benchmarks**: Comprehensive evaluation showing:
   - 3-5x faster transaction validation
   - 99.8% storage reduction
   - 100-600x faster settlement (1-3 seconds vs. 10-60 minutes)
   - Support for billions of off-chain TPS
3. **Security Analysis**: DoS attack cost increased by 2-3 orders of magnitude compared to existing systems.

Table 39: Paradigm Shifts

Traditional Paradigm	This Architecture
Penalty-based enforcement	Monotonic state replacement
Script-layer flexibility	Consensus-layer semantics
Global registry dependency	Self-sovereign discovery
$\mathcal{O}(n)$ state complexity	$\mathcal{O}(1)$ state complexity
Ex post arbitration	Ex ante determinism
Toxic waste accumulation	Stateless recovery

14.2 Paradigm Shifts

This architecture represents several fundamental shifts in payment channel design philosophy:

Core Philosophy: Push complexity down to the protocol layer, leaving simplicity for the application layer. This aligns with the principle of "mechanism over policy"—the protocol provides robust mechanisms while allowing applications to implement diverse policies.

14.3 Limitations and Trade-offs

Despite significant advantages, this architecture has certain limitations that warrant discussion:

14.3.1 Consensus Layer Modifications

Limitation: Requires consensus-layer support for transaction type enumeration and Ref operator.

Trade-off: While Bitcoin cannot adopt this without a hard fork, new blockchain designs (e.g., Kaspa, Sui) can integrate these features natively.

Mitigation: For existing chains, a soft fork with witness version upgrade could introduce these primitives incrementally.

14.3.2 UTXO Set Growth

Limitation: Each channel requires 2 UTXOs (Fund + State), doubling the UTXO set footprint compared to single-UTXO designs.

Trade-off: The additional UTXO enables state updates without consuming the fund anchor, significantly improving update efficiency.

Mitigation: UTXO set pruning strategies (Section 8.6.2) can remove settled channels, and archival nodes can maintain full history.

14.3.3 Privacy vs. Discovery

Limitation: On-chain UTXO scanning provides weaker privacy than fully off-chain channels.

Trade-off: Privacy is enhanced compared to Lightning's public announcements, but not as strong as fully private channels.

Mitigation: Ephemeral channel identities (Section 7.3.1) and balance commitments (Section 7.3.2) provide significant privacy improvements.

14.4 Future Research Directions

14.4.1 Short-Term Extensions

1. **Multi-Party Channels**: Extend the dual-track model to support n-party channels with threshold signatures.

   - Challenge: Efficient state agreement among n participants
   - Approach: Combine MuSig2 with consensus protocols like PBFT or HotStuff

2. **Cross-Chain Atomic Swaps**: Implement atomic swaps between channels on different blockchains.
   - Challenge: Ensuring atomicity across heterogeneous consensus protocols
   - Approach: Adaptor signatures with chain-specific timelocks

3. **Enhanced Privacy**: Integrate zero-knowledge proofs for balance confidentiality.
   - Challenge: Proving balance validity without revealing amounts
   - Approach: Bulletproofs or Halo2 for range proofs

4. **Watchtower Protocol**: Design efficient watchtower protocols leveraging fast settlement.
   - Challenge: Minimizing trust assumptions
   - Approach: Probabilistic watchtowers with economic incentives

14.4.2 Long-Term Research

1. **Formal Verification**: Machine-checked proofs of safety properties.
   - Tools: Coq, Isabelle/HOL, or TLA+
   - Goal: Verify state machine transitions, isolation properties, and value conservation

2. **Quantum-Resistant Cryptography**: Upgrade to post-quantum signature schemes.
   - Challenge: Signature size and verification cost
   - Candidates: CRYSTALS-Dilithium, SPHINCS+

3. **Adaptive Topologies**: Machine learning-driven topology optimization.
   - Goal: Predict payment flows and dynamically rebalance channels
   - Approach: Reinforcement learning with liquidity as reward signal

4. **Regulatory Compliance**: Privacy-preserving compliance mechanisms.
   - Challenge: Balance privacy with regulatory requirements
   - Approach: Selective disclosure with cryptographic commitments

5. **Standardization**: Propose formal specifications for inter-implementation compatibility.
   - Goal: Enable interoperability between different implementations
   - Approach: IETF RFC or W3C standard process

14.4.3 Open Research Questions

1. **Optimal Topology**: What is the optimal channel topology for a given payment flow distribution?

