LayerEdge: Bitcoin's Verification Layer (Draft)

LayerEdge

1st Ayush Gupta
LayerEdge
ayush@layeredge.io

***Abstract*—Trust has become one of the most valuable commodities in our digital age. The evolving landscape of the internet presents challenges where trust in information and transactions has become increasingly fragile. Technological advances such as artificial intelligence (AI) and decentralized financial technologies have redefined the possibilities for verifiable computation and decentralized coordination. However, issues like fragmented liquidity, rising verification costs, and scalability bottlenecks limit the widespread adoption of decentralized networks as a universal means to enable trustless transactions and computations.**

LayerEdge is proposed as a novel solution that addresses these challenges by unlocking the Proof-of-Work (PoW) security of Bitcoin to create a universal verification layer. This approach secures the Internet with verifiable computation that is efficient, scalable, and cost-effective. By leveraging a modular architecture that allows for verifiable, off-chain execution secured by Bitcoin, LayerEdge extends the utility of Bitcoin's trusted network to modern decentralized applications, allowing robust verification without the inefficiencies typically seen in traditional blockchain networks.

LayerEdge transforms Bitcoin into a decentralized computer capable of verifying complex computations, not by executing them directly, but by verifying zero-knowledge proofs—much like checking the result of a complex equation without redoing all the calculations. The proposed infrastructure reduces the dependency on fluctuating gas prices, lowers verification costs, and fosters an ecosystem that supports new trustless applications across finance, artificial intelligence, the Internet of Things (IoT), and more. By aligning Bitcoin's security with modern zk-proof innovations, LayerEdge will drive the next evolution in decentralized trust frameworks, making it possible to bridge the gap between traditional financial systems and a verifiable internet.

I. Introduction

In today's rapidly evolving digital world, trust has become the most valuable commodity. We find ourselves in an environment where the integrity of the information we consume is constantly challenged by misinformation, deep fakes, and centralized data breaches. The internet, which was initially envisioned as a decentralized network of knowledge sharing, has instead become dominated by a few powerful intermediaries. These entities control not only information flow but also the verification of transactions, whether financial, social, or informational. Trust, in this scenario, relies heavily on centralized gatekeepers—an arrangement that has proven to be both fragile and vulnerable.

Blockchain technology emerged as a promising solution to address these challenges. Bitcoin, introduced in 2008 by Satoshi Nakamoto, was the first successful implementation of a decentralized peer-to-peer electronic cash system, built on the foundation of transparency and trustlessness. Using Proof-of-Work (PoW) as a consensus mechanism, Bitcoin demonstrated how a decentralized network could secure digital transactions without any intermediaries. It provided an unbreakable foundation of trust for financial transactions, resisting censorship and manipulation even by state-level actors.

Ethereum expanded upon Bitcoin's foundation by introducing smart contracts, which allow developers to build decentralized applications (dApps) that execute code without the need for intermediaries. This innovation unlocked a vast range of possibilities beyond financial transactions, enabling decentralized finance (DeFi), gaming, identity management, and much more. However, the increasing demand for decentralized applications brought with it a range of limitations related to blockchain scalability, efficiency, and cost. The core issue lies in blockchain architecture itself: unlike traditional centralized systems, where adding hardware leads to increased speed and performance, the blockchain model, particularly in Bitcoin and Ethereum, depends on every node in the network re-executing every transaction for consensus. While this design significantly strengthens reliability and security, it severely restricts throughput and scalability.

This redundant re-execution by all nodes can be thought of as a global assembly line in which every worker is tasked with repeating the same job already done by others. Such a model inevitably leads to network congestion, increased latency, and high transaction costs. To address these challenges, several blockchain projects have introduced alternative Layer 1 solutions, such as Solana, Avalanche, and Binance Smart Chain, which modify the rules to increase throughput. However, these solutions have come at the cost of reduced decentralization and compromised security, which undermines the fundamental promise of blockchain technology as a trustless, secure, and decentralized system.

The advent of zero-knowledge (ZK) proofs and validity proofs presented a novel approach to scalability without

compromising on security. Zero-knowledge proofs allow a verifier to confirm the validity of a computation without needing to re-execute the entire process. This approach can be likened to verifying the correctness of a complex mathematical equation without redoing all the calculations. By enabling efficient proof verification, ZK proofs drastically reduce the on-chain computational burden, thereby improving blockchain scalability and reducing costs. This innovation laid the groundwork for a modular blockchain architecture, where execution is separated from the base layer, thereby facilitating scalability.

Ethereum embraced this modular model through Layer 2 (L2) solutions like zk-rollups and optimistic rollups, which bundle multiple transactions off-chain before settling them back on-chain. These rollups extend Ethereum's capabilities by making the blockchain faster and more cost-effective, while still inheriting Ethereum's security properties. However, these modular solutions also present their own set of challenges, such as fragmented liquidity and a complex user experience resulting from the need to bridge assets between the main chain and multiple Layer 2 networks.

