

Composable Bitcoin Verification for Genesis-Anchored Light Clients

December 2025

Abstract

This paper explores how Schnorr signatures and Bitcoin Simple Payment Verification (SPV) compose on top of a genesis-anchored lineage verification system to enable trust-minimized verification of external facts by resource-bounded light clients. We assume a network whose genesis is cryptographically anchored to Bitcoin, establishing an immutable identity and temporal origin. Building on this foundation, we demonstrate how Schnorr signatures enable compact, aggregatable attestations and how SPV proofs allow light clients to verify Bitcoin events without replaying execution or trusting intermediaries.

No new cryptographic primitives are introduced. The contribution is architectural: demonstrating that once network identity is fixed at genesis, the composition of Schnorr and SPV forms a minimal verification layer suitable for browser-native and intermittently offline clients. This approach is underexplored in existing blockchain architectures, where verification layers typically depend on global execution replay or trusted intermediaries.

Non-Goals

This work does not present a general-purpose smart contract virtual machine. The architecture does not execute Bitcoin Script or provide Turing-complete on-chain computation. Rather, it focuses on bounded verification of external facts under resource constraints typical of light clients. The goal is to establish *what* external events occurred and *who* attests to their meaning, not to replicate arbitrary execution environments. This design choice trades expressiveness for verification efficiency and light-client accessibility.

CRITICAL LIMITATIONS: This system does **not** provide: (1) censorship resistance, (2) liveness guarantees, (3) execution correctness guarantees, (4) fraud proofs, (5) slashing mechanisms, or (6) trustless bridging in the conventional sense. It verifies *facts* (that specific Bitcoin events occurred), not *behaviors* (that programs executed

correctly). Signers can withhold attestations. Data can be censored. The system halts if attestations are unavailable. This is a **verification primitive**, not a security protocol.

1. Preliminaries and Assumptions

1.1 Genesis-Anchored Trust Root

We assume the existence of a genesis-anchored lineage verification primitive satisfying:

- A unique genesis state G
- A cryptographic commitment C_G embedded in Bitcoin at block height h
- **Immutability**: no alternative genesis can exist without contradicting Bitcoin consensus

Formally:

$$\forall G', G' \neq G \Rightarrow C_{G'} \notin \text{Bitcoin}[h]$$

This establishes:

- Network identity
- Lower bound on genesis time
- A trust root external to the network itself

This paper assumes this anchoring result and focuses on the verification primitives that compose upon it. The genesis anchoring mechanism itself is not re-derived here.¹

¹ **Genesis Anchoring Scope:** The correctness of genesis anchoring to Bitcoin is an external prerequisite to this work and must be independently verified. Validation of the anchoring commitment C_G at block height h requires external proof mechanisms (e.g., Bitcoin block explorer verification, community consensus validation, or cryptographic attestation from trusted sources). If genesis anchoring is disputed or incorrect, this entire architecture is invalidated. This paper does not provide genesis anchoring proofs—it assumes their correctness as a foundational axiom.

1.2 Trust Model and Assumption Ranking

This architecture's trust model is **strictly weaker** than Bitcoin's native verification. We rank assumptions by severity:

Assumption	Severity	Consequence if Violated
Bitcoin PoW security	STRONG	Complete system failure
Genesis anchoring correctness	STRONG	Complete system failure
Threshold (t/n) signer honesty	MODERATE	False attestations possible
Data availability	MODERATE	Liveness failure (not soundness)
Signer availability	WEAK	Service degradation
Discrete log hardness	STRONG	Signature forgery possible

Comparison with Bitcoin Native Verification:

Property	Bitcoin Native	This System
Trust Requirements	PoW majority honest	PoW + (t/n) signers + DA + G valid
Verification Cost	Full UTXO validation	O(k) headers + O(log n) Merkle
Censorship Resistance	Yes (PoW based)	No (signer dependent)
Liveness Guarantee	Yes (mining incentives)	No (availability assumption)
Script Execution	Yes (Bitcoin Script)	No (verification only)

Formally, the trust sets satisfy $\text{Trust}_{\text{Bitcoin}} \subset \text{Trust}_{\text{System}}$, making this system's guarantees strictly weaker than Bitcoin's native verification.

