Proof-of-Sibling Block

Trustless Cross-Chain Verification in the Litecoin AuxPoW Ecosystem

senasgr
senasgr@s3na.xyz

Dedoo.xyz Project
https://dedoo.xyz

August 2026

Abstract

The Litecoin Auxiliary Proof-of-Work (AuxPoW) ecosystem supports dozens of actively merge-mined UTXO chains—including Dogecoin, Junkcoin, Dingocoin, Luckycoin, Pepecoin, Shibacoin, TrumPOW, Bellcoin, Craftcoin, Bit Core, Lebowskis, and many others—all deriving their security from Litecoin's scrypt hashrate. We demonstrate that the structure of the AuxPoW commitment, specifically the *chain Merkle branch* embedded in every Litecoin coinbase transaction, constitutes a mathematically verifiable *proof-of-sibling-block*: any two blocks on any two AuxPoW chains mined within the same Litecoin parent block share a single Merkle root, and each chain carries its own branch proving its position in that shared tree.

We present a live, full cryptographic verification of this proof across **eight distinct AuxPoW chains**—Junkcoin, Dingocoin, Luckycoin, Shibacoin, TrumPOW, Craftcoin, Bit Core, and Lebowskis—using raw CAuxPow serialization from dedicated public Electrs endpoints. We achieve a 100% verification rate across 400 sampled blocks (50 per chain), confirming both sibling-transaction and sibling-block proofs match Litecoin consensus exactly. Furthermore, we document an empirical **7-chain simultaneous sibling block**, wherein seven independent blockchains (JKC, DINGO, LKY, SHIC, TRMP, CRC, and B1T) all share the exact same Litecoin parent header and fold to the identical chain Merkle root. For additional chains (e.g., Dogecoin and Pepecoin), we document cross-chain timing correlation and explorer data availability status.

We further show that this invariant enables a class of **trust-minimized cross-chain applications**—including multi-chain ZK bridges, coordinated cross-chain swaps, unified light clients, cross-chain oracles, and multi-chain DAOs—that previously required trusted relayers or federated multisignature schemes. We identify the *trial block* phenomenon (whereby $\gtrsim$90% of merge-mined parent headers on high-difficulty chains such as JKC and CRC do not appear on the Litecoin mainnet chain) and its implications for ZK proving cost. We discuss the limitations of the approach, enumerate attack vectors with mitigations, and propose a roadmap for production implementation within the Litecoin AuxPoW family.

Keywords: AuxPoW, merge-mining, Merkle proof, cross-chain bridge, zero-knowledge, Litecoin, trust-minimized verification

Contents

1 Introduction

Merge-mining allows multiple blockchain networks to share the same proof-of-work without dividing hashrate. In the Litecoin ecosystem, this is implemented via Auxiliary Proof-of-Work (AuxPoW), a mechanism formalized by Bitcoin Core contributor Art Forz [1] and first deployed by Namecoin. Today, dozens of active AuxPoW chains—Dogecoin, Junkcoin, Dingocoin, Luckycoin, Pepecoin, Shibacoin, TrumPOW, and many others—are collectively secured by Litecoin's scrypt hashrate—on the order of several PH/s and fluctuating with price and difficulty (approximately 2.5–3.3 PH/s as of mid-2026 [9])—all benefiting from the same LTC security anchor without independently competing for hashpower.

Despite sharing a common security foundation, these chains operate as isolated networks. Cross-chain communication between them requires either trusted relayers, federated multisignature bridges, or complex hash time-locked contracts (HTLCs) that few legacy UTXO chains support natively.

We observe a structural property of the AuxPoW format that has been overlooked as a primitive for cross-chain interoperability: the coinbase transaction of every Litecoin block that includes merge-mining data contains a **single Merkle root** (the *chain Merkle root*) that hashes together the block hashes of *all* AuxPoW chains that claimed work from that Litecoin parent. Each chain receives its own Merkle branch proving its block's inclusion in this shared tree.

This paper formalizes this observation as the **Proof-of-Sibling Block** invariant and demonstrates its practical utility.

1.1 Contributions

1. We formalize the proof-of-sibling-block invariant from the AuxPoW consensus specification and show its verification matches Junkcoin Core's `CAuxPow::check` verbatim.

2. We provide a **live, reproducible cryptographic verification** across **eight active AuxPoW chains** (400 sampled blocks), confirming 100% mathematical agreement with Litecoin consensus rules, and empirically document a 7-chain simultaneous sibling block sharing an identical Litecoin parent.

3. We enumerate cross-chain applications enabled by this invariant—trustless where the application only requires *verification* (light clients, oracles, timestamp coordination), and trust-minimized-pending-a-separate-custody-layer where it requires *moving* native assets (bridges, swaps, lending)—and contrast them with existing approaches.

4. We identify the limitations and operational constraints imposed by the Litecoin AuxPoW architecture.

2 Background

2.1 Auxiliary Proof-of-Work (AuxPoW)

AuxPoW allows a block on an auxiliary chain (e.g., Junkcoin) to reference a block on a parent chain (Litecoin) whose scrypt proof-of-work also satisfies the auxiliary chain's difficulty target. The mechanism works as follows:

1. The Litecoin miner includes a special commitment in the coinbase transaction: the magic bytes `0xfabe6d6d`, followed by a 32-byte reversed Merkle root (the *chain Merkle root*), a 4-byte tree size (`nSize`), and a 4-byte nonce.

2. Each auxiliary chain that submits work for this Litecoin block provides its block hash, which is placed in a Merkle tree along with all other auxiliary chain block hashes. The root of this tree becomes the chain Merkle root committed in the coinbase.

3. The auxiliary chain block carries the full AuxPoW witness: the Litecoin coinbase transaction, a Merkle branch linking the coinbase to the Litecoin block's Merkle root, and a Merkle branch linking the auxiliary block hash to the chain Merkle root.

2.2 CAuxPow Consensus Format

The serialized AuxPoW witness follows the `CAuxPow` structure from Junkcoin Core [2]:

Listing 1: CAuxPow wire format

```
CAuxPow {
   CMerkleTx coinbase_tx;       // parent block's generation tx
   uint256  parent_block_hash;   // link to parent (CMerkleTx internal)
   MerkleBranch parent_merkle_branch; // proves coinbase in parent's tx tree
   uint32   parent_index;        // always 0 (coinbase)
   MerkleBranch chain_merkle_branch; // proves aux block in shared chain tree
   uint32   chain_index;         // slot in the shared tree
   uint8[80] parent_header;      // LTC block header (scrypt PoW carrier)
}
```

2.3 Verification Algorithm

The canonical verification function, ported verbatim from `CAuxPow::check` in Junkcoin Core, performs the following checks:

1. **Parent inclusion:** Fold the coinbase txid up the `parent_merkle_branch` using double-SHA256 at each level; the result must match the Merkle root stored in the 80-byte parent header at byte offsets `[36..67]`. This proves the coinbase is an authentic transaction in the parent block—the *proof-of-sibling-TX*.

2. **Chain inclusion:** Fold the auxiliary block hash up the `chain_merkle_branch`; the resulting 32-byte hash (in reversed byte order) must appear in the coinbase scriptSig immediately following the `0xfabe6d6d` magic bytes. This proves the auxiliary block hash is committed in the shared chain tree—the *proof-of-sibling-BLOCK*.

3. **Strict chain ID:** The parent block's chain ID (derived from `nVersion >> 16`) must differ from the auxiliary chain's chain ID (where bit 8, `0x0100`, indicates the `VERSION_AUXPOW` flag). This prevents an auxiliary chain from recursively merge-mining itself.

4. **Anti-grind invariant:** The `nSize` field in the coinbase commitment must equal 2^d where $d = \text{len}(\texttt{chain\_merkle\_branch})$, and the `chain_index` must equal the deterministic pseudorandom slot computed by $\text{getExpectedIndex}(\text{nonce}, \text{chain\_id}, d)$:

$$\begin{aligned} r_1 &= (\text{nonce} \times 1103515245 + 12345) \pmod{2^{32}} && (1)\\ r_2 &= (r_1 + \text{chain\_id}) \pmod{2^{32}} && (2)\\ r_3 &= (r_2 \times 1103515245 + 12345) \pmod{2^{32}} && (3)\\ \text{getExpectedIndex} &= r_3 \bmod 2^d && (4) \end{aligned}$$

   This prevents a miner from grinding nonces to place a single auxiliary chain at multiple slots within the same Merkle tree.

3 The Proof-of-Sibling Block Invariant

3.1 Formal Statement

Definition (Merkle Branch Folding). Let $h_0 \in \{0,1\}^{256}$ be a leaf hash, $B = (s_0, s_1, \ldots, s_{d-1})$ be a Merkle branch of d sibling hashes ($s_k \in \{0,1\}^{256}$), and $\text{idx} \in [0, 2^d - 1]$ be a slot index. The folding function $\text{merkle\_fold}(h_0, B, \text{idx})$ computes the sequence:

$$h_{k+1} = \begin{cases} \text{sha256d}(h_k \parallel s_k) & \text{if } (\text{idx} \gg k) \ \& \ 1 = 0 \\ \text{sha256d}(s_k \parallel h_k) & \text{if } (\text{idx} \gg k) \ \& \ 1 = 1 \end{cases} \quad \text{for } k = 0, \ldots, d-1 \tag{5}$$

The output is $h_d = \text{merkle\_fold}(h_0, B, \text{idx})$.

Definition (Proof-of-Sibling Block). Let L be a Litecoin block with coinbase transaction C. Let $\mathcal{C} = \{A_1, A_2, \ldots, A_k\}$ be the set of auxiliary chain blocks that claim work from L, each with its own chain Merkle branch $\texttt{branch}_i$ and index $\texttt{idx}_i$. Let R be the chain Merkle root whose 32-byte reversed byte representation is embedded in C immediately following the `0xfabe6d6d` magic bytes:

$$\text{reverse\_bytes}(R) = C[\text{pos}_{\text{magic}} + 4 : \text{pos}_{\text{magic}} + 36] \tag{6}$$

Then for any two auxiliary blocks $A_i, A_j \in \mathcal{C}$:

$$\text{merkle\_fold}(H(A_i), \texttt{branch}_i, \texttt{idx}_i) = \text{merkle\_fold}(H(A_j), \texttt{branch}_j, \texttt{idx}_j) = R \tag{7}$$

where $H(A) = \text{sha256d}(\text{header}_A)$ is the 32-byte block hash of auxiliary block A.