Despite Ethereum's innovations, Bitcoin—the world's most secure blockchain—has not yet fully leveraged these advancements. Although Bitcoin's Layer 2 solutions, such as the Lightning Network, have increased throughput for financial transactions by executing them off-chain and settling them back on the main chain, their high verification costs during times of network congestion have hindered broader adoption. Verifying cryptographic proofs directly on Bitcoin can be extremely resource-intensive due to its limited block space, resulting in high fees that make such operations cost-prohibitive. This significantly limits Bitcoin's potential as a universal verification network.

LayerEdge aims to address these challenges by transforming Bitcoin from merely a store of value to a powerful, decentralized computer capable of verifying complex computations without executing them directly. LayerEdge proposes a universal verification layer that enables verifiable off-chain execution, secured by Bitcoin's Proof-of-Work (PoW) mechanism. Instead of requiring every transaction to be processed on-chain, LayerEdge aggregates cryptographic proofs off-chain and verifies them on-chain using efficient zk-proofs, significantly reducing costs while retaining security. This approach is akin to leveraging Bitcoin's immense computational power and security without burdening its base layer with high transaction volumes or computational overhead.

The potential impact of LayerEdge extends far beyond financial transactions. The applications of zero-knowledge proofs are broad, with the ability to revolutionize industries such as artificial intelligence (AI), the Internet of Things (IoT), and Decentralized Physical Infrastructure Networks (DePIN). In these contexts, verifiable computation is crucial for ensuring trust in decentralized systems, where the authenticity of data, transactions, and commands must be verified independently of centralized authorities. With LayerEdge, Bitcoin's computational security can now support these emerging technologies, transforming Bitcoin from a passive financial asset into an active enabler of verifiable trust across the decentralized internet.

In addition to enhancing scalability and reducing verification costs, LayerEdge also addresses one of the most critical issues facing the blockchain ecosystem today—fragmented liquidity and user experience. The modular architecture of LayerEdge simplifies bridging between different blockchain networks, providing a unified verification layer that enhances both security and usability. By providing soft finality through quick, off-chain verification and hard finality through on-chain proof aggregation, LayerEdge ensures that decentralized applications can benefit from the best of both worlds—speed and security.

Bitcoin's security, anchored by its PoW consensus mechanism, represents one of the strongest foundations available for blockchain networks. However, its use has been limited primarily to financial transactions. LayerEdge seeks to unlock this security for a broader range of decentralized applications, enabling Bitcoin to secure not just value but also data, digital identities, supply chains, and more. By making Bitcoin's 21 million coins work beyond the Proof-of-Stake (PoS) economy, LayerEdge will enable it to become the backbone for zk protocols and beyond, establishing Bitcoin as the cornerstone of the verifiable internet.

The aim of this paper is to provide a detailed exploration of LayerEdge, its architecture, and its integration into the broader blockchain ecosystem. We will delve into the design principles that enable LayerEdge to transform Bitcoin's PoW security into a universal verification layer capable of supporting a wide range of decentralized applications. We will discuss how LayerEdge reduces the cost and complexity of verification while maintaining the high-security guarantees of Bitcoin. Ultimately, LayerEdge aims to extend Bitcoin's utility, making it an indispensable part of the decentralized internet, enabling verifiable computation across a variety of use cases, and thereby building a more transparent, trustless, and secure digital future.

A. Possible Use Cases

LayerEdge can serve as the foundation for a variety of decentralized applications beyond financial transactions. Some of the key potential use cases include:

- **Soft Finality for Rollups and Appchains:** LayerEdge can provide soft finality to rollups and appchains, allowing them to achieve faster confirmation times with

high-security guarantees.

- **New Settlement Layers:** With integration into existing Data Availability (DA) layers and Bitcoin's PoW, LayerEdge can serve as a new settlement layer for rollups, appchains, and intent-based systems that require robust and secure validation.
- **Verifiable Machine Learning:** AI models can benefit from verifiable training and inference results. LayerEdge can provide a mechanism to verify computations in machine learning, ensuring the authenticity of outcomes.
- **Decentralized Identity and Credential Verification:** Verification of digital identities and credentials could be streamlined with zk-proofs, allowing individuals to prove credentials without revealing sensitive information, with LayerEdge as the verifier.
- **Verifiable IoT and DePIN (Decentralized Physical Infrastructure Networks):** Internet of Things (IoT) networks can benefit from LayerEdge by verifying the authenticity of device data, ensuring the reliability of decentralized physical infrastructure networks without the need for centralized controllers.
- **On-Chain Gaming:** Blockchain-based gaming requires the verification of game logic and results. LayerEdge can be used to verify these computations off-chain and provide secure, tamper-proof proof verification on-chain.

These use cases exemplify the versatility and transformative potential of LayerEdge in various domains of the decentralized internet. By leveraging Bitcoin's PoW security to verify complex computations in a cost-effective manner, LayerEdge will not only enhance Bitcoin's utility but will also create new avenues for building and scaling trustless applications.

