SSLE-DAG: A High-Throughput Proof-of-Stake Consensus Protocol Combining an Adaptive DAG with a Single Secret Leader Election

Tomas Hladky, Martin Perešíni, Juraj Mariani, Ivan Homoliak
Brno University of Technology
Brno, Czech Republic
Email: hladkytomas21@gmail.com, {iperesini, imariani, homoliak}@fit.vut.cz

***Abstract*—Proof-of-Stake (PoS) blockchains with publicly visible leader schedules expose future proposers to targeted Denial-of-Service (DoS) attacks. Single Secret Leader Election (SSLE) techniques address this problem by hiding the leader's identity until block publication. However, existing SSLE techniques are difficult to integrate with high-throughput (Directed Acyclic Graph) DAG-based Proof-of-Stake consensus protocols. We introduce SSLE-DAG, a PoS consensus protocol that combines a zk-SNARK-based SSLE commitment scheme with the adaptive DAG-based consensus protocol that splits or merges parallel chains (and thus regulates throughput) upon transaction demand. The commitment scheme uses EdDSA signatures, MiMC hashing, and Merkle proofs to guarantee uniqueness, fairness, and unpredictability while keeping leader identities private.**

We implement SSLE-DAG in Go (gnark) and evaluate it in a geo-distributed simulation using real-world latency traces. In a 60-node network, we achieve about 990 TPS, and in a 40-node network with shorter rounds, we reach about 1,600 TPS with low variance in block rewards.

I. Introduction

Proof-of-Stake (PoS) consensus has become the dominant approach for permissionless blockchains due to its energy efficiency and strong economic security guarantees [17], [22]. Nevertheless, pure PoS protocols remain fundamentally limited in scalability in terms of transactional throughput. Most PoS systems retain a linear block structure with sequential block production, which constrains throughput and increases confirmation latency under high transaction demand. In general, reducing block creation time can partially alleviate this bottleneck, but it amplifies coordination overhead, fork rates, and sensitivity to network delays, limiting practical scalability [10], [27].

Directed Acyclic Graph (DAG)-based ledger structures were introduced to overcome these limitations in PoW by enabling parallel block production. Early PoW-based DAG protocols, such as PHANTOM/GHOSTDAG [27] and SPECTRE [26] demonstrated that high throughput can be achieved without enforcing a strictly linear chain. However, these designs introduce complex ordering rules and incentive attacks [24] in terms of the voluntary following of transaction selection rules, and are moreover not directly compatible with stake-based leader election in open permissionless environments.

More recent systems combine DAG structures with Byzantine Fault Tolerant (BFT) consensus, leading to hybrid PoS DAG designs such as Narwhal/Tusk [11], Bullshark [28], DAG-Rider [21], and Aleph [16]. Although these protocols achieve high throughput and fast finality, they rely on explicit BFT coordination, typically assuming a fixed small set of validators (or committees). As a consequence, scalability is achieved at the expense of decentralization [10], [11], [28], since BFT communication overhead restricts the size of the validator set.

Moreover, there exists a problem that in PoS consensual protocols leaders are known beforehand, enabling adversaries to launch targeted DoS attacks [5], [9] against upcoming block producers, directly threatening liveness and fairness. Despite advances in Single Secret Leader Election (SSLE) [5], integrating secret leader election with high-throughput DAG PoS designs remains challenging because parallel block production increases the number of simultaneously elected leaders and thus expands the attack surface, while leader secrecy complicates accountability for missed slots. This gap motivates our work, SSLE-DAG addresses this by combining a zk-SNARK-based commitment scheme for secret slot eligibility with a structured adaptive DAG that scales the number of parallel branches with observed transaction demand.

Contributions.

1) We design SSLE-DAG, a structured DAG-based PoS protocol that adapts parallelism using Sycomore-style [2] splitting and merging rules.
2) We instantiate a commitment scheme based on zk-SNARK in SSLE-DAG, which is inspired by shuffle-based SSLE proposals, where a validator proves eligibility for the epoch and uniqueness without revealing identity prior to block publication (which protects from DoS attacks).
3) We build a simulator of our approach (implementation of SSLE-DAG), which reflects network aspects of geo-distributed consensus nodes by modeling realistic network peer-to-peer (p2p) conditions drawn from existing dataset of network traces.
4) We evaluate our implementation on 40 and 60 nodes in the network, achieving up to about 1,600 TPS and 990 TPS, respectively.