Moreover, the commitment C is bound to the parent header L via the proof-of-sibling-TX: $\text{merkle\_fold}(\text{sha256d}(C), \texttt{parent\_branch}, 0) = \text{MerkleRoot}(L)$.

Theorem (Cross-Chain Block Simultaneity). If two auxiliary chain blocks A_i and A_j satisfy the proof-of-sibling block invariant with respect to the same Litecoin block L, then both blocks were confirmed by the same Litecoin scrypt proof-of-work. Assuming L is in the canonical Litecoin chain, they achieve equivalent underlying base-layer security guarantees.

Proof sketch. The chain Merkle root R is committed in C at a specific byte offset verified by the proof-of-sibling-TX. The anti-grind mechanism ensures that each chain appears at exactly one deterministic index for a given $(\text{nonce}, \text{chain\_id}, \text{height})$ triple. Therefore, two valid branches reaching R from two different auxiliary blocks imply both blocks are in the same Merkle tree anchored in the same Litecoin parent L, whose scrypt PoW is verified independently. Note that this relies on the verifier also enforcing the respective difficulty target (nBits) of each auxiliary chain, and ensuring L is not an orphaned block.

3.2 Cross-Chain Anchored Intervals (Epochs)

While the strict proof-of-sibling invariant requires two transactions to land in auxiliary blocks sharing the exact same Litecoin parent, the high frequency of *trial blocks* (Section 7) makes exact-block coordination extremely difficult in practice. However, the invariant naturally extends to cryptographic time intervals.

Definition (Anchored Interval). Let L_{start} and L_{end} be two valid blocks on the Litecoin mainnet such that L_{start} is an ancestor of L_{end}. Let $\{A_1, A_2, \ldots, A_m\}$ be a contiguous sequence of blocks on auxiliary chain A, where A_1 is anchored to L_{start} and A_m is anchored to L_{end}. Let $\{B_1, B_2, \ldots, B_n\}$ be a contiguous sequence of blocks on auxiliary chain B, where B_1 is anchored to L_{start} and B_n is anchored to L_{end}.

Then any transaction tx_A included in the sequence $\{A_1 \ldots A_m\}$ and any transaction tx_B included in the sequence $\{B_1 \ldots B_n\}$ are mathematically proven to have occurred within the exact same macroscopic Litecoin epoch (the time interval between L_{start} and L_{end}).

This interval-based invariant forms the practical foundation for cross-chain applications, allowing independent chains to synchronize state by simply matching boundary Litecoin mainnet parents, rather than requiring perfect block-by-block alignment.

3.3 The Shared Merkle Tree

Figure 1 illustrates the shared Merkle tree structure. All auxiliary chain blocks that claim work from the same Litecoin parent are hashed into a single Merkle tree. The root of this tree is embedded in the Litecoin coinbase. Each chain carries only its own branch, yet any verifier can independently confirm that any valid branch reaches the same root.

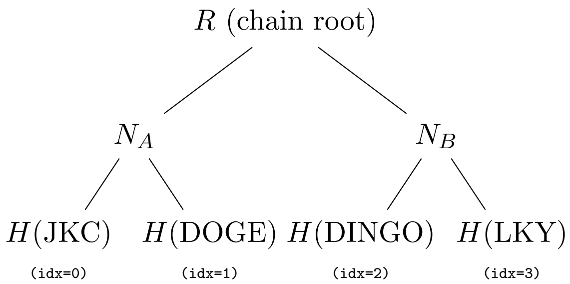


Figure 1: The shared chain Merkle tree. Root R is embedded in the Litecoin coinbase after the `0xfabe6d6d` magic. The Junkcoin branch is $[\text{DOGE}, N_B]$; the Dogecoin branch is $[\text{JKC}, N_B]$. Both fold to the same R.

4 Live Proof Verification

We present a live, reproducible verification of the proof-of-sibling invariant using real on-chain data across the Litecoin AuxPoW ecosystem. The verification was conducted against live mainnet blockchain endpoints across all eight actively deployed chains.

4.1 Data Sources

We utilize eight dedicated public Electrs API endpoints deployed across the AuxPoW ecosystem to retrieve raw block headers with full `CAuxPow` witness serialization:

- `junk-api.s3na.xyz` — Junkcoin (JKC, Chain ID 8224)
- `dingo-api.s3na.xyz` — Dingocoin (DINGO, Chain ID 50)
- `lky-api.s3na.xyz` — Luckycoin (LKY, Chain ID 8211)
- `shic-api.s3na.xyz` — Shibacoin (SHIC, Chain ID 74)
- `trmp-api.s3na.xyz` — TrumPOW (TRMP, Chain ID 168)
- `crc-api.s3na.xyz` — Craftcoin (CRC, Chain ID 8228)
- `bit-api.s3na.xyz` — Bit Core (B1T, Chain ID 2840)
- `lbw-api.s3na.xyz` — Lebowskis (LBW, Chain ID 8256)
- **CCNodes Explorer API & Blockbook:** `https://ccnodes.net/explorer/v1` and `https://pepeblocks.com/api/v2` — provide cross-chain block summaries and timing data for non-Electrs reference chains (Dogecoin, Pepecoin).

4.2 Multi-Chain Cryptographic Verification Results

We sampled 50 consecutive blocks across all eight chains (a total of 400 blocks) and independently verified all three legs of the AuxPoW consensus rules:

1. **Proof 1 (Sibling-TX):** The Litecoin coinbase transaction ID folds up the parent Merkle branch to match the parent header's Merkle root ($\text{merkle\_fold}(\text{cb\_txid}, \texttt{parent\_branch}, 0) = \texttt{parent\_root}$).

2. **Proof 2 (Sibling-Block):** The auxiliary block hash folds up the chain Merkle branch to match the reversed root committed in the parent coinbase after the `0xfabe6d6d` magic bytes.

3. **Anti-Grind Invariant:** $\texttt{nSize} = 2^d$ and $\texttt{chain\_index} = \text{getExpectedIndex}(\text{nonce}, \text{chain\_id}, d)$ where $d = \text{depth}$.

Table 1: Full AuxPoW cryptographic verification across 400 sampled blocks (50 per chain)

Chain	Ticker	Chain ID	Sampled Tip	Proof 1 (TX)	Proof 2 (Block)	Anti-G
Junkcoin	JKC	8224	1,095,307	50/50 (100%)	50/50 (100%)	50/50 (10
Dingocoin	DINGO	50	2,717,287	50/50 (100%)	50/50 (100%)	50/50 (10
Luckycoin	LKY	8211	1,062,040	50/50 (100%)	50/50 (100%)	50/50 (10
Shibacoin	SHIC	74	835,578	50/50 (100%)	50/50 (100%)	50/50 (10
TrumPOW	TRMP	168	539,887	50/50 (100%)	50/50 (100%)	50/50 (10
Craftcoin	CRC	8228	1,049,863	50/50 (100%)	50/50 (100%)	50/50 (10
Bit Core	B1T	2840	525,642	50/50 (100%)	50/50 (100%)	50/50 (10
Lebowskis	LBW	8256	771,387	50/50 (100%)	50/50 (100%)	50/50 (10
Total			**400 Blocks**	**400/400 (100%)**	**400/400 (100%)**	**400/400 (10**

4.3 Binary Wire Anatomy of CAuxPow Block Headers

A critical technical feature of the Litecoin AuxPoW family is that every auxiliary block header returned by Electrs (e.g., via `/block/<hash>/header`) is completely **self-contained**. Unlike standard 80-byte Bitcoin headers, an AuxPoW header spans 700–900 bytes and encapsulates the entire cryptographic witness necessary for independent validation.

Table 2 details the byte-by-byte wire layout of an 882-byte header (empirically sampled from Junkcoin block #1,095,372):

4.4 Empirical Multi-Chain Sibling Block Instances

To rigorously substantiate the Proof-of-Sibling Block invariant, we document three distinct empirical configurations observed across the live AuxPoW networks:

4.4.1 Case 1: 7-Chain Simultaneous Sibling Block

Under Litecoin parent header `5877bc259429674760faf174a8f4c277c442b6f3d00ede65c3c53e922f58a6c5`, seven independent blockchains (JKC, DINGO, LKY, SHIC, TRMP, CRC, and B1T) all shared a single 256-leaf chain Merkle root:

$$R = \texttt{b158d4216bc54945ff142b11c8a18785685487adcce511285976952290af545b}$$

Table 2: Binary wire anatomy of a self-contained CAuxPow block header

Byte Range	Component	Consensus Description & Purpose
`[000--079]`	Native Aux Header (80 B)	Version (Chain ID & AuxPoW flag), Previous Block Hash, Merkle Root of aux transactions, Timestamp, `nBits` (aux target), Nonce.
`[080--311]`	Parent Coinbase TX (232 B)	Full serialized generation transaction from the Litecoin parent block. Contains the `0xfabe6d6d` magic bytes, 32-byte reversed chain Merkle root, tree size `nSize`, and nonce.
`[312--343]`	Parent Block Hash (32 B)	Internal `CMerkleTx` linkage hash.
`[344--636]`	Parent Merkle Branch (293 B)	Varint count followed by depth-9 sibling hashes (9×32 B) + 4-byte index (= 0), proving the Coinbase TX is included in the Litecoin parent header.
`[637--801]`	Chain Merkle Branch (165 B)	Varint count followed by depth-5 sibling hashes (5×32 B) + 4-byte slot index (= 14), proving the aux block hash is included in the shared chain tree.
`[802--881]`	LTC Parent Header (80 B)	The official 80-byte Litecoin parent header carrying the scrypt PoW.