II. Architecture of LayerEdge

The architecture of LayerEdge is engineered to capitalize on Bitcoin's Proof-of-Work (PoW) security to build a scalable, cost-effective, and decentralized universal verification layer. This architecture leverages state-of-the-art zero-knowledge (zk) technologies and advanced modular concepts to enable secure off-chain computation, verifiable on-chain proofs, and decentralized infrastructure.

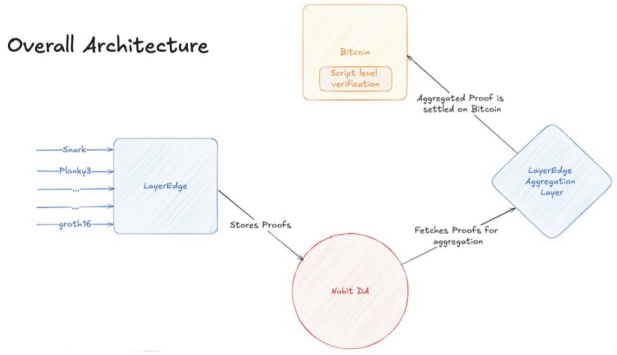
The following sections describe the major components, their functionality, the interactions between them, and how these interactions achieve efficient verification and universal trust.

A. Architectural Overview

LayerEdge's architecture is comprised of four primary components:

- **LayerEdge Verification Layer:** A decentralized off-chain layer that aggregates and verifies zk-proofs.
- **Data Availability Layer (DA Layer):** Ensures that all proof data is accessible and tamper-proof, integrating with existing DA layers for enhanced compatibility and scalability.
- **General Prover/Verifier System:** Handles recursive aggregation of proofs, enabling efficient scaling of zk-proof verification.
- **Bitcoin Settlement Layer:** Acts as the ultimate source of truth for proof verification and provides hard finality through its PoW security.

These components interact in a modular, layered fashion to create a highly resilient and secure verification ecosystem capable of supporting numerous decentralized applications.



B. LayerEdge Verification Layer

The LayerEdge Verification Layer is responsible for handling the ingestion, normalization, verification, and aggregation of proofs before posting the results on the Bitcoin blockchain.

1) Proof Reception and Normalization:

- **Multi-Proof Compatibility:** LayerEdge supports multiple zk-proof systems, including zk-SNARKs (such as Groth16, PLONK), zk-STARKs, and newer zk-proofs under development. Each zk-proof format requires different configurations in terms of fields, elliptic curves, and computational models.

- **Proof Normalization Process:** Upon receipt, the proofs undergo normalization. Proof normalization involves:
  - **Field Conversion:** Transforming field elements into a standard elliptic curve-compatible format for zk-SNARK-based aggregation.
  - **Constraint System Encoding:** Converting the underlying constraints (such as R1CS, AIR) into a unified format, allowing compatibility with recursive proof techniques.
- **Compatibility Layers:** The system employs compatibility layers that can abstract different zk-proof technologies into a unified framework that facilitates aggregation. For instance, zk-STARKs, which are optimized for scalability but have larger proof sizes, are transformed to compressed zk-SNARK-compatible formats through intermediate zk-commitment schemes.

These components interact in a modular, layered fashion to create a highly resilient and secure verification ecosystem capable of supporting numerous decentralized applications.

2) Proof Verification and Aggregation:

- Batch Verification Using Recursive zk-SNARKs:
  - **Recursive zk-SNARK Circuits:** Recursive circuits are implemented to verify individual proofs. These circuits take multiple zk-proofs as input and generate a single succinct proof that confirms the correctness of all underlying computations.
  - **zk-Merkle Tree Aggregation:** Proofs are batched and organized into a Merkle tree structure for efficient verification. Each leaf node represents an individual zk-proof, while intermediate nodes represent recursive verifications, culminating in a root that attests to the validity of all computations.

3) Finality Mechanisms:

- Soft Finality (Off-Chain Verification):
  - **LayerEdge Verification Layer Finality:** Soft finality is achieved when zk-proofs are verified by the LayerEdge Verification Layer. These proofs are aggregated and verified off-chain, allowing for immediate confirmation of computational correctness.
  - **Fast Validation for Low Latency Applications:** Soft finality is used for applications where immediate feedback is crucial, such as decentralized exchanges, payments, and P2P networks.
- Hard Finality (On-Chain Verification with Bitcoin):
  - **Aggregated Proof Commitment:** After aggregating proofs, the final recursive proof is submitted to Bitcoin for on-chain settlement. Posting these proofs on Bitcoin establishes hard finality, leveraging the unparalleled security of Bitcoin's PoW.
  - **Data Composition:** During on-chain posting, OP_CAT is employed to concatenate multiple components of an aggregated zk-proof (e.g., proof signatures, metadata, commitments) into a single transaction output. This significantly reduces the verification load by making on-chain data handling simpler.