II. Related Work

In this section, we focus on a brief review of existing SSLE designs and DAG-based consensus protocols aimed at improving scalability.

A. Single Secret Leader Election (SSLE)

Boneh et al. [5] formalized SSLE to counter public leader election in PoS and defined the requirements of uniqueness, fairness, and unpredictability. SSLE differs from VRF-based leader election used in PoS protocols such as Algorand [17] or Ouroboros [22], where "zero" or "multiple" leaders can occur in a single slot. Ethereum's Whisk [20] proposal introduces a shuffle-based SSLE mechanism and has been standardized in EIP-7441 [14], however, its global deployment is still missing. Homomorphic Sortition [15] provides an asynchronous SSLE construction based on homomorphic encryption. Recent empirical analysis of Whisk and Homomorphic Sortition (HS) highlights performance and security trade-offs in realistic settings of multiple coordinated attackers [9], identifying HS as more resistant than Whisk but at the same time prohibitively expensive. In contrast, our work proposes a custom SSLE design (inspired by Whisk) focusing on a structured DAG-based PoS consensus protocol that we propose.

B. DAG-Based Consensus and Adaptivity

DAG-based protocols enable parallel block production, improving scalability. PoW DAG-based designs such as SPECTRE [26] and PHANTOM/GHOSTDAG [27] explore ledger ordering in a setting of unstructured chains of DAG. Conflux [23] introduces a Tree-Graph ledger to combine scalability with confirmation guarantees. More recent DAG-based BFT protocols include Aleph [16], DAG-Rider [21], Narwhal/Tusk [11], and Bullshark [28] – these protocols are BFT or hybrid combination with PoS, not pure PoS consensual protocols. Sycomore [2] proposed a structured DAG-based PoW consensus protocol that adapts its structure upon transaction load. In contrast, SSLE-DAG utilized the adaptive idea of Sycomore in the setting of PoS, and moreover adds resistance to DoS on the parallel leaders, which is a specific problem to PoS.

III. System Model and Preliminaries

In this section, we describe system model, our assumptions, and requirements on the protocol we indetend to design.

A. System Model

We consider a permissionless PoS network in which each validator registers a public key and locks a stake to become eligible for block production. The eligible validator set is fixed per epoch and represented by a Merkle root known to all nodes. Nodes communicate over authenticated channels in a partially synchronous network. The adversary can attempt targeted DoS attacks against leaders, but cannot break standard cryptographic assumptions (EdDSA unforgeability, hash preimage resistance, or zk-SNARK soundness). We assume an unbiased randomness beacon for shuffling commitments. In the proof-of-concept implementation we replace the beacon with fixed keys, Section V evaluates under this simplification.

B. zk-SNARKs

Zero-knowledge proofs [18] (ZKPs) are cryptographic protocols that enable one party, the prover, to convince another party, the verifier, of the correctness of a statement without revealing any information beyond the validity of the statement itself. In blockchain systems, ZKPs are increasingly used to balance privacy, verifiability, and scalability, e.g., in cross-chain interoperability [30], private transactions [29], decentralized identity [3], some SSLE schemes [5], [6] (including our work), etc. zk-SNARKs provide succinct non-interactive zero-knowledge arguments of knowledge, especially groth16 [19]. Succinct zk-SNARK systems and their circuit-based execution model were popularized by vnRAM construction [4] and have been widely adopted since then in many use-cases for blockchains. Since many practical zk-SNARKs (groth16) rely on a structured reference string (trusted setup [7]), multi-party ceremonies mitigate trust by ensuring security if at least one participant is honest. We use ZK-friendly primitives in the circuit, such as MiMC hashing [1], to reduce multiplicative complexity as optimization.