1.3 Threat Model

Adversaries may:

- Control network peers
- Withhold data (degrading liveness)
- Attempt equivocation across forks
- Attempt to forge attestations
- Collude among signers (if threshold t is compromised)

Adversaries may not:

- Break Bitcoin PoW security
- Forge Schnorr signatures under standard discrete logarithm assumptions
- Break collision-resistant hashing

2. Why Schnorr Matters After Genesis

2.1 Schnorr Signature Properties

Let $(x, P = xG)$ be a Schnorr keypair over an elliptic curve group.

A Schnorr signature $\sigma = (R, s)$ over message m satisfies:

$$sG = R + H(R \parallel P \parallel m)P$$

Properties critical for light clients:

1. Linearity

$$\sum s_i G = \sum R_i + H(\cdot) \sum P_i$$

2. Aggregation: Multiple signatures over the same message can be compressed into a single constant-size proof.

3. Non-interactive verification: No challenge rounds, no on-chain execution.

These properties are well-established in the literature. Our contribution is demonstrating how they compose with Bitcoin SPV in the context of a genesis-anchored identity.

2.2 Why This Is Not Sufficient Alone

Without a trusted genesis:

- A Schnorr signature proves authorization, not contextual binding
- The signer's identity is ambiguous across chains or forks
- Fork-relative semantic meaning is undefined

Genesis anchoring resolves this ambiguity by fixing network identity at inception, allowing signatures to be meaningfully interpreted within a single, well-defined lineage.

3. Bitcoin SPV as a Fact Verification Primitive

3.1 SPV Proof Definition

An SPV proof for a Bitcoin transaction tx consists of:

1. Bitcoin header chain $H_0 \rightarrow H_n$
2. Merkle path $\pi(tx)$

3. Proof-of-work validation

Verification cost:

$O(\log n)$ storage for Merkle path, $O(k)$ header validation

where k is a policy-bounded confirmation depth. In practice, light clients use checkpoints or bounded verification windows to avoid linear header chain validation. No script execution is required.

3.2 What SPV Proves

SPV proves:

$\exists h, tx \in \text{Block}(h) \wedge \text{PoW}(h) \geq \tau$

It does not prove:

- Script correctness beyond inclusion
- Absence of reorgs beyond depth k
- Full node validation rules (UTXO set integrity, double-spend prevention)

These limitations are inherent to SPV and are well-documented in the Bitcoin literature. Our architecture accepts these tradeoffs in exchange for light-client efficiency.

3.3 Data Availability Assumptions and Failure Modes

The verification function $V(E)$ is only computable if proof components are available:

$\text{Computable}(V) \blacksquare \text{Available}(\text{headers}) \wedge \text{Available}(\text{merkle_path}) \wedge \text{Available}(\text{signatures})$

Failure Modes:

- **DA failure** \rightarrow **System unavailability**: If Bitcoin headers, Merkle proofs, or attestations are withheld, verification cannot proceed. The system halts without producing incorrect results.
- **Signer unavailability** \rightarrow **Service degradation**: If fewer than t signers are online, attestations cannot be generated. This degrades liveness but does not compromise soundness.
- **Network partition** \rightarrow **Delayed verification**: Proof relay delays increase latency but do not produce false positives.

Critical distinction: DA failures result in *unavailability*, not *incorrectness*. The system cannot produce false verifications due to missing data—it can only fail to verify. This is acceptable for a verification primitive but unacceptable for a liveness-critical protocol.

4. Composition: Genesis + SPV + Schnorr

4.1 Composed Verification Statement

A light client verifies the following statement:

"A Bitcoin event E occurred at or after the genesis-anchored block, and a quorum of authorized signers attests to its semantic interpretation for this network."