C. General Prover/Verifier System

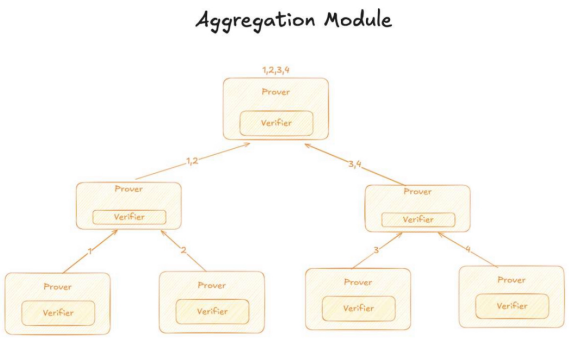
The General Prover/Verifier System is a critical part of LayerEdge, responsible for proof aggregation through recursive zk-techniques. It ensures that multiple zk-proofs can be efficiently compressed into a single proof.

1) General zk Virtual Machines:

- General-Purpose zk-Prover VMs:
  - **zk-VM Compatibility:** The General Prover/Verifier system leverages zk-VMs (Virtual Machines) such as SP1, Risc0, and Nexus. These zk-VMs are capable of executing general-purpose computations written in Rust, WASM, or LLVM IR, allowing compatibility with a wide range of zk-proof systems.
  - **Constraint System Execution:** zk-VMs execute computations within a zk-arithmetic circuit that encodes constraints. This allows for proving arbitrary programs, which is essential for supporting complex decentralized applications.
- Optimization Techniques:
  - **Constraint Folding and Optimization:** To minimize computational overhead, the General Prover/Verifier applies advanced optimization techniques such as constraint folding. This reduces the number of linear and quadratic constraints required for zk-proof generation.
  - **Lookup Tables and Batch Execution:** Techniques like plookup tables are used to expedite specific computations by replacing general constraints with precomputed lookup operations. This is particularly useful in scenarios that require repetitive tasks.

2) Recursive Proof Aggregation:

- Recursive zk-SNARK Aggregation:
  - **Recursive Verification Strategy:** zk-SNARKs (e.g., Halo2) are employed to aggregate multiple individual proofs recursively. This recursive strategy ensures that the resulting proof is both succinct and verifiable.
  - **Layered Recursive Aggregation Trees:** The aggregation process is represented as a recursive Merkle tree structure, where internal nodes recursively verify and aggregate the child zk-proofs. The final output is a single zk-proof representing all underlying computations.



D. Bitcoin as the Base Settlement Layer

Bitcoin is the base settlement layer of LayerEdge, providing ultimate security through its PoW mechanism. This integration ensures that verification results are secured with the highest available level of decentralized trust.

1) Leveraging Bitcoin's Proof-of-Work Security:

- Restaking of Bitcoin Security:
  - **Bitcoin PoW Integration:** LayerEdge's security relies on Bitcoin's existing PoW infrastructure, effectively restaking Bitcoin's hash power to provide security for proof verification. This eliminates the need for a separate staking mechanism, thus inheriting Bitcoin's resilience against attacks.
  - **On-Chain Validation with Smart Contracts:** Aggregated zk-proofs are submitted to Bitcoin, where specialized smart contracts (Taproot-enabled scripts) handle the receipt, verification, and recording of these proofs. These smart contracts minimize the need for full on-chain execution by using succinct zk-proofs.
- On-Chain Compression with OP_CAT
  - **Signature and Proof Concatenation:** The use of OP_CAT allows concatenation of multiple zk-proof elements and signatures into a single transaction, reducing the overall on-chain footprint and minimizing complexity during on-chain verification.

2) Aggregated Proof Submission:

- Single Aggregated zk-Proof Posting:
  - **Cost Minimization via Aggregation:** Posting a single zk-proof that attests to the correctness of hundreds or thousands of computations minimizes on-chain interactions and thereby reduces the associated costs.
  - **Immutable Settlement with Bitcoin:** Once a proof is posted on Bitcoin, it becomes part of an immutable blockchain, leveraging Bitcoin's security guarantees and making the verification irreversible and tamper-proof.

E. Communication and Interaction

Efficient communication between components is fundamental to the performance of LayerEdge. The system's interactions are characterized by secure protocols that facilitate proof generation, verification, and on-chain settlement.

1) Proof Lifecycle:

- Proof Generation:
  - Computations are executed off-chain, generating zk-proofs that attest to their correctness.
- Storage in the DA Layer:
  - Proof data and metadata are committed to the DA Layer, ensuring data is readily accessible for verification and re-verification.
- Operator Verification:
  - Operators retrieve proofs, verify them, and submit the results to the General Prover System for aggregation.
- Recursive Aggregation and zk-SNARK Compression:
  - Recursive aggregation takes place, with the General Prover System generating a single aggregated

zk-proof.