C. SSLE Requirements – Security Goals

Several SSLE schemes are reviewed in Section II-A, and we assume that SSLE schemes in general require publishing of the private commitments that must satisfy the following attributes [5]:

- **Uniqueness:** Exactly one chosen leader per slot per epoch.
- **Fairness:** the selection probability of each validator is proportional to her stake.
- **Unpredictability / Unlinkability:** SSLE commitments cannot be linked to authors before secrets are revealed in a block. As a result, participants can only know their own leader status for a round, and the leader's identity is revealed only when the block is published.

IV. Proposed Approach

Building on the system model in Section III, we now describe SSLE-DAG, an adaptive structured-DAG Proof-of-Stake protocol that hides slot leaders until block publication. Section IV-A gives an end-to-end overview of the per-epoch pipeline. Section IV-B defines epochs, rounds, and slot assignment from the shuffled commitment bucket. Section IV-C to Section IV-F specify the commitment construction, signature choices, and how blocks reveal secrets for public verification and Section IV-G details shuffling, the grid mapping, and the DAG split and merge rules.

A. Protocol Overview

In the following, we describe the key ingredients of our approach. The protocol proceeds in three phases per epoch (see Figure 2):

1) **Registration and commitment generation**. Validators lock stake, construct a zk-SNARK commitment, and publish it for the next epoch. Commitment generation

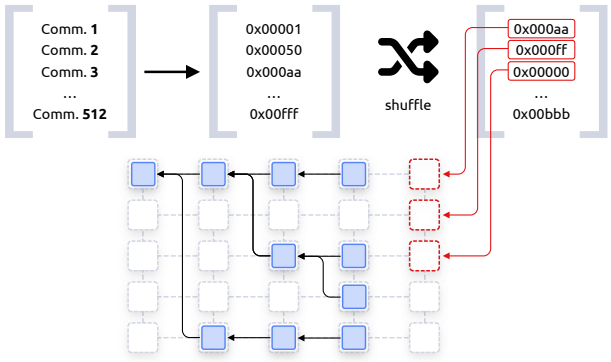

Figure 1: Commitment shuffling and slot mapping.

starts early in the epoch and is randomized with short delays to reduce timing side channels.

2) **Shuffling**. Nodes agree on a randomized ordering of commitments using a deterministic shuffle seeded by a randomness beacon. The shuffled bucket is interpreted as a 2D matrix that maps commitments to slots.
3) **Block production and adaptive DAG.** Elected leaders reveal their identity by publishing blocks in their assigned slots. Each round, branches can split or merge based on transaction load.

Shuffling illustrated in Figure 1 displays commitments shuffling associated with blocks inside DAG structure. It is a process of computing the block producers assigned slots for the next epoch, which can only be done after the node agreement on a single, uniformly randomized ordering of the collected commitments (bucket). To get there, each node performs the following:

1) **Commitment identification and sorting**. Each commitment, comprising its public inputs and ZKP proof, is serialized using Base64 encoding and hashed using Keccak-256. These hashes serve as commit unique identifiers. Nodes then sort the list of commitments by commitment hashes using lexicographical ordering.
2) **Deterministic shuffling**. To achieve unpredictable yet deterministic permutation of the bucket, the Whisk describes Feistelshuffle [20] while utilizing the RANDAO beacon. In our implementation, we abstracted RANDAO and used fixed key strings for shuffling. Moreover, we use shuffle implementation in Go, which swaps the positions of commitments in the bucket.
3) **Bucket hash agreement**. During the last round, nodes share the hash of their shuffled bucket to compare if they have the same content. If the consensus is reached, nodes use this commitment in the upcoming epoch. The bucket is then interpreted in the form of a two-dimensional matrix to determine the correct slot by row and column.