Formally:

$$\text{Verify}(G) \wedge \text{SPV}(E) \wedge \text{SchnorrAgg}(E)$$

Where:

- G is the anchored genesis
- E is a Bitcoin fact
- SchnorrAgg binds semantic meaning to E , not execution

4.2 Resource Requirements for Light Clients

A verifier must store only:

- $O(1)$ genesis commitment
- $O(k)$ recent Bitcoin headers (bounded by policy)
- $O(1)$ Schnorr aggregate signature per attestation

This eliminates the need for global execution replay, full historical state, or trusted RPC endpoints. Verification remains computationally feasible for browser-native clients and intermittently connected devices.

4.3 Quantitative Performance Bounds

We provide rough estimates for proof sizes, verification costs, and latency. Concrete values depend on implementation choices and policy parameters.

Proof Size:

$$\text{Size}(\text{proof}) = k \times 80 \text{ bytes (headers)} + \log_2(n) \times 32 \text{ bytes (Merkle)} + 64 \text{ bytes (signature)}$$

Example: $k = 6$ confirmations, $n = 2048$ transactions:

$$\text{Size} = 6 \times 80 + 11 \times 32 + 64 = 896 \text{ bytes}$$

Verification Time:

$$\text{Time}(\text{verify}) = k \times \text{SHA256} + \log_2(n) \times \text{SHA256} + \text{Schnorr_verify}$$

On modern hardware: $\approx 17 \times \text{SHA256} + 1 \times \text{Ed25519 verification} \approx 0.1\text{--}0.5 \text{ ms}$

End-to-End Latency:

$$\text{Latency} = \text{Bitcoin_confirmation_time}(k) + \text{Attestation_propagation} + \text{Verification_time}$$

Example: $k = 6$ blocks ≈ 60 minutes + network propagation (seconds to minutes) + verification (<1 second)

These estimates establish feasibility for browser-native and resource-bounded clients. Production implementations may optimize further using header compression, batched verification, or checkpoint schemes.

5. Concrete Bitcoin-Verified Contract Examples

The verification primitives established in Sections 2–4 enable trust-minimized enforcement of Bitcoin-derived state transitions. We present three illustrative use cases: oracles, verified mints, and escrow. In each case, the system verifies *facts* (via SPV) and optionally *semantic meaning* (via threshold signatures), but does not execute Bitcoin Script or rely on trusted data feeds. These examples demonstrate architectural patterns, not exhaustive implementations.

Terminology Note: These are **Bitcoin-verified predicates**, not Bitcoin smart contracts in the traditional sense. They do not execute on Bitcoin; they use Bitcoin as a timestamped, immutable fact source.

5.1 Oracle Example

Scenario: A Bitcoin transaction embeds a price commitment using a Taproot output with a leaf script containing $H(\text{price_data})$. When the commitment is revealed on-chain, the SPV proof demonstrates inclusion.

Verification Flow:

1. **Bitcoin Layer:** Transaction tx_{oracle} is included in block h_B with confirmation depth N .
2. **SPV Proof:** Light client receives $\pi_{\text{BTC}} = (\text{txid}, \text{merkle branch}, \text{header chain segment}, \text{depth } N)$.
3. **Semantic Attestation:** A threshold t -of- n signer set provides a Schnorr aggregate signature over the statement: " tx_{oracle} at height h_B commits to BTC/USD = \$42,000."
4. **State Enforcement:** Upon verification of both SPV proof and attestation, the network updates its oracle contract state.

Limitations: This does not prevent data withholding, signer collusion, or censorship. Signers control attestation availability. This sacrifices liveness guarantees for cryptographic verifiability.

5.2 Verified Mint / Redemption Gate Example

Scenario: A user locks BTC in a time-locked Bitcoin output or burns it to an OP_RETURN. The goal is to mint an equivalent asset on the verification network.

Verification Flow:

1. **Bitcoin Layer:** Transaction tx_{lock} locks X BTC in a specified output format.
2. **SPV Proof:** π_{BTC} proves tx_{lock} exists in block h_B with depth $\geq N$.