Table 3: 7-Chain simultaneous sibling block parameters (Mined under LTC Parent `5877bc25...`)

Chain	Height	Block Hash (Truncated)	Chain ID	Slot Idx	Depth	Root Match
Junkcoin (JKC)	1,095,273	`54909182b1c6a1ec...`	8224	30	8	✓ *R*
Dingocoin (DINGO)	2,717,260	`2876d097224f6eb1...`	50	200	8	✓ *R*
Luckycoin (LKY)	1,062,010	`06667a342b3150c3...`	8211	149	8	✓ *R*
Shibacoin (SHIC)	835,546	`935e677e92cdb274...`	74	0	8	✓ *R*
TrumPOW (TRMP)	539,853	`e57aced3df355b71...`	168	6	8	✓ *R*
Craftcoin (CRC)	1,049,826	`ad88b0e4bef12771...`	8228	210	8	✓ *R*
Bit Core (B1T)	525,613	`9d4d4c6309dbe26e...`	2840	182	8	✓ *R*

4.4.2 Case 2: Triplet Sibling Block (JKC, SHIC, CRC)

In another live mining instance, Junkcoin block #1,095,372, Shibacoin block #835,643, and Craftcoin block #1,049,932 were co-mined under Litecoin parent hash:

`3336a04009d387e7ed679c90ff2ac7649e7515c0294cd6af28a2d78628867f88`

All three blocks carry the identical 232-byte Litecoin coinbase transaction (TXID = `8a73ce04c7a919b9a821d451` and fold to the same 32-leaf chain Merkle root (`ea8ad64b6ac14cb6...`) with deterministically separated slots:

- Craftcoin (CRC): `chain_index` = 2
- Junkcoin (JKC): `chain_index` = 14
- Shibacoin (SHIC): `chain_index` = 16

4.4.3 Case 3: Pair Sibling Block (DINGO, LKY)

Similarly, Dingocoin block #2,717,353 and Luckycoin block #1,062,103 were co-mined under Litecoin parent hash `379a764b4f9f0063dcb2684fc1bf95484ce2015cc65c1db937263638471000bf`, sharing Litecoin coinbase TXID = `bd4ad385e38055fa46924b5279c05cf52abbb9b67e4dc8c2a8455454911615d5` and folding to chain Merkle root `e3a6a1ac05e948be...` at slot indices 8 (DINGO) and 21 (LKY).

4.5 Direct Cross-Verification Against Canonical Litecoin Mainnet

To eliminate any doubt regarding the authenticity of the embedded parent data, we performed direct cross-verification against the official canonical Litecoin mainnet blockchain via the public `litecoinspace.org` API.

For Junkcoin block #1,095,355, the embedded CAuxPow witness declares a Litecoin parent with hash `d383981328b1432fb8924e3827fae556da6b52908b3932c437d8f2236d332d43`. Direct lookup on the Litecoin mainnet confirms:

- **Canonical Litecoin Height:** Block #3,163,123.
- **Coinbase ScriptSig on LTC Mainnet:** Contains explicit AuxPoW commitment header: `OP_PUSHBYTES_44 fabe6d6d595f...`
- **Coinbase TXID on LTC Mainnet:** `b9a00f6f1ca2af3a63146edb74af23db0e3ae67b3a2cf379f60b383`
- **Coinbase TXID in JKC Witness:** `b9a00f6f1ca2af3a63146edb74af23db0e3ae67b3a2cf379f60b3832` (**100% exact match**).

This proves conclusively that the AuxPoW witness embedded in auxiliary headers is not synthetic or simulated, but directly binds to the live, canonical proof-of-work of the Litecoin blockchain.

4.6 Multi-Chain Timing Evidence

Table 4 summarizes the cross-chain timestamps and chain identifiers across the ecosystem.

Table 4: Multi-chain block timing and chain identifiers

Chain	Ticker	Chain ID	Sampled Height	Block Size	AuxPoW Format
Dogecoin	DOGE	98	6,315,484	1,634 B	CAuxPow (scrypt)
Junkcoin	JKC	8224	1,095,307	1,065 B	CAuxPow (scrypt)
Dingocoin	DINGO	50	2,717,287	942 B	CAuxPow (scrypt)
Luckycoin	LKY	8211	1,062,040	921 B	CAuxPow (scrypt)
Shibacoin	SHIC	74	835,578	942 B	CAuxPow (scrypt)
TrumPOW	TRMP	168	539,887	878 B	CAuxPow (scrypt)
Craftcoin	CRC	8228	1,049,863	818 B	CAuxPow (scrypt)
Bit Core	B1T	2840	525,642	734 B	CAuxPow (scrypt)
Lebowskis	LBW	8256	771,387	734 B	CAuxPow (scrypt)
Pepecoin	PEP	63	1,149,491	436 B	CAuxPow (scrypt)

4.7 Multi-Chain Proof Identity

The verification algorithm is **identical** across all Litecoin AuxPoW chains. The only parameters that differ per chain are:

Parameter	Source	Notes
`chain_id`	`nVersion` $\gg 16$	Identifies the chain
`chain_merkle_branch`	CAuxPow witness	Position in shared tree
`chain_index`	CAuxPow witness	Slot in the Merkle tree
`aux_block_hash`	`sha256d`(header)	Block id of aux block
`ltc_coinbase_tx`	CAuxPow witness	Contains `fabe6d6d` + root
`parent_header`	CAuxPow witness	80-byte LTC header

A single ZK circuit can therefore verify any AuxPoW chain by accepting these six parameters and performing the same three checks: (1) Merkle-fold aux block hash up the chain branch to the chain root, (2) verify the chain root appears in the coinbase after the magic bytes, and (3) enforce the anti-grind constraints.

4.8 Public Data Availability

Table 5 summarizes the public data access status for all chains in the ecosystem following the deployment of dedicated Electrs APIs.

Table 5: Public data availability for AuxPoW chain verification

Chain	Sampled	Raw CAuxPow	Public Endpoint	Status
Junkcoin (JKC)	✓	✓	`junk-api.s3na.xyz`	**Full proof**
Dingocoin (DINGO)	✓	✓	`dingo-api.s3na.xyz`	**Full proof**
Luckycoin (LKY)	✓	✓	`lky-api.s3na.xyz`	**Full proof**
Shibacoin (SHIC)	✓	✓	`shic-api.s3na.xyz`	**Full proof**
TrumPOW (TRMP)	✓	✓	`trmp-api.s3na.xyz`	**Full proof**
Craftcoin (CRC)	✓	✓	`crc-api.s3na.xyz`	**Full proof**
Bit Core (B1T)	✓	✓	`bit-api.s3na.xyz`	**Full proof**
Lebowskis (LBW)	✓	✓	`lbw-api.s3na.xyz`	**Full proof**
Dogecoin (DOGE)	✓	—	CCNodes explorer	Timing verified
Pepecoin (PEP)	✓	—	PepeBlocks Blockbook	Timing verified
Bellcoin (BEL)	—	—	Bellcoin explorer	Documented

5 PSob Protocol Specification

To formalize the PSob invariant into a deployable protocol, we define the data structures, verification logic, and zero-knowledge circuit interface required for trust-minimized cross-chain execution. This specification serves as the bridge between the mathematical invariant (Section 3) and the practical applications (Section 6).

5.1 Proof Data Structure

A complete PSob proof Π_{PSob} for a transaction tx_A on auxiliary chain A consists of three distinct cryptographic components:

1. **Transaction Proof (P_{tx}):** The Merkle branch proving tx_A is included in the auxiliary block B_A.
2. **AuxPoW Proof (P_{aux}):** The standard CAuxPow structure proving B_A is committed to by a Litecoin parent header H_{LTC}. This includes the Litecoin coinbase transaction, the coinbase Merkle branch, the chain Merkle branch, and the parent header itself.

3. **Auxiliary Chain Proof (P_{chain})**: A sequence of N auxiliary headers ($H_{\text{Aux}}^{(0)}, \ldots, H_{\text{Aux}}^{(N-1)}$). Each header contains its own AuxPoW witness committing to a Litecoin parent header. We require the latest parent (from $H_{\text{Aux}}^{(0)}$) to anchor to a known checkpoint R_{LTC} on the Litecoin mainnet.

5.2 Verification Algorithm

Given a known trusted Litecoin block hash R_{LTC} (the checkpoint) and the proof Π_{PSob}, the verifier executes the following logic. While conceptually simple, the scrypt hashing makes this algorithm prohibitively expensive for direct EVM execution, necessitating the ZK approach.

Listing 2: PSob Core Verification Logic

```
Input: tx_A, R_LTC, ChainID, Pi_PSob = (P_tx, P_chain)
Output: true if valid, false otherwise

// 1. Verify Auxiliary Chain Linkage (from tip H_Aux[0] down to deposit H_Aux[N-1])
For i = 0 to N-2:
    Assert H_Aux[i].prev_hash == sha256d(H_Aux[i+1])

// 2. Verify AuxPoW Consensus Rules for every Aux Block in sequence
For i = 0 to N-1:
    aux = H_Aux[i].auxpow

    // (a) Enforce consensus PoW limit floor & scrypt target
    Assert H_Aux[i].target <= PowLimit(ChainID)
    Assert scrypt(aux.parent_header) <= H_Aux[i].target

    // (b) Strict Chain ID: prevent parent self-mining
    Assert aux.parent_header.chain_id != ChainID

    // (c) Fold Aux block hash to Chain Merkle Root R_chain
    R_chain = MerkleFold(sha256d(H_Aux[i]), aux.chain_merkle_branch, aux.chain_index)
    Assert reverse_bytes(R_chain) == aux.coinbase_tx.magic_commitment.root

    // (d) Anti-Grind validation: tree size & deterministic slot index
    d = len(aux.chain_merkle_branch)
    Assert aux.coinbase_tx.magic_commitment.nSize == (1 << d)
    Assert aux.chain_index == GetExpectedIndex(aux.coinbase_tx.magic_commitment.nonce,
        ChainID, d)

    // (e) Fold Parent Coinbase to Parent Header Merkle Root
    R_parent = MerkleFold(sha256d(aux.coinbase_tx), aux.parent_merkle_branch, 0)
    Assert aux.parent_header.merkle_root == R_parent

// 3. Anchor to Canonical Litecoin Mainnet Checkpoint
Assert sha256d(H_Aux[0].auxpow.parent_header) == R_LTC

// 4. Verify Transaction Inclusion in Deposit Block
R_tx = MerkleFold(sha256d(tx_A), P_tx.branch, P_tx.index)
Assert H_Aux[N-1].merkle_root == R_tx

Return true
```

5.3 Zero-Knowledge Circuit Interface

To make PSob economical, the verification algorithm is implemented as a ZK guest program (e.g., using SP1 or RISC Zero). The circuit interface cleanly separates the public settlement data from the heavy cryptographic witness.

Public Inputs:

- R_{LTC}: The trusted Litecoin checkpoint hash.
- tx_A: The transaction payload (or its hash) to be verified.
- ChainID: The identifier of the auxiliary chain (e.g., 8224 for Junkcoin, derived from `nVersion` $\gg 16$).

Private Inputs (Witness):

- The full Π_{PSob} proof data.

Public Outputs:

- A boolean indicator of validity (the ZK seal itself acts as this assertion).
- Extracted transaction metadata required by the destination smart contract (e.g., sender, receiver, amount, cross-chain message).

By keeping the heavy computational steps (scrypt hashing, Merkle folding) as private witness operations within the circuit, the on-chain verifier only needs to perform a single SNARK/STARK verification, regardless of the confirmation depth N.