B. Epochs and Slots

We assume epochs consisting of slots that are processed within rounds. Each round provides a time window for block production (12 seconds by default, as in Ethereum) and contains a fixed number of parallel slots (a power of two),

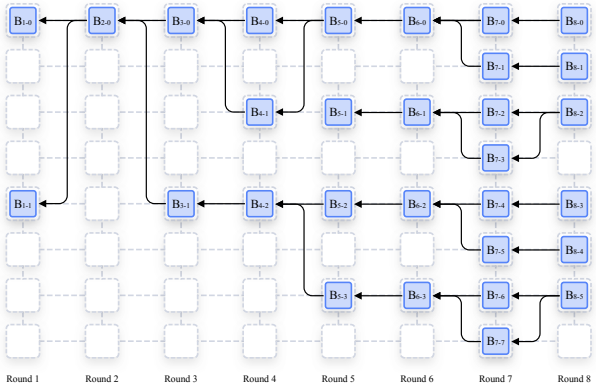

Figure 2: SSLE-DAG structure of a single epoch: consisting of rounds (depicted horizontally), and round contains slots (depicted vertically). Slot may or may not contain a block.

where each slot can contain at most one block. The slots correspond to the parallel branches of the protocol's structured DAG, so a block additionally carries its DAG coordinates (row, column, depth) and parent references needed for branching and merging decisions. The assignment of the leader is derived from the commitment bucket per-epoch. After shuffling, the bucket is interpreted as a 2D matrix and mapped to rounds and slots by column-major addressing, so each commitment maps to exactly one slot in the epoch. The elected leader for a slot is the validator who can later open the corresponding commitment when publishing the block for that slot (see Figure 2).

Note (experimental commitment multiplicity). An epoch contains C leader slots, where C equals the number of rounds times the number of parallel branches. In experimental deployment, the validator set can be smaller than C, which would *permanently* leave many unassigned slots. In this case, we optionally allow each validator to publish up to V_{max} commitments per epoch, distinguished by a bounded commitment variant index $v \in \{0, \ldots, V_{max} - 1\}$ that is encoded into the commitment message. In real deployment, V_{max} is set to 1, so each validator publishes only a single commitment per epoch.

C. Digital Signature Scheme

We use the Twisted Edwards variant of the EdDSA (Edwards-curve Digital Signature Algorithm), instantiated over the BLS12-381 curve, rather than the more common ECDSA. This choice is driven by two practical advantages: Twisted Edwards curves map more efficiently into arithmetic circuits used by zk-SNARKs, and EdDSA produces deterministic signatures – an important property for our commitment construction. In contrast, under certain conditions, an ECDSA signature pair (r, s) can be used to recover the signer's public key [8], which would undermine the commitment scheme's goal of keeping leader identities confidential and thus increase DoS vulnerability.

Each node generates a private scalar a and derives its public key $A = aB$, where B denotes the standard base point of the BLS12-381 curve. The public key serves as the node's unique

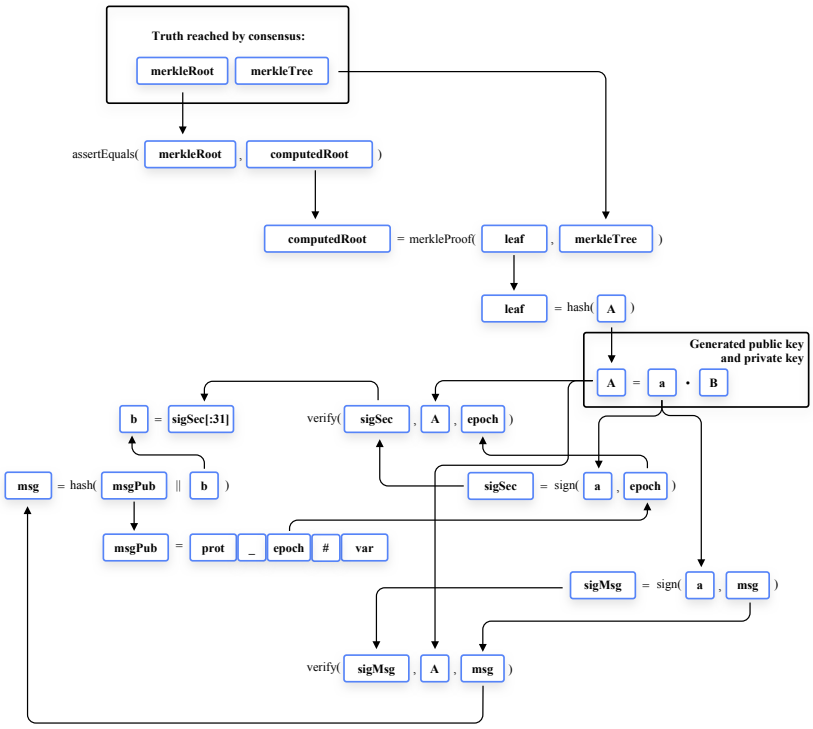

Figure 3: Commitment scheme in our SSLE-DAG.