3. **Semantic Attestation (Optional):** Operators may co-sign to attest that the lock conforms to expected rules (e.g., output script matches expected format).

4. **State Enforcement:** The network mints X wrapped BTC tokens to the user's account, authorized by SPV verification rather than custodial trust.

Critical Distinction from Trustless Bridges: This is **not** a trustless bridge. It is a **verification-gated custodian** or **attestation-gated mint**. It lacks:

- Fraud proofs (no challenge mechanism)
- Slashing (no economic penalties for misbehavior)
- Forced exits (no on-chain dispute resolution)
- Automatic reversion (no rollback on invalid mints)

Signers may collude to mint without valid locks. Data availability failures prevent legitimate mints. This architecture derives mint authority from Bitcoin SPV + threshold attestation, which is weaker than Bitcoin's native security.

5.3 Escrow Example

Scenario: Two parties agree to a Bitcoin-based escrow with a Taproot script tree: Alice can spend after timelock T , Bob can spend with hash preimage h , or both can spend cooperatively.

Verification Flow:

1. **Bitcoin Layer:** Escrow output is created in tx_{escrow} . When the condition is satisfied (e.g., Bob reveals the preimage), tx_{spend} spends the output.
2. **SPV Proof:** π_{BTC} proves tx_{spend} is confirmed in block h_B with depth $\geq N$.
3. **Semantic Attestation:** Signers attest that tx_{spend} satisfies the agreed condition (e.g., "Bob provided valid preimage").
4. **State Enforcement:** Upon verification, the network releases escrowed assets to Bob's account.

Key Property: Enforcement logic resides in the verification network, while the *trigger condition* is proven via Bitcoin's ledger. This separates concerns: Bitcoin provides the timestamped, censorship-resistant fact; the network interprets and enforces the consequence. Script execution remains on Bitcoin; only the *result* is verified.

5.4 Positioning Relative to Existing Systems

We position this verification-first architecture relative to established approaches. This comparison is intended to clarify architectural tradeoffs, not to claim categorical superiority.

Ethereum L1 Smart Contracts: Ethereum executes arbitrary logic on-chain via the EVM. Every node replays every transaction to maintain consensus. Light clients must trust sync committees or verify zkSNARKs of execution traces. This architecture verifies Bitcoin facts via SPV and interprets meaning via Schnorr attestations—no global execution replay required. Trade-off: Ethereum provides composable on-chain logic; this approach provides efficient verification of external facts with bounded resources.

Rollups (Optimistic and ZK): Rollups post state commitments to Ethereum L1 and inherit its security assumptions. They depend on L1 data availability, sequencer liveness, and (for optimistic rollups) fraud proof windows. This architecture does not depend on a base layer's execution environment—only on Bitcoin's timestamping and PoW finality. Trade-off: Rollups offer EVM compatibility and established tooling; this approach offers lighter verification and Bitcoin-native anchoring without base-layer execution dependency.

Babylon-Style Bitcoin Security Borrowing: Babylon enables chains to periodically checkpoint to Bitcoin for finality. This provides ex-post security via Bitcoin timestamps. Genesis anchoring is architecturally distinct: the network's *identity* is fixed at inception, not just its finality checkpoints. Trade-off: Babylon-style systems can retrofit existing chains; genesis anchoring requires commitment at launch and binds identity immutably.

Chainlink Oracles: Chainlink aggregates off-chain data via trusted node operators who sign price feeds. Security depends on operator reputation and economic incentives. The oracle model in Section 5.1 verifies that Bitcoin itself contains the data commitment, then uses threshold signatures to attest meaning—reducing trust in data providers. However, this system sacrifices liveness guarantees and economic incentives for cryptographic verifiability. Trade-off: Chainlink supports diverse, off-chain data sources with SLA guarantees; this approach is limited to Bitcoin-verifiable facts but provides cryptographic inclusion proofs without ongoing incentive requirements.

Summary: This architecture is verification-external, not execution-internal. It trades on-chain programmability for lower light client costs and genesis-anchored trust minimization. The design is suited for applications where Bitcoin's timestamping and PoW finality provide sufficient security guarantees, and where global execution replay is undesirable or infeasible.