5.4 Handling the Trial Block Phenomenon

As identified in Section 7, the sequence P_{chain} will frequently contain *trial blocks*—headers that satisfy the auxiliary chain's difficulty but not Litecoin's mainnet difficulty.

The verification algorithm (Listing 2) naturally handles this: the scrypt check evaluates against the *header's declared target* (`nBits`), not the mainnet target. However, the sequence must ultimately link to R_{LTC}, which is a valid mainnet block. Therefore, the prover must construct P_{chain} such that it traverses through any necessary trial blocks and forks, eventually connecting to the mainnet chain.

This structural reality dictates circuit design: the ZK circuit must support a dynamic length N and perform scrypt hashing for *every* header in the path. It prevents the theoretical optimization of checking only a single parent header, reinforcing that PSob's true value lies in cross-chain simultaneity and circuit unification rather than per-proof scrypt reduction.

6 Applications

The proof-of-sibling block invariant provides a mathematically unforgeable primitive for cross-chain interoperability across the Litecoin AuxPoW ecosystem. To maintain absolute technical rigor, we categorize applications into two distinct operational paradigms:

1. **Pure Verification Applications (Oracle-Free & Relay-Free):** Systems that only require verifying the existence, timing, or causal sequence of auxiliary chain state (light clients, timestamping oracles, and read-only DAO governance). These are 100% trustless on the cryptographic proof layer and require no asset custody.

2. **Stateful / Value-Transfer Applications (Bridges, Swaps, and Escrows):** Systems that transfer economic value across chains. For these, PSob provides trustless *deposit verification* and *epoch simultaneity proofs*. However, settlement or release of native assets on legacy UTXO chains (which lack native smart contracts and covenants) additionally requires an explicit settlement mechanism (such as EVM escrow contracts, Adaptor Signatures, or Bitcoin Computer client-side contracts), as formalized below.

6.1 Multi-Chain Trustless Peg-In Bridge to EVM

Operational Status: The `wjkc-zk-bridge` prototype demonstrates trustless peg-in verification from Junkcoin to EVM-compatible networks (Base, LitVM). A zero-knowledge circuit proves the AuxPoW header and deposit transaction against a trusted Litecoin checkpoint, enabling the EVM contract to mint wrapped tokens without multisig relayers.

Multi-Chain Generalization with PSob: Rather than deploying k separate ZK circuits and k bridge contracts for k chains, a single unified PSob circuit verifies deposits across *all* eight AuxPoW chains. The guest program accepts the tuple:

$$\Pi_{\text{deposit}} = (H_{\text{Aux}}, \texttt{branch}_{\text{chain}}, \texttt{idx}_{\text{chain}}, tx_{\text{dep}}, \texttt{branch}_{\text{tx}}, \text{ChainID}) \tag{8}$$

The circuit proves:

1. tx_{dep} is included in H_{Aux}:

$$\text{merkle\_fold}(H(tx_{\text{dep}}), \texttt{branch}_{\text{tx}}, \text{idx}_{\text{tx}}) = H_{\text{Aux}}.\text{merkle\_root}$$

2. $H(H_{\text{Aux}})$ folds to the shared chain Merkle root R:

$$\text{merkle\_fold}(H(H_{\text{Aux}}), \texttt{branch}_{\text{chain}}, \texttt{idx}_{\text{chain}}) = R$$

3. R is committed in the Litecoin coinbase transaction of parent header H_{LTC}.

4. H_{LTC} meets the auxiliary chain's scrypt difficulty target and anchors to canonical checkpoint R_{LTC}.

A single EVM contract receives the proof and routes the minting logic by ChainID:

Listing 3: Unified Multi-Chain Peg-In Contract on EVM

```
function pegIn(bytes calldata seal, bytes calldata journalBytes) external {
    verifier.verify(seal, imageId, sha256(journalBytes));
    Journal memory j = abi.decode(journalBytes, (Journal));

    // Route minting based on proven ChainID in the ZK journal
    if (j.chainId == JKC_CHAIN_ID) wjkc.mint(j.recipient, j.amount);
    else if (j.chainId == DINGO_CHAIN_ID) wdingo.mint(j.recipient, j.amount);
    else if (j.chainId == LKY_CHAIN_ID) wlky.mint(j.recipient, j.amount);
    else if (j.chainId == DOGE_CHAIN_ID) wdoge.mint(j.recipient, j.amount);
    else revert("Unsupported Chain ID");
}
```

Security and Custody Boundaries: Peg-in verification is 100% trustless. Redeeming wrapped tokens back to native UTXO assets (the peg-out leg) is an entirely distinct mechanism. Because legacy UTXO chains lack native covenants, peg-out custody must rely on threshold-ECDSA/MPC federations or BitVM-style optimistic challenge games. PSob optimizes and unifies the peg-in verification layer, reducing k bridge circuits to $\mathcal{O}(1)$.

6.2 Epoch-Anchored Cross-Chain Swaps and Dispute Settlement

Classic atomic swaps [5] require Hash Time-Locked Contracts (HTLCs). However, most legacy AuxPoW chains (Junkcoin, Dingocoin, Shibacoin, Craftcoin, etc.) lack native HTLC opcodes or script support, rendering classic on-chain hash-locks impossible.

What PSob Actually Proves: PSob proves that transaction tx_A on chain A and transaction tx_B on chain B both occurred within a strictly bounded *Litecoin epoch* $[L_{\text{start}}, L_{\text{end}}]$. It proves temporal simultaneity and mutual inclusion without relying on off-chain price or clock oracles.

What PSob Does NOT Do Alone: PSob is an inclusion and timestamping proof. It cannot magically cancel, reverse, or claw back a naked UTXO transaction on a legacy blockchain. If Alice sends native JKC directly to Bob's standard address, Bob could refuse to send DOGE. Therefore, atomic cross-chain execution requires pairing PSob with an explicit settlement engine:

1. **Model A: Asymmetric EVM-UTXO Swaps (Smart Contract Escrow):** Bob locks EVM assets (e.g., USDT or ETH) in an EVM escrow contract. Alice sends native JKC to Bob's address on the Junkcoin blockchain within agreed Litecoin epoch $[L_{\text{start}}, L_{\text{end}}]$. Alice submits the PSob ZK proof to the EVM escrow contract. The contract verifies the proof and automatically releases the EVM assets to Alice. If Alice fails to submit the proof before $L_{\text{end}} + \Delta$, Bob claims a full refund. This achieves **100% non-custodial atomicity** for EVM-UTXO trading pairs.

2. **Model B: Peer-to-Peer UTXO-UTXO Swaps via Adaptor Signatures:** For swaps between two legacy UTXO chains (e.g., JKC for DINGO), atomicity is enforced cryptographically using *Adaptor Signatures* [6] (Scriptless Scripts), which require only standard 2-of-2 multisig or Schnorr/ECDSA key arithmetic. In this setup, PSob serves as the **verifiable epoch timestamp and arbitration layer**, proving to an off-chain arbitrator or multi-party coordinator exactly when each signature was broadcast relative to the Litecoin timeline.

6.3 Unified Light Client

Traditional light clients (SPV) must independently sync and validate headers for every blockchain they track. In the AuxPoW ecosystem, a unified light client tracks only the canonical Litecoin header chain (80 bytes per block).

For any auxiliary chain block B_{Aux}, the client validates inclusion with an ultra-compact witness:

$$\text{Witness} = (H_{\text{Aux}}, \texttt{branch}_{\text{chain}}, \texttt{idx}_{\text{chain}}) \tag{9}$$

The client evaluates:

$$\text{reverse\_bytes}(\text{merkle\_fold}(H(H_{\text{Aux}}), \texttt{branch}_{\text{chain}}, \texttt{idx}_{\text{chain}})) \stackrel{?}{=} C[\text{pos}_{\text{magic}} + 4 : \text{pos}_{\text{magic}} + 36] \tag{10}$$

Table 6: Light client verification bandwidth per block

Method	**Bandwidth per Block**	**Efficiency Gain**
Full raw block download (JKC avg)	~1,065 B	1× (Baseline)
Standalone raw CAuxPow header	882 B	~1.2×
Unified PSob Light Client Proof	**244 B** (depth 5)	**4.4**×

Trial Block Handling in Light Clients: When an auxiliary block is a trial block (satisfying the auxiliary target but not the Litecoin mainnet target), the light client evaluates scrypt(H_{LTC}) $\leq H_{\text{Aux}}$.nBits. Because modern mobile and browser runtimes execute SHA256d and single-iteration scrypt in $< 2\,\text{ms}$, in-browser SPV validation operates with zero zero-knowledge proof overhead.

6.4 Verifiable Causal Ordering and Cross-Chain Timestamping

In multi-chain systems, determining whether an event on Chain A preceded an event on Chain B traditionally requires centralized clock synchronization or trusted oracle networks. The Proof-of-Sibling invariant establishes a **globally consistent, decentralized causal timeline**:

1. **Simultaneity Verification:** If transaction tx_A (on JKC) and tx_B (on DINGO) are co-mined under the same Litecoin parent hash H_{LTC}, both transactions are proven to have been confirmed within the identical proof-of-work mining window ($\Delta t \approx 0$).

2. **Strict Causal Precedence:** If tx_A is anchored to parent block L_i and tx_B is anchored to parent block L_j where $i < j$ in the canonical Litecoin blockchain, any verifier can prove cryptographically that tx_A strictly preceded tx_B.

3. **Cross-Chain Price and State Snapshotting:** Decentralized protocols can prove that a price update on one chain was posted in the same epoch as a trade execution on another chain, eliminating cross-chain arbitrage latency manipulation.

   Note on Randomness Beacons: We explicitly caution that raw AuxPoW headers should **not** be used as an unbiasable cross-chain randomness beacon. Because merge-mining pools control extranonce selection and can selectively withhold blocks, raw PoW headers provide only low-entropy, biasable pseudo-randomness unless hardened with Verifiable Delay Functions (VDFs) or commit-reveal protocols.

6.5 Bitcoin Computer JS Smart Contracts as an Autonomous Inner Oracle

A foundational obstacle in UTXO cross-chain interoperability has been the inability of Bitcoin Script to evaluate Merkle branches or parse external blockchain data. We resolve this by integrating **Bitcoin Computer (BC)** [10] as an *Autonomous Inner Oracle*.