- Final Posting to Bitcoin:
  - The final zk-proof is posted to the Bitcoin blockchain for immutable settlement

2) Secure Communication Protocols:

- Layered Data Exchange:
  - **P2P Data Transmission:** Off-chain proof data exchange is handled using secure peer-to-peer (P2P) channels with end-to-end encryption. These channels facilitate the communication between operators, verifiers, and provers.
  - **Merkle Proof Validation:** The interaction between the DA Layer and operators involves Merkle inclusion proofs to verify data authenticity without the need for full data retrieval.

F. Security, Scalability, and Robustness

LayerEdge is designed to achieve high levels of security and scalability while maintaining robustness against adversarial conditions.

1) Robust Security through Bitcoin PoW:

- **PoW Trust Extension:** By leveraging Bitcoin's PoW as the ultimate verification layer, LayerEdge ensures that all proofs are as secure as Bitcoin's decentralized network. This PoW trust extension is a cornerstone of LayerEdge's architecture, preventing any single point of failure.
- **zk-Proofs for Trustless Verification:** zk-SNARKs are used for succinct proof generation, ensuring that verification can take place without exposing underlying data, thus preserving privacy and enhancing trustlessness.

2) Modular Scalability through Off-Chain Computation:

- **Separation of Execution and Verification:** Off-chain computation and zk-proof verification minimize the load on Bitcoin's blockchain, which in turn provides significant scalability improvements compared to traditional blockchains that rely solely on on-chain execution.
- **High-Throughput Proof Aggregation:** Recursive proof aggregation drastically reduces the size of proofs that need to be submitted on-chain, enabling LayerEdge to handle thousands of computations without overwhelming the Bitcoin blockchain.

3) Robustness Against Congestion:

- **Minimized On-Chain Interaction:** Only aggregated proofs are posted on-chain, reducing Bitcoin blockchain congestion and associated gas fees during periods of high network activity.
- **Optimized zk-Proof Strategies**: zk-STARKs, which do not require a trusted setup, are used for off-chain scalability and are subsequently aggregated via zk-SNARKs for on-chain settlement. This dual approach allows LayerEdge to adapt to network conditions and minimize congestion risks.

III. LayerEdge STV (State Transition Verification)

LayerEdge STV is an innovative system designed to verify state transitions directly on the Bitcoin blockchain. It allows for arbitrary program execution with Turing-complete capabilities while ensuring scalability and security through an optimistic computation model, SNARK-based verification, and a state-centric architecture. Below, the core components of the LayerEdge STV system are discussed in detail, covering their roles, technical implementation, and cryptographic interactions.

A. Optimistic Computation for State Transition

LayerEdge STV leverages an optimistic computation model for verifying state transitions on-chain, building on a modular approach to handle the computational and storage constraints of Bitcoin's scripting language.

1) Off-chain State Transition Execution:

- Operators perform the execution of a state transition off-chain for a given program f and generate an updated state. The transition is assumed correct until proven otherwise.
- State transitions involve changing an input state S_i to an output state S_o, which represents updates in a distributed system's state machine.
- The computation is split into smaller, sequential steps such that each transition can be verified independently. These steps yield intermediate states $z0, z1, ..., zk$, which help in modularizing verification.

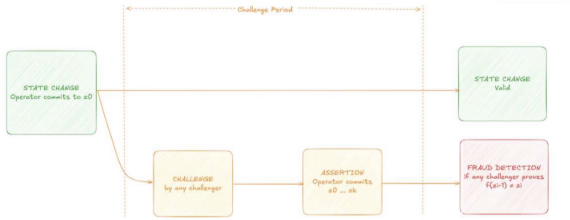
2) State Transition Proof Generation:

- **SNARK Proof Generation:** The operator generates a SNARK proof for the entire state transition from S_i to S_o, representing the correctness of the transition without revealing the full computational details.
- The generated proof is compact and easily verifiable on-chain. The SNARK proof π validates that the

transition, including all intermediate states, was computed accurately off-chain.

3) Commitment to State Changes:

- Operators commit to their computed state transition by publishing an "Assert" transaction on the Bitcoin blockchain. This transaction acts as a verifiable commitment to both the SNARK proof and the intermediate states.
- The Assert Transaction includes:
  - SNARK Proof π that validates the overall transition from S_i to S_o.
  - **Lamport Signatures for State Commitments:** Each intermediate state $(z1, z2, ..., zk)$ is signed using Lamport signatures, providing an integrity guarantee for each step of the computation.
  - **State Commitment:** A commitment to the final state S_o is also provided. This commitment is essential for ensuring that subsequent state transitions in the distributed system can be chained correctly.