5.5 Architectural Diagram

Figure 1 — Bitcoin Event → SPV Proof → Verified State Transition

(Oracle / Verified Mint / Escrow)

Diagram Description for Implementation:

Layout: Left-to-right pipeline with 4 columns, connected by directional arrows. This diagram is explanatory and does not imply performance claims beyond what is explicitly stated in the text.

Column 1: Bitcoin Layer (Source of Fact)

- **Box A:** "Bitcoin Tx (Taproot output / spend path / OP_RETURN if used)"
 - Visual: Rectangle with Bitcoin logo or "■" symbol
 - Sub-labels: "txid: 0x3a7f...", "Output: P2TR script"
- **Box B:** "Block Header + Merkle Root"
 - Visual: Rectangle below Box A
 - Sub-labels: "Height h_B ", "Merkle root: 0x9f2c..."
- **Arrow:** From Box A → Box B, labeled "Tx included in block"
- **Dashed boundary:** Below Column 1, labeled "Bitcoin does NOT execute network logic"

Column 2: SPV Proof Bundle (Evidence Package)

- **Box C:** "SPV Proof $\pi_{BTC} = (txid, merkle\ branch, header\ chain\ segment, confirmations)$ "
 - Visual: Rounded rectangle, distinct color (e.g., light blue)
 - Internal sub-labels (stacked vertically):
 1. "Merkle inclusion: $tx \in block$ "
 2. "PoW chain / difficulty / chainwork"
 3. "Confirmation depth N (policy-bounded)"
- **Arrow:** From Column 1 → Box C, labeled "Extract proof from Bitcoin data"

Column 3: Light Client Verifier (Decision Point)

- **Box D:** "Verifier $V(\pi_{\text{BTC}}, \text{policy}) \rightarrow \{\text{accept, reject}\}$ "

- Visual: Hexagon or diamond shape (decision node), distinct color (e.g., light green)

- Internal checklist (numbered, vertical stack):

1. ✓ Header linkage valid

2. ✓ Difficulty / chainwork rules satisfied

3. ✓ Merkle proof valid

4. ✓ Confirmation threshold met

5. ✓ (Optional) Semantic attestation present (Schnorr threshold signature)

- **Box E:** Connected to Box D via dashed line (side annotation)

- Label: "Local policy: N confirmations, signer set rules, expiry windows, replay protection"

- Visual: Smaller box, off to the side (not in main flow)

- **Note near Box D:** "Verification of facts + optional meaning attestation; not arbitrary computation"

Column 4: Network State Transition (Effect / Settlement)

- **Box F:** "Network state update / account-chain transition"

- Visual: Rectangle, distinct color (e.g., light purple)

- Internal structure: Split into 3 stacked mini-boxes:

1. "**Oracle:** update value if proof accepted"

2. "**Verified Mint:** mint/unlock if BTC lock proven"

3. "**Escrow:** release if condition proven"

- **Arrow:** From Box D \rightarrow Box F, labeled "Accepted proof triggers bounded state update"

- **Dashed boundary:** Encompassing Columns 3–4, labeled "Network does NOT execute Bitcoin Script; it verifies events"

Additional Annotations:

- **Threats Callout Bubble:** Positioned near Column 2–3 boundary

- Visual: Cloud/bubble shape with red or orange outline

- Contents (bulleted list):
- "Withheld proofs / data availability"
- "Reorg risk within N"
- "Signer corruption (if used)"
- "Replay across domains (mitigate with domain tag / unique event ID)"

- **Mitigations Callout Bubble:** Positioned near Box E (policy box)

- Visual: Cloud/bubble shape with green outline
- Contents (bulleted list):
- "Domain separation tag"
- "Finality threshold N"
- "Frontier window / expiry"
- "Multi-source proof retrieval"

Color Scheme Recommendation:

- Column 1 (Bitcoin): Orange/yellow tones
- Column 2 (SPV Proof): Light blue
- Column 3 (Verifier): Light green
- Column 4 (Network): Light purple/violet
- Threats bubble: Red/orange outline
- Mitigations bubble: Green outline

Typography:

- Box titles: Bold, 11pt
- Sub-labels: Regular, 9pt
- Arrow labels: Italic, 9pt
- Boundary labels: Italic, 10pt, dashed underline

This diagram illustrates the verification pipeline from Bitcoin fact to network state transition. It highlights the separation of concerns: Bitcoin provides immutable timestamping, SPV provides inclusion proofs, Schnorr provides semantic attestation, and the network enforces state updates—all without executing Bitcoin Script or replaying global execution. The diagram is descriptive of the architecture; it does not imply performance superiority over other systems.

6. Schnorr as Semantic Compression

Bitcoin transactions encode raw data but lack high-level semantic context. Schnorr signatures provide a mechanism to compress semantic interpretation into a compact, verifiable attestation:

- Multisig attestations of meaning
- Threshold agreement on interpretation
- Compact proofs consumable by browsers and edge devices

Example:

"This Bitcoin transaction represents a finalized state commitment."

This statement is attested, not executed. The verification network accepts the attestation as evidence of semantic binding, conditional on threshold signature validity. This separates the concerns of *what happened* (Bitcoin SPV) from *what it means* (Schnorr attestation).

7. Security Analysis

7.1 Soundness

Forgery of a valid verification requires:

- Breaking Bitcoin PoW assumptions **or**
- Forging Schnorr signatures under discrete logarithm hardness **or**
- Producing an alternative anchored genesis contradicting Bitcoin consensus

All are assumed computationally infeasible under standard cryptographic assumptions. This does not address social or economic attacks (e.g., signer bribery), which fall outside the cryptographic threat model.

7.2 Liveness

Liveness depends on:

- Bitcoin header availability (assumed via peer-to-peer network)
- Attestation availability (signer liveness)
- Proof relay infrastructure (network-dependent)

Failures in any of these degrade service availability but do not compromise soundness. This is a standard tradeoff in light client architectures.

8. Related Work

This architecture composes established cryptographic primitives in a manner that is underexplored in existing blockchain systems. We briefly survey related approaches:

Bitcoin SPV (Nakamoto, 2008): The original Bitcoin whitepaper introduced SPV as a lightweight verification method for resource-bounded clients. Our work extends this by composing SPV with Schnorr attestations and genesis anchoring to enable semantic interpretation of Bitcoin events without full node validation.

FlyClient and NiPoPoWs: FlyClient (Bünz et al., 2019) and Non-Interactive Proofs of Proof-of-Work (Kiayias et al., 2016) provide succinct blockchain verification via probabilistic sampling or superblocks. These reduce header verification costs asymptotically. Our approach uses bounded verification windows and checkpoints rather than probabilistic sampling, accepting policy-bounded overhead in exchange for simpler verification logic.

Mina and Recursive SNARKs: Mina (formerly Coda) uses recursive zkSNARKs to compress blockchain state to a constant-size proof. This achieves $O(1)$ verification but requires heavy proving infrastructure. Our approach prioritizes prover simplicity and browser-native verification, accepting $O(k)$ header verification in exchange for eliminating SNARK proving overhead.

Ethereum Rollups: Optimistic and ZK rollups post state commitments to Ethereum L1 and inherit its security. They depend on L1 data availability and sequencer honesty (or fraud proofs). Our architecture does not depend on a base-layer execution environment, only on Bitcoin's timestamping, which provides a distinct security model.

Babylon: Babylon (David et al., 2023) enables chains to checkpoint to Bitcoin for finality. This is ex-post security via periodic anchoring. Genesis anchoring is architecturally distinct: it fixes network identity at inception, not just finality windows. Both approaches leverage Bitcoin's PoW, but with different trust assumptions.