2. **Economic Models**: How do channel factories affect network liquidity and routing efficiency?

3. **Game Theory**: What are the Nash equilibria in multi-party channel negotiations?

4. **Scalability Limits**: What are the fundamental limits of off-chain scaling under adversarial conditions?

5. **Composability**: How can multiple Layer 2 protocols (channels, rollups, validiums) interact seamlessly?

14.5 Broader Impact

14.5.1 Impact on Blockchain Scalability

This architecture demonstrates that Layer 2 solutions can achieve:

- **Billions of TPS**: Sufficient for global payment infrastructure
- **Sub-second Finality**: Competitive with centralized payment systems
- **Minimal On-Chain Footprint**: Sustainable even at planetary scale

14.5.2 Impact on Decentralization

By eliminating registries and reducing watchtower dependency:

- **Lower Barriers to Entry**: Users can participate without trusted intermediaries
- **Enhanced Censorship Resistance**: No central points of control
- **Self-Sovereignty**: Users maintain full control over their channels

14.5.3 Impact on Privacy

Registry-free discovery and ephemeral identities provide:

- **Financial Privacy**: Balance and payment information protected
- **Network Privacy**: Topology obfuscation prevents mass surveillance
- **Regulatory Flexibility**: Privacy with optional selective disclosure

14.6 Call to Action

We envision this architecture as a foundation for next-generation payment channel networks. To realize this vision, we invite the community to:

1. **Implement and Test**: Deploy the reference implementation in testnet environments

2. **Formal Verification**: Apply formal methods to verify safety properties

3. **Protocol Extensions**: Develop multi-party channels, cross-chain swaps, and enhanced privacy features

4. **Standardization**: Contribute to formal specifications for interoperability

5. **Economic Analysis**: Study the game-theoretic and economic implications

14.7 Concluding Remarks

Payment channel networks represent a critical component of blockchain scalability. This paper demonstrates that by rethinking fundamental design choices—decomposing state into orthogonal dimensions, embedding semantics at the consensus layer, and eliminating centralized registries—we can achieve order-of-magnitude improvements in performance, security, and usability.

The dual-track state machine architecture is not merely an incremental optimization but a fundamental reimagining of how off-chain state can be managed. By achieving $\mathcal{O}(1)$ state complexity, sub-second settlement, and registry-free operation, this architecture brings payment channels closer to the vision of a truly decentralized, scalable, and private global payment infrastructure.

Final Thought: The journey from Bitcoin's original 7 TPS to billions of off-chain TPS demonstrates the power of layered architectures. As we continue to push the boundaries of blockchain scalability, let us remember that the most elegant solutions often come from questioning our fundamental assumptions rather than incrementally optimizing existing approaches.

"The best way to predict the future is to invent it." — Alan Kay

Acknowledgments

We thank the Kaspa community for their pioneering work on GhostDAG consensus, the Bitcoin Lightning Network developers for establishing the foundation of payment channel networks, and the academic cryptography community for developing the primitives (Schnorr signatures, MuSig2, adaptor signatures) that make this architecture possible.

A Glossary and Preliminaries

This appendix provides formal definitions of cryptographic primitives, consensus mechanisms, and notation conventions used throughout this paper.