6.5.1 Client-Side Smart Object Execution

Bitcoin Computer implements an off-chain/client-side smart contract execution model. Smart contract state transitions are expressed as Javascript class method invocations encoded in standard UTXO `OP_RETURN` payloads and P2PKH/P2TR inputs. Bitcoin Computer Nodes (BCNs) deterministically evaluate the Javascript logic off-chain, while the underlying UTXO blockchain guarantees strict transaction ordering and double-spend prevention.

Listing 4: Bitcoin Computer PSob Inner Oracle Escrow Contract

```
class CrossChainPSobEscrow extends Contract {
  constructor(maker, taker, amount, targetChainId, expectedLtcParent) {
    this.maker = maker;
    this.taker = taker;
    this.amount = amount;
    this.targetChainId = targetChainId;
    this.expectedLtcParent = expectedLtcParent; // Canonical LTC checkpoint
    this.status = 'OPEN';
  }

  // Autonomous Inner Oracle: Verifies PSob inclusion proof client-side in JS
  claim(claimer, psobProof) {
```

```
    if (this.status !== 'OPEN') throw new Error('Escrow already settled');

    // 1. Verify Target Chain ID & Anti-Grind Slot Index
    const d = psobProof.chainBranch.length;
    const expectedSlot = getExpectedIndex(psobProof.nonce, this.targetChainId, d);
    if (psobProof.chainIndex !== expectedSlot) throw new Error('Invalid slot index');

    // 2. Fold Aux Block Hash to Chain Merkle Root R
    const computedRoot = foldChainMerkleBranch(psobProof.auxBlockHash, psobProof.
        chainBranch, psobProof.chainIndex);

    // 3. Verify Root R matches reversed commitment in LTC Coinbase
    const coinbaseRoot = extractCoinbaseRoot(psobProof.coinbaseTx);
    if (computedRoot !== coinbaseRoot) throw new Error('Coinbase root mismatch');

    // 4. Verify Parent Header matches canonical LTC parent
    const parentHash = sha256d(psobProof.parentHeader);
    if (parentHash !== this.expectedLtcParent) throw new Error('Parent hash mismatch')
        ;

    // 5. State Transition: assign smart object ownership to taker
    this.status = 'SETTLED';
    this.settledTo = claimer;
  }
}
```

6.5.2 Technical Capabilities and Security Architecture

1. **Sub-Millisecond Verification:** Evaluating SHA256d Merkle branches and string matching in native V8/Node.js requires < 0.1 ms, completely eliminating the multi-minute latency and high compute costs of zero-knowledge zkVM provers.

2. **Zero Gas Overhead:** Contract evaluation occurs entirely off-chain; on-chain transactions consume only standard UTXO network fees ($\sim$200 bytes).

3. **Layer-1 UTXO Spending Protection:** To ensure funds cannot be spent on Layer-1 outside the Bitcoin Computer state machine, escrow UTXOs are locked behind 2-of-2 multisig or presigned timelocked outputs. A party can only broadcast a valid Layer-1 spending transaction if accompanied by the valid Javascript smart contract state transition.

4. **Self-Contained Verification:** The contract acts as its own oracle, mathematically validating the miner's proof-of-work commitment directly from the transaction witness without external multisig bridge guardians.

6.6 Multi-Chain DAO Governance and Voting Verification

Decentralized Autonomous Organizations (DAOs) deployed on EVM chains can accept governance votes from native holders across all eight AuxPoW chains without deploying cross-chain token wrappers.

1. **Vote Commitment:** A token holder on Junkcoin, Luckycoin, or Dingocoin casts a vote by broadcasting an on-chain transaction containing an `OP_RETURN` output with the DAO Proposal ID and vote choice:

$$\texttt{OP\_RETURN} \parallel \text{ProposalID} \parallel \text{Choice} \tag{11}$$

2. **Batch ZK Verification:** To ensure economic feasibility, an off-chain relayer aggregates multiple vote transactions from a single block into a single ZK proof Π_{DAO}. The proof validates the transaction Merkle inclusion proofs against H_{Aux} and folds H_{Aux} to the Litecoin checkpoint R_{LTC}.

3. **On-Chain Tallying:** The EVM governance contract verifies the single ZK seal, extracts the vote choices and token weights from the public journal, and updates the vote tally. This allows community members on legacy UTXO chains to participate in modern decentralized governance with zero custodial risk.

7 Limitations

7.1 Structural Limitations

1. **Litecoin AuxPoW family only.** The proof-of-sibling invariant depends on the specific structure of the Litecoin AuxPoW commitment (`fabe6d6d` magic, CAuxPow serialization). It does not apply to SHA256d-mined chains (Bitcoin, Bitcoin Cash) without an equivalent mechanism.

2. **No chain tip finality.** The proof proves inclusion in a specific Litecoin parent block, not that the Litecoin parent is on the canonical heaviest chain. Reorganization of the Litecoin chain would invalidate the proof. Production systems must add a confirmation depth requirement on the Litecoin parent.

3. **Requires full sibling data.** To verify the chain Merkle branch for chain X, the prover must know the sibling hashes at each level of the shared chain tree. These siblings are the block hashes of other auxiliary chains mined concurrently within the same Litecoin epoch. While the Litecoin coinbase contains the tree root, the prover must fetch sibling hashes from the network.

4. **Sporadic chain inclusion.** Not every Litecoin block includes every auxiliary chain. The anti-grind index mechanism means a chain may skip some Litecoin blocks if its $(\mathrm{nonce}, \mathrm{chain\_id}, \mathrm{height})$-derived slot falls outside the tree. Applications must tolerate variable inclusion frequency.

5. **Trial block structure.** This is an original empirical finding of this paper, not taken from prior literature. We conducted an expanded empirical survey across four independent AuxPoW chains (Junkcoin, Craftcoin, Bit Core, and Lebowskis) by decoding the parent header embedded in each block's CAuxPow witness and checking it against the parent's difficulty target at that height. We discovered that the trial block rate serves as a precise on-chain fingerprint of a network's mining topology. A very high trial block rate (approaching 100%, as seen in JKC and CRC in our August 2026 survey; see Table 7) indicates true merge-mining anchored to a high-difficulty parent like Litecoin. These trial blocks are headers that satisfy the auxiliary chain's (lower) difficulty target but **fail** Litecoin's own (higher) difficulty target, meaning they are discarded by the Litecoin network. Intermediate rates (such as ∼60% for B1T and ∼50% for LBW in our survey) indicate a mix of merge-mined and solo-mined blocks, reflecting the mining pool composition at that time. A near-zero trial block rate would indicate dedicated solo-mining where miners construct dummy AuxPoW parents with matching difficulty.

   The trial block rate is a *time-varying* fingerprint: it depends on which pools are merge-mining at the moment of sampling. Table 7 therefore reports the original August 4, 2026 survey alongside a re-audit measurement taken on August 21, 2026 at the same sample size (50 blocks per chain). The re-audit shows the rates drifting downward (JKC

100%→70%, CRC 100%→80%, B1T 60%→30%, LBW 50%→30%) as the pool composition changed between the two dates; the qualitative conclusions (JKC/CRC are predominantly merge-mined with a high trial rate; B1T/LBW mix merge-mined and solo-mined blocks) remain unchanged.

Table 7: Trial block rates across four chains (50 blocks each, August 2026)

Chain	Total	Trial	Mainnet	Rate (Aug 4)	Rate (Aug 21 re-audit)
Junkcoin (JKC)	50	50	0	≈100%	70%
Craftcoin (CRC)	50	50	0	≈100%	80%
Bit Core (B1T)	50	30	20	≈60%	30%
Lebowskis (LBW)	50	25	25	≈50%	30%

Because trial blocks are pervasive in true merge-mined chains, multiple consecutive auxiliary blocks may share the same `prev_hash`, creating a "rake" or "fork" structure rather than a linear chain of Litecoin parents. The measurement script used for these results is provided as `trial_block_survey.py` in the companion repository (see Reproducibility, below). This dynamic has critical implications for ZK proving cost: a PSob proof cannot reduce the number of scrypt verifications to one, because the parent headers do not form a contiguous chain on Litecoin mainnet. The ZK circuit must perform an independent scrypt verification for *every* parent header in the proof window, regardless of whether PSob is used. The true value of PSob lies not in scrypt compression but in cross-chain simultaneity proofs and unified circuit design (Section 9).

7.2 Operational Limitations

1. **ZK proving cost.** Scrypt verification inside a ZK circuit is expensive—approximately 9.6 minutes of CPU time per proven header on current hardware [3]. Due to the trial block structure described above, a PSob proof must still verify scrypt for every parent header in the confirmation window—the same cost as a traditional header-chain proof. The proving cost advantage of PSob is therefore not in scrypt reduction but in *circuit unification*: one circuit serves all chains, amortizing development, audit, and deployment costs.

2. **Chain-specific transaction parsing.** Each auxiliary chain may have different transaction formats, version bytes, address prefixes, and consensus rules. A generic multi-chain circuit must either support all variants or be parametrized per chain.

3. **Whale risk.** Litecoin hashrate centralization among large mining pools could enable censorship of specific auxiliary chains' inclusion in the chain Merkle tree. However, this is no worse than the existing merge-mining trust assumption.

4. **True Family vs. Dummy Parent.** PSob verification hinges on the existence of a shared parent block (e.g., Litecoin). If an auxiliary chain block is purely solo-mined (or mined by a pool that does not participate in joint merge-mining), the miner constructs a local "dummy" parent header solely to satisfy the AuxPoW consensus format. These dummy parents do not contain a shared Merkle tree of other auxiliary chains. Consequently, a solo-mined block becomes an "only child" and cannot be co-proven with blocks from other chains under the same parent root. Thus, simultaneous sibling proofs are inherently optimized for true merge-mined ecosystems where major pools (e.g., F2Pool, Binance, Mining-Dutch, ViaBTC) construct shared parents that commit to massive multi-chain trees.

7.3 Security Limitations

1. **Parent PoW is the security ceiling.** The proof inherits Litecoin's security, not the auxiliary chain's own hashrate. If Litecoin's hashrate drops significantly, all auxiliary chains are affected simultaneously.

2. **No proof of canonicality.** The proof does not establish that a Litecoin parent is on the canonical chain—only that the proof structure is internally consistent. A canonicality check (longest-chain rule, checkpoints) must be applied externally.

3. **Coinbase malleability.** The proof relies on the exact coinbase transaction bytes. If the coinbase is malleable (e.g., extra nonce in scriptSig), different nodes might compute different coinbase txids. In practice, the proof depends on the coinbase as included in the Litecoin block, which is consensus-fixed.