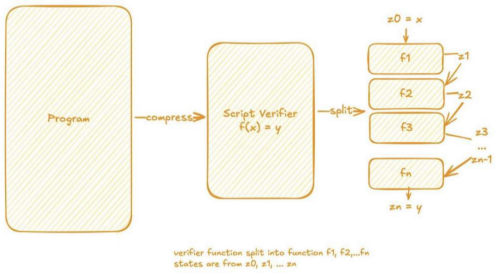
Verification Module

B. SNARK Verifier Integration for State Verification

LayerEdge STV integrates a SNARK verifier within Bitcoin's UTXO model to ensure that state transitions are verified with minimal on-chain computation and storage.

1) SNARK Setup Phase: LayerEdge STV integrates a SNARK verifier within Bitcoin's UTXO model to ensure that state transitions are verified with minimal on-chain computation and storage.

- SNARK Setup Phase:
  - The setup phase involves generating a common reference string (CRS), which serves as a trusted cryptographic setup. The CRS allows both proof creation and succinct verification using zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs).
  - The Groth16 SNARK protocol is selected due to its efficient proof generation and minimal verification cost. This SNARK verifier is implemented in a form that can be modularized to fit within Bitcoin's scripting constraints.



Bitcoin Script Verification

- Verifier Program Implementation:
  - Given Bitcoin's block size and scripting language limitations, the SNARK verifier cannot be implemented as a monolithic program. Instead, the verifier is divided into multiple sub-verifiers $v1, v2, ..., vk$.
  - Each sub-verifier verifies a segment of the proof, ensuring that the state transition is computed accurately through each phase. This modular verification approach enables the system to handle large state transitions incrementally.
  - **Intermediate State Handling:** Each sub-verifier operates on an intermediate state (zi) and outputs the next state $(zi + 1)$. The operator commits to the correctness of these states by using OP_CAT to concatenate state fragments, ensuring efficient on-chain representation.
  - Using OP_CAT, the sub-verifier scripts can assemble different parts of the input and output data required for each state transition, ensuring the transitions are correctly linked and verified sequentially.
- On-chain State Transition Verification:
  - Each **sub-verifier script** is implemented in Bitcoin Script, using OP_CAT to concatenate and

manipulate state data, enabling on-chain verification for each segment of the state transition. This approach provides flexibility in verifying parts of the computation as needed, rather than performing the entire verification at once.

- **Taproot Integration:** Taproot is leveraged to encode multiple possible verification paths into a single output, minimizing on-chain overhead. Each potential state challenge or successful verification outcome is represented by a separate Tapleaf, allowing compact storage and efficient verification.

C. Commitment and Challenge Mechanism

The commitment and challenge mechanism forms the backbone of the security model for LayerEdge STV, allowing challengers to verify state transitions and detect errors in state computation.

1) Assert Transaction:

- The Assert Transaction is used by an operator to publish the committed states $(z0, z1, ..., zk)$, including the final state S_o. This transaction serves as a binding commitment to the computed state transition.
- The transaction output contains:
  - **State Commitments and Proof:** It includes commitments for each intermediate state as well as the SNARK proof (π) verifying the correctness of the transition.
  - **Taproot Conditions:** The output is a Taproot tree, encoding multiple conditions for spending, such as payout to the operator if no dispute occurs or initiating a challenge in case of detected inconsistencies.
  - **Fraud Proof Window:** A time-lock using $CHECKLOCKTIMEVERIFY(CLTV)$ or $CHECKSEQUENCEVERIFY(CSV)$ ensures that a window is provided during which anyone can challenge the asserted state transition.

2) Challenge Mechanism (Disprove Transaction):

- Any party can verify the committed states off-chain and initiate a challenge if they detect a discrepancy. The challenger issues a Disprove Transaction to dispute the correctness of the state transition.
- The challenger must:
  - Provide the specific incorrect intermediate state and the associated computation step that deviates from the committed value.
  - Execute the corresponding sub-verifier script on-chain to demonstrate that the operator's state transition was incorrect.
- If the challenge is successful, the operator's collateral is slashed as a penalty, with a portion rewarded to the challenger for ensuring the system's integrity. The remaining collateral may be burned to maintain economic balance.

3) Collateral Mechanism:

- Operators are required to stake collateral (dB) when submitting an Assert transaction. This collateral serves as a deterrent to submitting incorrect state transitions.
- Challengers must also post collateral when issuing a Disprove transaction, preventing malicious actors from issuing frivolous challenges. If the challenge is deemed fraudulent or incorrect, the challenger loses their collateral.
- The collateral system is designed to align economic incentives, ensuring that both operators and challengers are motivated to act honestly and verify state transitions accurately.

D. Security Mechanisms

The security mechanisms in LayerEdge STV are critical for ensuring that state transitions are correctly computed, publicly verifiable, and that dishonest participants are economically penalized.

1) Lamport Signatures for State Integrity:

- Lamport One-Time Signatures are used for signing each intermediate state in the computation. These signatures provide a high level of security due to their resistance to quantum attacks and the fact that they are computationally efficient for one-time use.
- Operators generate a Lamport key pair (pkz, skz) for each intermediate state. The intermediate states (zi) are signed using the corresponding private key (skz). These signatures are then verified during the challenge process to ensure state integrity.