A.1 Cryptographic Foundations

Definition A.1 (Elliptic Curve Group). *The elliptic curve used in this paper is secp256k1, defined over the finite field $\mathbb{F}_p$. Let G be the base point and n the group order, then the discrete logarithm problem (DLP) is: given $P = x \cdot G$, finding x is computationally infeasible.*

Definition A.2 (Schnorr Signature). *Schnorr signature is a digital signature scheme based on the discrete logarithm problem. Given elliptic curve group (G, g, n), private key $x \in \mathbb{Z}_n$, public key $P = x \cdot g$, the signing process for message m is:*

1. *Choose random number k, compute $R = k \cdot g$*
2. *Compute $e = H(R \| P \| m)$*
3. *Compute $s = k + e \cdot x \mod n$*
4. *Signature is (R, s)*

*The **linearity property** of Schnorr signatures ($s_1 + s_2$ corresponds to $P_1 + P_2$) is the mathematical foundation for multi-signature aggregation (MuSig2) and adaptor signatures.*

Definition A.3 (MuSig2 Multi-Party Signature). *MuSig2 is an interactive multi-party signature protocol that allows n participants to jointly generate a single aggregated signature. Let the set of participant public keys be $\{P_1, \ldots, P_n\}$, the aggregated public key is:*

$$P_{agg} = \sum_{i=1}^{n} a_i \cdot P_i, \quad \text{where } a_i = H(L \| P_i), L = H(P_1 \| \cdots \| P_n)$$

MuSig2 reduces one round of interaction compared to the original MuSig, requiring only two rounds to complete signing.

Definition A.4 (Adaptor Signature). *Adaptor signature is an "incomplete" pre-signature $\tilde{\sigma}$ that requires knowledge of a secret value t to be converted into a valid signature σ:*

$$\sigma = Adapt(\tilde{\sigma}, t)$$

Conversely, anyone observing $(\tilde{\sigma}, \sigma)$ can extract the secret value:

$$t = Extract(\tilde{\sigma}, \sigma)$$

Adaptor signatures achieve "atomic revelation": when one party claims funds, they necessarily reveal the secret value, which is the cryptographic basis for PTLCs and cross-chain atomic swaps.

Definition A.5 (Hash Function and Commitment). *The hash function $H : \{0,1\}^* \to \{0,1\}^{256}$ used in this paper satisfies the following security properties:*

- ***Preimage resistance**: Given h, finding m such that $H(m) = h$ is computationally infeasible*
- ***Collision resistance**: Finding $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$ is computationally infeasible*

Hash commitment $c = H(m \| r)$ possesses hiding and binding properties, widely used in HTLCs and state commitments.

Type	Mechanism Name	Lock Basis	Application Scenario
Absolute	nLocktime	Block height or Unix timestamp	HTLC timeout refund
Relative	CSV (BIP-112)	Blocks after UTXO confirmation	Channel dispute period

A.2 Timelock Mechanisms

Definition A.6 (Timelock). *Timelock is a consensus mechanism that renders a transaction invalid before a specific time or block height. This paper involves two types of timelocks:*

Definition A.7 (DAA Score). *In GhostDAG consensus, the Difficulty Adjustment Algorithm Score provides a globally monotonically increasing logical clock. Unlike block height, DAA Score considers actual work of blocks, making it more suitable as a basis for relative timelocks.*

A.3 Directed Acyclic Graph Consensus

Definition A.8 (GhostDAG Protocol). *Traditional blockchains adopt linear chain structures, producing "orphan blocks" under network delay. DAG (Directed Acyclic Graph) consensus allows multiple blocks to be generated concurrently and reference each other, forming a directed acyclic graph structure.*

Core parameters of the GhostDAG protocol:

- *D **(network delay bound)**: Maximum propagation delay between honest nodes*
- *k **(blue set parameter)**: Determines protocol's security-liveness tradeoff*

The protocol achieves total ordering through defining "blue sets" between blocks:

$$\forall b_1, b_2 \in DAG : b_1 \prec_{blue} b_2 \iff Blue(b_1) < Blue(b_2)$$

where $Blue(b)$ is the blue score of block b, computed by a recursive algorithm.