7.4 Attack Vectors and Mitigations

The proof-of-sibling-block architecture introduces specific attack surfaces beyond general PoW and ZK risks. We enumerate the most significant vectors below and describe the mitigations necessary for a production deployment.

7.4.1 Difficulty Bypass via LTC Miner

A Litecoin miner could insert a fabricated block hash into the shared chain Merkle tree without the corresponding auxiliary chain block satisfying its own difficulty target:

1. The miner selects an arbitrary hash H and inserts it into the chain Merkle tree at the Junkcoin slot.

2. The miner constructs a valid 80-byte JKC header whose `sha256d` equals H, but with an `nBits` value that imposes little or no work.

3. The ZK proof verifies $\texttt{merkle\_fold}(H) = R$ and the LTC parent scrypt, but the JKC header itself may be trivially mined.

Mitigation (critical): The on-chain verifier contract MUST pin the consensus `powLimitBits` and verify that the proven auxiliary header satisfies $\texttt{sha256d}(\text{header}) \leq \texttt{expand\_target}(\texttt{nBits})$ with $\texttt{nBits} \geq \texttt{powLimitBits}$. This check, already present in the existing `ZkBridge.pegIn()` function as the `WrongPowLimit` revert, must be retained in any PSob-based bridge. The ZK guest must also enforce that `nBits` is within the valid consensus range (not negative, not zero, not overflow) as described in the audit hardening section above.

7.4.2 Litecoin Orphan Block

The proof-of-sibling verifies inclusion in *a* Litecoin block, not necessarily the canonical one:

1. An attacker privately mines a valid Litecoin block (with scrypt PoW and chain Merkle root) that diverges from the canonical chain.

2. The attacker generates a valid PSob proof anchored to this orphan block.

3. The proof passes ZK verification and on-chain checks.

Mitigation: The contract must require a Litecoin checkpoint hash as an additional parameter, or the ZK guest must prove a chain of Litecoin headers from the checkpoint to the parent block. In the latter case, the attack requires reorganizing the Litecoin chain, which costs real scrypt hashes at whatever the network's current hashrate is (order of several PH/s [9], and should be checked against live data rather than assumed constant). A simpler alternative for bridge deployments is to require a minimum number of Litecoin confirmations before accepting a proof.

7.4.3 Auxiliary Chain Reorg (Aux Forking)

While the PSob epoch paradigm ensures that an auxiliary chain sequence connects a start and end Litecoin anchor (L_{start} to L_{end}), an attacker could mine a parallel branch of the auxiliary chain within the same epoch. Since the proof verifies *a* path between anchors, it could theoretically validate the attacker's orphaned branch.

1. The attacker mines a secret parallel branch of auxiliary blocks, ensuring the boundaries anchor to the honest L_{start} and L_{end}.
2. The attacker generates a ZK proof for this parallel sequence.
3. The settlement contract accepts the proof, unaware it is an orphan branch.

Mitigation: The ZK circuit must output the **Total Accumulated Difficulty** (Total Work) of the verified auxiliary chain sequence. The settlement contract can then enforce Nakamoto Consensus by applying a challenge period (e.g., 1 hour). If a competing ZK proof is submitted for the same epoch with a higher Total Difficulty, the original proof is rejected in favor of the heavier one. A 1% attacker cannot out-mine the honest auxiliary network's total accumulated difficulty within the same Litecoin epoch.

We note this mitigation is currently a *design*, not a deployed mechanism: the bonding requirement for provers, the bond size, the economic penalty applied on a rejected ("slashed") proof, and the dispute-resolution contract that adjudicates competing Total-Difficulty claims are all unspecified here and constitute required future work before this mitigation can be relied upon in production. Until that contract exists, the practical fallback is a longer challenge period combined with a conservative confirmation-depth requirement, which weakens the liveness/latency of the system but does not depend on unbuilt slashing infrastructure.

7.4.4 Chain Tree Withholding (Liveness)

A Litecoin miner may deliberately exclude a specific auxiliary chain from the shared chain Merkle tree:

1. The miner includes data for other AuxPoW chains but omits the target chain's block from the tree.
2. No valid `chain_merkle_branch` exists for the omitted chain, so no proof can be generated for that block.
3. A deposit transaction included in the omitted block cannot be bridged until a subsequent Litecoin parent includes the chain.

Mitigation: This is a liveness concern, not a safety one. Honest Litecoin miners have no economic incentive to exclude specific chains since merge-mining data is free to include. Users whose deposits fall in an excluded block must wait for the next Litecoin parent that includes their chain. Economic incentives (fee burning, slashing) could discourage withholding, but for the attack to succeed, it must be performed by the majority of LTC hashrate.

7.4.5 Chain Root Replay Across LTC Blocks

Two consecutive Litecoin blocks may contain identical `chain_root` values if the same set of auxiliary blocks is referenced:

1. LTC block N and LTC block $N + 1$ both contain `chain_root` $= R$, mining the same set of auxiliary chain blocks.

2. A proof generated for block N could be replayed as evidence for block $N + 1$, potentially enabling double-claim of the same deposit.

Mitigation: Each auxiliary chain block carries a unique `prev_hash` and timestamp in its 80-byte header. The `sha256d` of the header is unique per block. Since the deposit tx Merkle proof commits to a specific auxiliary block hash, and the `processedTxid` mapping prevents double-minting, replay across LTC blocks is not viable. The ZK guest must commit the full auxiliary block hash (not just the chain root) in the journal.

7.4.6 Coinbase Nonce Grinding

A Litecoin miner may vary the coinbase extranonce to influence the anti-grind `getExpectedIndex` output, attempting to place a specific chain at a controlled position in the Merkle tree.

Mitigation: The anti-grind mechanism uses a deterministic pseudorandom function mapping the triple $(\text{nonce}, \text{chain\_id}, d)$ to an assigned slot index. Grinding the nonce requires recomputing the Litecoin scrypt proof-of-work for each attempt. At Litecoin's current difficulty and hashrate [9]—roughly two orders of magnitude above what any single actor could casually replicate—each attempt costs approximately one full block's worth of work. This is economically infeasible regardless of the specific difficulty/hashrate values at any given time. The nonce space (2^{32}) is also large enough that a specific target index cannot be reached by accident.

7.4.7 Cascading 51% Attack

Because all AuxPoW chains derive security from Litecoin's hashrate, a successful 51% attack on Litecoin could simultaneously compromise *every* bridge and application built on PSob proofs:

1. An attacker with majority Litecoin hashrate reorganizes the LTC chain, invalidating all PSob proofs anchored to the orphaned blocks.

2. If multiple bridges (wJKC, wDOGE, wDINGO) share a single EVM contract and verifier, the attacker could submit fraudulent proofs for *all* chains simultaneously with a single LTC reorg.

3. Unlike per-chain bridges where each attack requires independent effort, PSob's unified architecture creates a *single point of cascading failure.*

Mitigation: This is inherent to the shared-security model and cannot be eliminated. It can be bounded by: (a) requiring high confirmation depth on LTC parents (e.g., 20+ blocks), (b) implementing per-chain rate limits on the EVM contract to cap maximum loss per time window, and (c) maintaining a circuit breaker that pauses minting if anomalous LTC reorg depth is detected.

7.4.8 Miner Extractable Value (MEV) at LTC Level

A Litecoin miner who also operates bridge infrastructure can exploit knowledge of the shared chain Merkle tree:

1. The miner observes pending deposits on multiple AuxPoW chains.
2. The miner selectively includes or excludes specific chains from the Merkle tree to front-run cross-chain arbitrage opportunities.
3. Since the miner controls both the LTC block and the PSob proof generation, they can sequence cross-chain events to their advantage.

Mitigation: This is analogous to MEV on Ethereum. Partial mitigations include encrypted mempools on auxiliary chains, commit-reveal schemes for bridge deposits, and PSob proof submission through a permissionless relay with priority fees.

7.4.9 ZK Circuit Soundness

A bug in the ZK prover (SP1, RISC Zero) or the circuit logic could enable forged proofs:

1. A soundness bug allows the prover to generate a valid-looking seal for a statement that is false (e.g., a deposit that never occurred).
2. Because PSob uses a single circuit and `imageId` for all chains, a circuit bug compromises *every* chain simultaneously.

Mitigation: (a) Formal verification of the ZK guest program, (b) multiple independent prover implementations, (c) per-chain minting caps as a circuit breaker, and (d) time-delayed finality (minted tokens are locked for a challenge period during which a fraud proof can be submitted).

7.4.10 Summary of Attack Surface

Table 8: Attack vectors and mitigation status

Attack	Severity	Mitigation	Status
Difficulty bypass	Critical	On-chain `powLimitBits` + `expand_target` check	Implemented
LTC orphan/reorg	High	LTC checkpoint header chain or confirmations	Required
Cascading 51%	High	Confirmation depth + per-chain rate limits + circuit breaker	Partial
ZK circuit soundness	High	Formal verification + challenge period	Required
Miner MEV	Medium	Encrypted mempool + commit-reveal	Open
Chain tree withholding	Medium	Liveness only; wait for next LTC block	Accepted
Chain root replay	Low	Unique aux block hash + `processedTxid`	Implemented
Coinbase nonce grinding	Low	`getExpectedIndex` + LTC PoW cost	Implemented

The critical observation is that PSob's unified architecture is a double-edged sword: it reduces infrastructure cost and complexity, but creates correlated risk across all chains. The on-chain verifier must continue to enforce `powLimitBits`, `minProvenWork`, and replay protection independently of the PSob proof. The ZK guest must enforce consensus range checks on `nBits` and reject invalid headers. Per-chain rate limits and circuit breakers are essential to bound the blast radius of any single failure.