2) Taproot and Connector Outputs:

- Taproot Trees are used to efficiently manage the multiple spending conditions that arise from possible challenges. Each condition—such as payout to the operator, challenge by a third party, or time-out spend—is represented by a different Tapleaf in the Taptree structure.

- Connector Outputs are utilized to enforce atomicity between the commitment and challenge phases. A connector output is used in the Assert transaction to prevent the operator from finalizing their state transition until the challenge window expires.

3) Optimistic State Transition Verification:

- The LayerEdge STV model is built on an optimistic verification approach:
  - In the first stage, operators commit to a state transition and post the Assert transaction with the SNARK proof and state commitments.
  - In the second stage, challengers can verify these commitments off-chain and issue a fraud proof on-chain if they identify a discrepancy.
- This model combines game-theoretic incentives with cryptographic proofs to ensure that rational participants—motivated by economic incentives—act in a way that upholds the correctness of state transitions.

E. Pre and Post OP_CAT in State Transition Verification

One of the significant innovations that LayerEdge STV introduces is how it optimizes state transitions through the introduction of OP_CAT, which simplifies state concatenation and verification.

1) Pre OP_CAT: Pointer-based State Transition: Before the introduction of OP_CAT, handling state transitions required storing pointers to subsequent programs and intermediate states in each sub-program. This approach, while functional, was computationally expensive and introduced storage inefficiencies due to Bitcoin's script size limitations.

- **Program Split into Sub-programs:** The verification program f was divided into multiple smaller sub-programs $f1, f2, ..., fk$ due to Bitcoin's block size and script limitations.
  - For each intermediate state transition, a sub-program computes: $f_1(z_0) = z_1$, $f_2(z_1) = z_2$ ... $f_k(z_{k-1}) = z_k$
  - **Pointer Storage:** Each sub-program $f1, f2, ...fk$ needed to store a pointer to the next sub-program. For example, after computing z_1 in $f1$, a pointer to $f2$ was stored in $f1$ for the next execution step.
  - **Challenges:** Storing pointers required significant additional data, leading to increased on-chain storage requirements and a higher computational burden for tracking the program flow across state transitions.

2) Post OP_CAT: Optimized State Transition through Concatenation: With the introduction of OP_CAT, LayerEdge STV can concatenate the outputs of the sub-programs, which significantly reduces the complexity of state transition handling.

- Concatenation of Sub-programs: The output of each sub-program $f1, f2, ..., fk$ is concatenated using OP_CAT, allowing the entire state transition sequence to be represented as a single chain:
  - Instead of storing a pointer, the output of each sub-program is concatenated with the next state, such that: $f_1(z_0)||f_2(z_1)||...||f_k(z_{k-1}) = z_k$
  - **Advantages:**
    - **Reduced Storage:** By concatenating the outputs directly, there is no need to store pointers or track the location of subsequent programs.
    - **Improved Computation Efficiency:** The verifier can execute the entire state transition sequence in a streamlined fashion, using the concatenated output to move from one state to the next without additional computational steps for pointer handling.
    - **Compact Representation:** This allows the entire state transition to be represented more compactly on-chain, fitting within Bitcoin's scripting constraints while maintaining efficient and secure verification.

3) Benefits to LayerEdge STV's Architecture: The transition from pointer-based state storage to concatenated state transitions via OP_CAT has a profound impact on LayerEdge STV's architecture:

- **Scalability:** By reducing the overhead associated with pointer storage, OP_CAT enables LayerEdge STV to handle larger and more complex state transitions with minimal on-chain overhead.
- **Security:** The reduction in computational steps and storage requirements lowers the attack surface for state transition challenges, further enhancing security.
- **Modularity:** With state transitions concatenated into a single continuous process, the modular verification system of LayerEdge STV can handle sub-verifiers more efficiently, improving the overall performance of state verification on the Bitcoin blockchain.

References

[1] Aligned Layer: Universal Verification Layer (DRAFT). (2024). https://whitepaper.alignedlayer.com

[2] Bitcoin Forum. BitVM Bridges Considered Unsafe. (2013). https://bitcointalk.org/index.php?topic=277389.0

[3] Ethereum Research. Optimistic rollups. (2022). https://ethereum.org/en/developers/docs/scaling/optimist

[4] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. (2008). https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging

[5] Robin Linus. BitVM: Compute Anything on Bitcoin. (2023). https://bitvm.org/bitvm.pdf

[6] Salvatore Ingala. Merkleize all the things. (2022). https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-November/021182.html

[7] Tyler Whittle, Rijndael. CoinWitness: Really ultimate blockchain compression. (2024). https://medium.com/@twhittle/bitvm-bridges-considered-unsafe-9e1ce75c8176

[8] Nexus 1.0: Enabling verifiable computation. (2024). https://www.nexus.xyz/whitepaper.pdf

[9] Babai, L., Fortnow, L., Levin, L. A., Szegedy, M. Checking computations in polylogarithmic time. (1991).