A.4 Finite State Machine Foundations

Definition A.9 (Finite State Machine). *A finite state machine (FSM) is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$:*

- *Q: Finite set of states*
- *Σ: Input alphabet (set of events/inputs)*
- *$\delta : Q \times \Sigma \to Q$: State transition function*
- *$q_0 \in Q$: Initial state*
- *$F \subseteq Q$: Set of final states*

Definition A.10 (State Machine Determinism). *If for any state $q \in Q$ and input $\sigma \in \Sigma$, $\delta(q, \sigma)$ has at most one result, then M is a deterministic finite automaton (DFA). The channel state machines in this paper strictly satisfy the determinism condition.*

Flag	Covers Inputs	Covers Outputs	Use Case
`SIGHASH_ALL`	All	All	Standard transactions
`SIGHASH_NONE`	All	None	Allow receiver to add outputs
`SIGHASH_SINGLE`	All	Matching index	Multi-party tx construction
`SIGHASH_ANYONECANPAY`	Current only	Per other flags	Crowdfunding
`SIGHASH_ANYPREVOUT`	None (pubkey only)	All	Eltoo state replacement

Symbol	Meaning
$\mathcal{U}$	UTXO set
U_{fund}	Fund UTXO (funding anchor)
$U_{state}^{(n)}$	State UTXO with sequence number n
τ	Transaction
δ	State transition function
$\text{Ref}(\cdot)$	Read-only reference operation
$\text{Spend}(\cdot)$	Spend operation
$\prec$	Partial order relation
$\cong$	Isomorphism relation
$\perp$	Orthogonality/Independence

A.5 Covenants and Script Extensions

Definition A.11 (Covenant). *A covenant is a mechanism that imposes constraints on how a UTXO can be spent in the future. Formally, a covenant is a predicate $C : Tx \to \{0, 1\}$, where spending transaction τ must satisfy $C(\tau) = 1$.*

Covenant classification:

- ***Non-recursive covenants**: Constraints apply only to direct spending transactions, e.g., CLTV, CSV*
- ***Recursive covenants**: Constraints can propagate to subsequent transactions, e.g., CTV (BIP-119), APO (BIP-118)*

Definition A.12 (SIGHASH Flags). *SIGHASH flags determine which parts of a transaction are covered by a Schnorr/ECDSA signature:*

A.6 Notation Conventions

This paper uses the following notation conventions:

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2016.

[3] C. Decker, R. Russell, and O. Osuntokun, "eltoo: A Simple Layer2 Protocol for Bitcoin," 2018.

[4] C. Decker and A. J. Towns, "BIP-118: SIGHASH_ANYPREVOUT for off-chain protocols," Bitcoin Improvement Proposal, 2019.

[5] Y. Sompolinsky and A. Zohar, "Secure High-Rate Transaction Processing in Bitcoin," in Financial Cryptography and Data Security, 2015.

[6] Y. Sompolinsky et al., "Phantom and GhostDAG: A Scalable Generalization of Nakamoto Consensus," Cryptology ePrint Archive, 2021.

[7] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr Multi-Signatures with Applications to Bitcoin," in IACR ePrint, 2018.

[8] J. Nick, T. Ruffing, and Y. Seurin, "MuSig2: Simple Two-Round Schnorr Multi-Signatures," in CRYPTO, 2021.

[9] A. Poelstra, "Mimblewimble," 2016.

[10] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability," in NDSS, 2019.

[11] L. Aumayr et al., "Generalized Bitcoin-Compatible Channels," Cryptology ePrint Archive, 2021.

[12] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and State Channels: Payment Networks that Go Faster than Lightning," in Financial Cryptography, 2019.

[13] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual Payment Hubs over Cryptocurrencies," in IEEE S&P, 2019.

[14] L. Lamport, "Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers," Addison-Wesley, 2002.

[15] T. Coquand and G. Huet, "The Calculus of Constructions," Information and Computation, 1988.