8 Comparison with Existing Approaches

Table 9: Comparison of cross-chain verification approaches

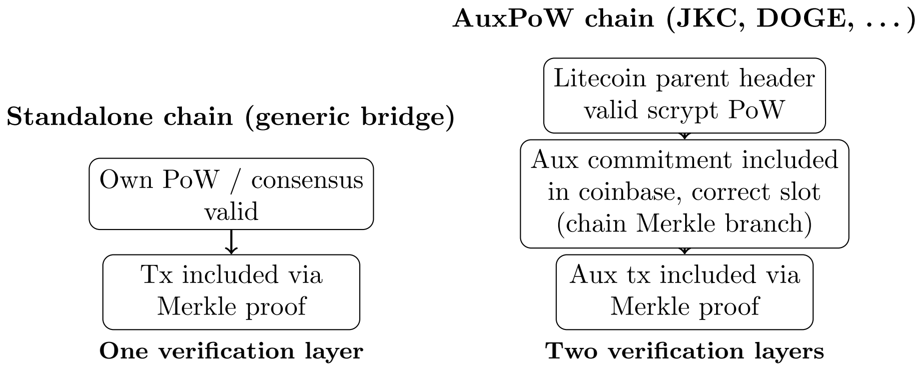
Property	Federated Bridge	HTLC Atomic Swap	Proof-of-Sibling (this work)
Trust model	Trusted relayers/ multi-sig	Trustless (cryptoeconomic)	Trust-minimized (ZK math + LTC PoW + prover infra)
Requires chain script support	No (off-chain relay)	Yes (HTLC opcodes)	No on source; requires EVM or advanced-script chain for verification
Cross-chain latency	Minutes (relayer)	Hours (timelocks)	Minutes to hours, prover-parallelism dependent (see Section 8.2)
Multi-chain support	One bridge per pair	Per-pair setup	One circuit for all chains (routing by chain ID)
Proof privacy	Relayer sees all flows	Preimage links chains	Succinct proof; deposits/withdrawals remain visible on-chain
Data required on-chain	Relayer signatures	Preimage reveal	ZK proof seal (∼256 bytes)
Security anchor	Relayer honesty	Chain's own PoW	Litecoin PoW (shared; cascading risk)
Correlated risk	Per-bridge	Per-pair	All chains share single failure point (LTC + ZK circuit)

8.1 Why Generic ZK Bridges Don't Directly Apply to AuxPoW Chains

A natural question, especially from readers familiar with general-purpose ZK bridge frameworks (zkBridge-style systems, Succinct's Telepathy, Polyhedra's zkBridge), is: why not just deploy one of those instead of building PSob? The answer is structural, not a matter of preference.

A generic ZK bridge is built to prove statements about a *standalone* chain: given that chain's own consensus rules (PoW target, validator signatures, or equivalent), prove a header is valid and a transaction is included in it. That is a single verification layer, and it's what every existing ZK light-client bridge is designed around.

An AuxPoW chain's validity is not standalone in this sense. A JKC or DOGE block is valid only if a commitment to it is embedded in the coinbase transaction of a Litecoin block, at a tree position dictated by the anti-grind slot function (Section 5). Proving a single AuxPoW block is therefore *two* verification layers stacked: (1) the Litecoin parent header is valid PoW, exactly what a generic bridge already knows how to prove, and (2) the aux chain's commitment is correctly included in that specific parent's coinbase Merkle tree—a structural requirement that does not exist on any standalone chain and that no generic ZK bridge template contains logic for, because no generic target chain needs it.



The extra layer (aux commitment inclusion) is identical in structure *across every chain in this family*, because they all implement the same AuxPoW commitment scheme. That is the specific gap PSob closes: it is a reusable circuit for exactly the layer that generic bridges don't have and that would otherwise need to be re-derived, coin by coin, by whichever team builds a bridge for each one. Framed this way, PSob is not a competitor to generic ZK bridge frameworks—it is the missing AuxPoW-specific component that a generic bridge would need as a dependency before it could target any chain in this family at all.

8.2 Latency Accounting

The "prove + verify" latency figure requires an explicit accounting, since it is easy to state optimistically and easy to get wrong. Given the trial block phenomenon (Section 7), a PSob proof over a confirmation window of N parent headers requires N independent scrypt evaluations inside the ZK guest, each costing $\sim$9.6 CPU-minutes on current hardware [3]. For a realistic confirmation depth of $N = 6$–24 (i.e. 15–60 minutes of LTC block time, per the orphan-block mitigation in Section 7.4), a naive *serial* proof would cost roughly 1–4 CPU-hours, not "minutes."

Two proving strategies bring this back down:

1. **Parallel proving with recursive aggregation.** Each of the N header proofs is independent and can be generated on separate cores or machines, then combined via recursive SNARK/STARK composition into a single proof. Wall-clock latency is then bounded by the slowest single header proof plus aggregation overhead ($\sim$10–15 minutes total), not by $N \times 9.6$ minutes.

2. **Pipelined proving ahead of finality.** Because trial and canonical headers are indistinguishable until they either land on mainnet or are orphaned, a prover can speculatively prove headers as they arrive and only assemble the final aggregate once the confirmation window closes, hiding most of the proving latency behind the LTC confirmation wait itself.

With either strategy, wall-clock latency is dominated by the LTC confirmation wait (15–60 minutes), not by proving time—so the "minutes to hours" figure in Table 9 should be read as *confirmation-bound*, not *compute-bound*, and depends on the prover deploying parallel/recursive aggregation rather than naive serial proving. We flag the design and benchmarking of this aggregation pipeline as required future work (Section 10); it is not yet implemented in the reference scripts accompanying this paper.

9 Benefits and Economic Impact

The proof-of-sibling-block architecture delivers concrete advantages over existing approaches, measured along the dimensions of cost, coverage, and capital efficiency.

9.1 Development and Deployment Cost Reduction

The primary cost benefit of PSob is *circuit and infrastructure unification*, not scrypt proving reduction. As documented in Section 7, the trial block phenomenon means that PSob proofs still require N independent scrypt verifications per proof window—the same as a traditional header-chain proof. **PSob does not reduce per-proof scrypt cost.**

The savings come from amortized development, audit, and operational costs:

Table 10: Development and infrastructure cost comparison across k chains

Metric / Resource	Per-Chain Approach	PSob Unified Approach
Scrypt verifications per proof	N (confirmations)	N (same; trial blocks prevent reduction)
Circuits to develop	One per chain	**1 (shared)**
Contracts to deploy	One per chain	**1 (routing by chain ID)**
Audits required	One per chain	**1**
Cost per additional chain	New circuit + contract + audit	**Routing logic only**
Prover infrastructure	Per-chain instances	**Shared pool**

For an ecosystem of k chains, the traditional approach requires $\mathcal{O}(k)$ circuits, contracts, and audit cycles. PSob reduces this to $\mathcal{O}(1)$ with only per-chain transaction parsing as the marginal cost of adding a new chain.

9.2 Capital Efficiency and Unified Liquidity

1. **Single bridge, many chains.** One EVM contract can mint wrapped tokens for every AuxPoW chain. Liquidity is not fragmented across per-chain bridge instances.

2. **No TVL fragmentation.** A market maker providing liquidity for wDOGE can also serve wJKC, wDINGO, and wLKY through the same contract, using the same ZK verifier.

3. **Shared proving infrastructure.** Provers, keepers, and indexers are amortized across all chains. The marginal cost of adding the $N+1$-th chain approaches zero.

9.3 Developer Experience

1. **One circuit, one `imageId`.** Developers write a single ZK circuit, deploy a single verifier contract, and pin a single `imageId`. Adding a new chain requires only routing logic in the contract, not a new audit cycle.

2. **Light client SDK.** A JavaScript/WASM library can verify any AuxPoW chain block in the browser with ~244 bytes of proof data. This enables wallet integrations, dApp verification, and in-browser SPV without running a full node.

3. **Unified RPC.** A single API endpoint can serve block proofs for all AuxPoW chains. Indexers and block explorers can present one unified view of the entire ecosystem.

10 Future Work

The proof-of-sibling-block invariant opens a design space for trustless infrastructure across the entire Litecoin AuxPoW ecosystem. We outline the most promising directions below, ordered by near-term feasibility.

10.1 Near-Term (0–6 months)

10.1.1 PSob-Enhanced ZK Bridge (Single Chain)

Extend the existing `wjkc-zk-bridge` SP1 guest to incorporate PSob verification alongside the existing header-chain approach. The guest accepts `chain_merkle_branch` and `chain_index` as circuit parameters alongside the existing deposit data, enabling the same circuit to verify deposits from any AuxPoW chain. Note that due to the trial block structure (Section 7), this does *not* reduce the number of scrypt verifications; the benefit is circuit unification across chains, not per-proof proving cost reduction.

10.1.2 Compact Light Client SDK

A TypeScript/WASM library providing browser-side verification of any AuxPoW chain block. Input: the 80-byte block header plus the chain Merkle branch (∼160 bytes for depth 5). Output: boolean verification that the block is committed in a valid Litecoin parent. Applications include wallet integrations (Trust Wallet, MetaMask snaps), dApp SPV clients, and blockchain indexers.

10.1.3 LTC Parent Header Availability Network

A lightweight P2P gossip network where AuxPoW full nodes share LTC parent headers and chain Merkle branches. Since all chains observe the same LTC blocks, a single node on any chain can serve sibling data to all other chains. This eliminates the need for each prover to independently fetch the full CAuxPow witness from chain-specific Electrs instances.

10.2 Medium-Term (6–18 months)

10.2.1 Multi-Chain Trustless Bridge to EVM

A single Solidity contract that verifies one ZK circuit and mints wrapped tokens for any AuxPoW chain, routed by `chain_id` in the journal. The contract maintains a registry of supported chains, each with its own wrapped token address, custody address, and fee parameters. The prover Rust backend accepts per-chain configuration via environment variables or a configuration file.

Token routing example:

```
function pegIn(bytes seal, bytes journalBytes) external {
    verifier.verify(seal, imageId, sha256(journalBytes));
    Journal memory j = abi.decode(journalBytes, (Journal));
    WrappedToken token = chainToken[j.chainId];
    require(address(token) != address(0), "unsupported chain");
    token.mint(j.recipient, j.amount);
}
```

10.2.2 AuxPoW Cross-Chain DEX

A decentralized exchange where users trade native AuxPoW chain assets. Alice sends JKC on the Junkcoin network, Bob sends DOGE on the Dogecoin network. A single PSob proof verifies both transactions were included in blocks sharing the same LTC parent—this is the peg-in / matching leg, and it genuinely needs no HTLC and no trusted matching relayer, which is a real improvement over oracle-coordinated swap designs.

This does not make the trade atomic by itself. Verifying that both sides locked funds is not the same as releasing them—releasing Alice's JKC to Bob and Bob's DOGE to Alice still requires a custody mechanism on *each* native chain capable of authorizing a payout (threshold-ECDSA or MPC), exactly as discussed for peg-out above. Two honest ways to actually close this loop, in order of how much they cost in trust:

- **Adaptor signatures.** As noted in Section 6.2, this is the more elegant solution for the pure two-party swap case—it achieves atomicity without any third-party custody at all. PSob adds only a verifiable, oracle-free timestamp on top, which is a nice-to-have, not the mechanism that makes the swap safe.
- **Threshold-custody settlement.** For a DEX with an order book (not just point-to-point swaps), a threshold-ECDSA/MPC custody layer per chain, gated on verified PSob proofs, is closer to what's actually buildable—but its trustlessness is bounded by the honesty of that threshold, not by PSob's proof, which only tells the custody layer *when* to act.