Chainlink and Oracle Networks: Chainlink provides off-chain data aggregation via trusted node operators. Our oracle model (Section 5.1) reduces trust in data providers by verifying Bitcoin inclusion proofs, though it remains limited to Bitcoin-verifiable facts.

These comparisons are not exhaustive. The key distinction is the composition of genesis anchoring, SPV, and Schnorr for lightweight semantic verification—an approach that is rarely implemented in this specific configuration.

8.1 Comparative Summary

System	Identity Root	Verification Cost	Browser-Native
Bitcoin SPV	Bitcoin	$O(k)$ headers	Yes
FlyClient	Ethereum	$O(\log^2 n)$	Partial
Mina	Recursive SNARK	$O(1)$ (heavy prover)	No
Rollups	Ethereum L1	$O(n)$ or $O(1)$ w/ SNARK	No
This work	Bitcoin Genesis	$O(k)$ headers	Yes

This table summarizes verification costs and design choices. Each system optimizes for different constraints; no single approach dominates across all metrics.

9. Limitations and Assumptions

This architecture makes explicit tradeoffs and assumptions that limit its applicability:

- 1. Data Availability:** Light clients depend on proof availability from the peer-to-peer network. Adversaries controlling network peers may withhold proofs, degrading liveness without compromising soundness. This is inherent to SPV and cannot be eliminated without full node validation.
- 2. Prover Availability:** Schnorr attestations require signers to be online and responsive. Threshold schemes mitigate single points of failure, but t -of- n liveness assumes at least t honest signers remain available. Signer set management is an orthogonal concern not addressed in this paper.
- 3. Frontier Window Tradeoffs:** Policy-bounded verification windows (confirmation depth N , header chain length) trade off security against resource costs. Deeper confirmation thresholds increase reorg resistance but delay finality. This is a configurable policy parameter, not a protocol constant.
- 4. Not a Full Execution Environment:** This is a verification primitive, not a general-purpose smart contract platform. It does not execute arbitrary logic, does not provide Turing-complete computation, and does not replicate Bitcoin Script semantics. Applications requiring complex on-chain execution must layer additional primitives on top of this foundation.
- 5. Trust in Signers (Where Used):** When semantic attestations are required, the system trusts the threshold signer set not to collude or provide false attestations. This is not eliminated by cryptography—it is a conscious design tradeoff. Applications with higher assurance requirements may need additional mechanisms (e.g., bonding, slashing, or reputation systems).

These limitations are not deficiencies but explicit boundaries of the threat model. The architecture is designed for contexts where these assumptions are acceptable.

10. Conclusion

Genesis anchoring establishes *who* the network is, fixing identity at inception via Bitcoin consensus. Schnorr signatures and SPV together establish *what* external facts can be verified and *how* their semantic meaning is attested. The composition of these primitives—while individually well-understood—forms a minimal verification layer that is underexplored in existing blockchain architectures.

This approach is suitable for edge devices and offline-resilient clients where bounded resource requirements are critical. It does not replace execution layers, data availability systems, or consensus mechanisms—it provides a trust root upon which they can safely compose. The architecture accepts explicit limitations (data availability, prover liveness, bounded verification windows) in exchange for verification efficiency and light-client accessibility.

Future work may explore header verification optimizations (e.g., NiPoPoWs, FlyClient), signer set rotation mechanisms, and integration with zkSNARK-based execution proofs. The core contribution is demonstrating that genesis-anchored identity, when composed with SPV and Schnorr, enables a verification-first architecture suitable for resource-bounded environments.

Appendix A: Formal Verification Predicate

The verification predicate is defined as:

$$V(E) := \begin{cases} 1 & \text{if } G \in \text{Bitcoin} \wedge \text{SPV}(E) \wedge \text{SchnorrAgg}(E) \\ 0 & \text{otherwise} \end{cases}$$

Where G is the genesis commitment anchored to Bitcoin, $\text{SPV}(E)$ proves event E inclusion under PoW consensus, and $\text{SchnorrAgg}(E)$ is an optional threshold attestation binding semantic meaning to E . The predicate evaluates to 1 (accept) only if all conditions hold.