[10] EigenLayer Team. Eigenlayer: The restaking collective.

[11] Goldwasser, S., Micali, S., Rackoff, C. The knowledge complexity of interactive proof systems. (1989).

[12] Parno, B., Gentry, C., Howell, J., Raykova, M. Pinocchio: Nearly practical verifiable computation. (2013). https://eprint.iacr.org/2013/279

[13] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. (2018). https://eprint.iacr.org/2018/046

[14] Groth, J. On the size of pairing-based non-interactive arguments. (2016). https://eprint.iacr.org/2016/260

[15] Gabizon, A., Williamson, Z. J., Ciobotaru, O. PLONK: Permutations over Lagrange-bases. (2019). https://eprint.iacr.org/2019/953

[16] Chen, B., Bünz, B., Boneh, D., Zhang, Z. Hyperplonk: PLONK with linear-time prover and high-degree custom gates. (2022). https://eprint.iacr.org/2022/1355

[17] Diamond, B. E., Posen, J. Succinct arguments over towers of binary fields. (2023). https://eprint.iacr.org/2023/1784

[18] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N. Marlin: Preprocessing zkSNARKs. (2019). https://eprint.iacr.org/2019/1047

[19] Thaler, J. Proofs, Arguments and Zero-Knowledge. (2023).

[20] Setty, S., Thaler, J., Wahby, R. Customizable constraint systems for succinct arguments. (2023). https://eprint.iacr.org/2023/552

[21] Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. (2021). https://eprint.iacr.org/2021/1043

[22] Haböck, U., Levit, D., Papini, S. Circle STARKs. (2024). https://eprint.iacr.org/2024/278

[23] Gabizon, A., Williamson, Z. J. plookup: A simplified polynomial protocol for lookup tables. (2020). https://eprint.iacr.org/2020/315

[24] Papini, S., Haböck, U. Improving logarithmic derivative lookups using GKR. (2023). https://eprint.iacr.org/2023/1284

[25] Bünz, B., Chiesa, A., Mishra, P., Spooner, N. Proof-carrying data from accumulation schemes. (2020). https://eprint.iacr.org/2020/499

[26] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M. Scalable zero knowledge via cycles of elliptic curves. (2014). https://eprint.iacr.org/2014/595

[27] Boneh, D., Shoup, V. A graduate course in applied cryptography. (Draft 0.6, 2023).

[28] Dong, X., Thyfronitis Litos, O. S., Tas, E. N., Tse, D., Woll, R. L., Yang, L., Yu, M. Remote staking with economic safety. (2024).

[29] Garay, J., Kiayias, A., Leonardos, N. The Bitcoin backbone protocol: Analysis and applications. (2015).

[30] Gentry, C., Wichs, D. Separating succinct non-interactive arguments from all falsifiable assumptions. (2011).

[31] Goldwasser, S., Micali, S., Rivest, R. L. A digital signature scheme secure against adaptive chosen-message attacks. (1988).

[32] Groth, J. On the size of pairing-based non-interactive arguments. (2016).

[33] Harding, D., Schmidt, M. Bitcoin optech: Covenants. (2024).

[34] Herlihy, M. Atomic cross-chain swaps. (2018).

[35] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., Felten, E. W. Arbitrum: Scalable, private smart contracts. (2018).

[36] Kiayias, A., Miller, A., Zindros, D. Non-interactive proofs of proof-of-work. (2020).

[37] Lamport, L. Constructing digital signatures from a one-way function. (1979).

[38] Linus, R. Stakechain: A bitcoin-backed proof-of-stake. (2022).

[39] Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P. Simple Schnorr multi-signatures with applications to Bitcoin. (2019).

[40] Möser, M., Eyal, I., Sirer, E. G. Bitcoin covenants. (2016).

[41] Nick, J., Ruffing, T., Seurin, Y. Musig2: Simple two-round Schnorr multi-signatures. (2021).

[42] Nick, J., Ruffing, T., Seurin, Y., Wuille, P. Musig-DN: Schnorr multi-signatures with verifiably deterministic nonces. (2020).

[43] O'Connor, R., Piekarska, M. Enhancing Bitcoin transactions with covenants. (2017).

[44] Russell, R. The great script restoration. (2024).

[45] Teutsch, J., Reitwießner, C. A scalable verification solution for blockchains. (2024).

[46] Bitcoin Wiki. Contract: Sighash flags. (2023).

[47] Wuille, P., Nick, J., Towns, A. BIP 0341, Taproot: Segwit version 1 spending rules. (2020).

[48] Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W. J. SoK: Communication across distributed ledgers. (2019).

[49] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W. J. Xclaim: Trustless, interoperable cryptocurrency-backed assets. (2018). https://eprint.iacr.org/2018/643

[50] ZeroSync. BitVM Github repository. (2023). https://github.com/BitVM/BitVM