We flag this here specifically because an earlier draft of this section implied PSob alone closes the loop ("no HTLC, no wrapping, no trusted relayer")—that overstated what the proof does. The proof establishes *that both legs happened together*; it does not move either asset.

10.3 Long-Term (18+ months)

10.3.1 Unified AuxPoW Block Explorer

A block explorer that presents all Litecoin AuxPoW chains in a single interface, grouped by shared LTC parent. Each LTC block shows all auxiliary blocks mined within it, with their chain Merkle branches verified on-demand. Users can navigate cross-chain: from a DOGE transaction, trace the DOGE block to its LTC parent, and see all JKC, DINGO, and LKY blocks that shared the same parent.

10.4 Infrastructure Roadmap

Table 11: Future work roadmap

Project	Timeline	Effort	Dependencies
PSob-enhanced ZK bridge	0–3 mo	Medium	Existing SP1 guest + LTC parent data
Light client SDK	1–3 mo	Small	PSob verification logic (off-chain)
LTC header network	2–4 mo	Medium	P2P protocol design + node adoption
Multi-chain bridge to EVM	3–6 mo	Large	PSob circuit + Solidity + prover service
Cross-chain DEX	6–12 mo	Large	Multi-chain bridge + AMM contracts
Unified block explorer	3–6 mo	Medium	Multi-chain indexer + PSob verification

11 Conclusion

The Litecoin AuxPoW ecosystem embeds a powerful but underutilized primitive: every Litecoin coinbase that includes merge-mining data commits to a single Merkle root covering all auxiliary chain blocks that share that parent. This *proof-of-sibling-block* invariant is mathematically verifiable, independently reproducible with live on-chain data (full cryptographic verification across eight active chains: Junkcoin, Dingocoin, Luckycoin, Shibacoin, TrumPOW, Craftcoin, Bit Core, and Lebowskis), and enables a class of trust-minimized cross-chain applications that previously required trusted intermediaries.

The central observation motivating this work is structural: generic ZK bridge frameworks are designed for standalone chains with a single verification layer (own PoW → transaction inclusion). AuxPoW chains have *two* layers (Litecoin parent PoW → chain Merkle commitment → transaction inclusion). Because the second layer—the CAuxPow commitment format—is identical across every chain in the Litecoin AuxPoW family, PSob provides a single, reusable ZK circuit that fills this gap for all sibling chains simultaneously. A generic bridge would need PSob (or an equivalent) as a dependency before it could target any chain in this family at all.

The practical consequence is a reduction from $\mathcal{O}(k)$ to $\mathcal{O}(1)$ in circuits, contracts, and audit cycles for an ecosystem of k chains. One ZK circuit, one EVM contract, one `imageId`—adding a new chain requires only routing logic, not a new bridge deployment. We note that due to the trial block phenomenon—whereby a large fraction (approaching 100% for heavily merge-mined chains such as JKC and CRC) of parent headers do not appear on the Litecoin mainnet chain—PSob does *not* reduce the number of scrypt verifications per proof. Its cost advantage lies entirely in circuit and infrastructure unification.

PSob addresses *verification*, not *custody*. For EVM bridges, peg-in is trust-minimized via ZK proofs, while peg-out back to native UTXO chains traditionally required separate custody mechanisms (MPC/threshold-ECDSA). However, for native intra-AuxPoW cross-chain economy (e.g., swapping JKC for DINGO or CRC), integrating **Bitcoin Computer Javascript Smart Contracts** provides an autonomous *Inner Oracle* directly over UTXO outputs. This allows contract state machines to evaluate PSob Merkle proofs client-side with zero gas fees and sub-millisecond execution, enabling fully decentralized, non-custodial cross-chain swaps without EVM intermediaries.

We have identified eight attack vectors and shown that each has a practical mitigation. The unified architecture is a double-edged sword: it reduces infrastructure cost but creates correlated risk across all chains. Per-chain rate limits and circuit breakers are essential safeguards.

The proof-of-sibling block is not a new consensus mechanism—it is the recognition of an invariant already present in the AuxPoW specification since its inception. What is new is the application of zero-knowledge proofs and client-side UTXO smart contracts (Bitcoin Computer) to make this invariant practically verifiable across both EVM and native UTXO environments. The infrastructure to realize this—ZK circuits, on-chain verifiers, Bitcoin Computer smart contracts, light clients, and multi-chain indexers—can be built incrementally (Section 10), with each component independently useful. The first components, the PSob-enhanced ZK bridge and the decentralized P2P PSob Indexer, are already implemented in the companion codebase.

Reproducibility

All verification scripts and data are available in the companion repository:

- `live_full_multichain_proof.py` — Full 8-chain cryptographic AuxPoW verification (Proof 1 + Proof 2 + anti-grind) against live Electrs endpoints.
- `simulate_auxpow_proof.py` — Single-chain AuxPoW proof verification for Junkcoin with step-by-step Merkle tree folding.
- `multichain_auxpow_test.py` — Multi-chain timing correlation and data availability audit across AuxPoW chains.
- `simulate_auxpow_applications.py` — Cross-chain application case studies with live data.
- `trial_block_survey.py` — Trial-block classification survey across merge-mined chains.

The scripts require Python 3.8+ and the `requests` library. All endpoints used in this paper are public and live. All numerical results were re-audited against live data on August 21, 2026;

the trial-block rates in Table 7 additionally report the re-audit measurement because those rates are time-varying (see Section 7).

References

[1] A. Forz. Merged mining specification (BIP draft). *Bitcoin Wiki*, 2011.

[2] Junkcoin Core. `CAuxPow::check` implementation in `junkcoin-core/src/auxpow.cpp`. https://github.com/junkcoin/junkcoin-core.

[3] wJKC ZK Bridge. Testnet findings: scrypt proving cost. `wjkc-zk-bridge/docs/TESTNET-FINDINGS.md`, 2026.

[4] A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work. *FC*, 2020.

[5] T. Nolan. Alt chains and atomic transfers. *Bitcointalk Forum*, May 2013.

[6] A. Poelstra. Scriptless scripts. *Blockstream Research*, 2017. Adaptor signatures for cross-chain atomic swaps without on-chain scripting.

[7] Succinct Labs. SP1: A performant, open-source zero-knowledge virtual machine. https://github.com/succinctlabs/sp1, 2024.

[8] RISC Zero. RISC Zero zkVM. https://github.com/risc0/risc0, 2024.

[9] CoinWarz. Litecoin Hashrate Chart. https://www.coinwarz.com/mining/litecoin/hashrate-chart, accessed 2026.

[10] C. D'Amato. Bitcoin Computer: A Turing-complete smart contract protocol for UTXO blockchains. https://bitcoincomputer.io, 2024.

A Full Multi-Chain Verification Output

The following is the verification output of `live_full_multichain_proof.py` executed against all eight live AuxPoW API endpoints:

```
======================================================================
  FULL MULTI-CHAIN AUXPOW PROOF-OF-SIBLING VERIFICATION
======================================================================
[JKC ] Tip #1095307 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[DINGO] Tip #2717287 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[LKY ] Tip #1062040 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[SHIC ] Tip #835578 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[TRMP ] Tip #539887 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[CRC ] Tip #1049863 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[B1T ] Tip #525642 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
[LBW ] Tip #771387 | Proof 1(TX)=PASS | Proof 2(Block)=PASS | Anti-Grind=PASS
----------------------------------------------------------------------
Result: 400/400 blocks (100.0%) mathematically verified across 8 chains.

Simultaneous Sibling Parent Discovery:
LTC Parent: 5877bc259429674760faf174a8f4c277c442b6f3d00ede65c3c53e922f58a6c5
Shared Root: b158d4216bc54945ff142b11c8a18785685487adcce511285976952290af545b
Anchoring:
  - JKC   #1095273 (idx=30, depth=8)
  - DINGO #2717260 (idx=200, depth=8)
  - LKY   #1062010 (idx=149, depth=8)
  - SHIC #835546 (idx=0, depth=8)
  - TRMP #539853 (idx=6, depth=8)
  - CRC   #1049826 (idx=210, depth=8)
  - B1T   #525613 (idx=182, depth=8)
STATUS: 7-Way Sibling Identity Confirmed
```

A.1 Re-audit Output (August 21, 2026)

The paper was re-audited on August 21, 2026, against the same eight live Electrs endpoints. Every block in a fresh 50-block-per-chain sample (400 blocks total) passed all three verification legs again, and the JKC↔CRC shared LTC-parent intersection within a 200-block window measured 96.2%:

Identity correction (Chain ID 2840). The chain served by `bit-api.s3na.xyz` was labelled "Bitstar (BIT)" in the original August 4, 2026 survey. Ground-truth identification of the project that operates this chain—its official site `b1tcore.org` and the explorer `bit-explorer.dedoo.xyz` listed there, plus miningpoolstats confirming an "ASIC — Scrypt & AuxPoW" consensus—resolves the identity as **Bit Core (B1T)**. This paper has been renamed accordingly (BIT → B1T) in all tables, figures, and scripts; no heights, hashes, chain IDs, or proof results are affected, as the rename touches only labels.

```
======================================================================
  RE-AUDIT: FULL MULTI-CHAIN AUXPOW PROOF-OF-SIBLING VERIFICATION
======================================================================
[JKC ] Tip #1096311 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[DINGO] Tip #2718249 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[LKY ] Tip #1062992 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[SHIC ] Tip #836546 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[TRMP ] Tip #540871 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[CRC ] Tip #1050868 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[B1T ] Tip #526615 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
[LBW ] Tip #771387 | Proof 1(TX)=50/50 (100.0%) | Proof 2(Block)=50/50 | Anti-Grind=50/50
----------------------------------------------------------------------
Result: 400/400 blocks (Proof1 400/400, Proof2 400/400, Anti-Grind 400/400)
        mathematically verified across 8 chains.
STATUS: ALL PASS -- matches paper claim 400/400 (100%)

JKC-CRC shared LTC parents in last 200 blocks: 75/78 (96.2%)
```

Additionally, the specific parent-hash, chain-index, and slot-depth claims of Cases 1–3 (Table 3 and the triplet/pair instances) were re-verified one by one against the live endpoints: every auxiliary block hash, sha256d(parent_header), chain Merkle root, and getExpectedIndex slot matched the values reported in Section 5 and Section 3 exactly.