consensus-layer identifier and as proof that the node has locked stake. In a production deployment the same key pair would be used to sign blocks and transactions, preventing third-party impersonation. Finally, because both values are standard elliptic-curve key material, the public key is mathematically and uniquely bound to its corresponding private scalar.

D. Commitment Structure and Circuit Variables

Figure 3 provides a graphical representation of our commitment scheme and the relations between its parts. Each commitment includes a zk-SNARK proof for a fixed arithmetic circuit that binds (i) epoch-specific metadata, (ii) eligibility of the prover among registered validators, and (iii) uniqueness constraints that limit commitment multiplicity. Our circuit uses the following inputs.

Public inputs (6 total):

- **EdDSA signature ($sigMsg$) over message (m)** represented as the compressed point R (x, y) and scalar s.
- **Public message ($pubMsg$)** used together with secret b to derive m.
- **Epoch number**, which must match the epoch encoded in $pubMsg$.
- **Merkle root** of the eligible validator set.

Private inputs (26 total):

- **Prover's public key** coordinates (A_x, A_y).
- **Secret EdDSA signature ($sigSec$)** (same representation as $sigMsg$).
- **Secret** b derived from $sigSec$.
- **Hash** $m = MiMC(pubMsg||b)$.
- **Merkle membership witness**: starting leaf (hash of the prover's public key), nine siblings' hashes, and nine path indices.

E. Commitment Construction and Message Flow

A commitment must be verifiable without revealing the leader's identity before publishing a block. For this reason, the prover's public key is treated as a secret circuit witness, while the circuit enforces correctness through (i) EdDSA verification and (ii) Merkle membership under the epoch's public Merkle root. The private signing key is never included in the circuit.

We use two messages with distinct roles. The public message $pubMsg$ is an externally validated descriptor that encodes protocol version, epoch identifier, and an optional bounded variant index used only in small-scale experiments. The signed message msg is computed inside the circuit as $m = MiMC(pubMsg||b)$ and is the message actually bound to the prover via $sigMsg$. This separation prevents brute-force linkage of a published commitment to a candidate public key, because an adversary would need the hidden m and b values to test signature validity.

The secret b must be (i) unknown to others until reveal and (ii) not freely choosable, otherwise a validator could create many distinct commitments and bias leader selection. We therefore derive b deterministically from a second signature $sigSec$ over an epoch-specific string, then keep $sigSec$ and b as circuit witnesses. The circuit verifies $sigSec$ against the hidden public key and uses b only through the constraint $m = MiMC(pubMsg||b)$, so the prover cannot arbitrarily vary b without invalidating the proof.

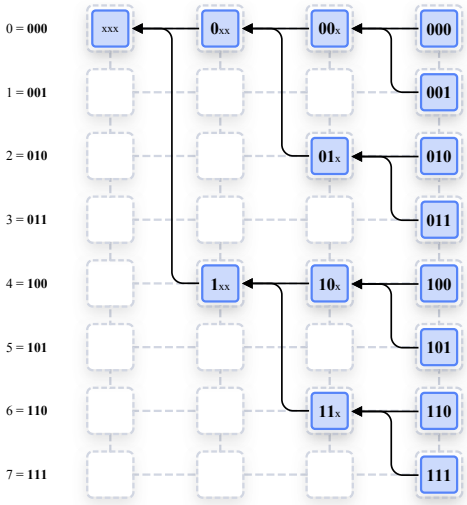

Figure 4: Hash mask for transaction applied in DAG structure. The x value states that the bit value is irrelevant to the result.

Finally, to prevent unauthorized keys from publishing commitments, the prover additionally proves that its hidden public key is included in the epoch's validator set represented by a fixed-depth Merkle tree, whose root is public and agreed by the network for the epoch. This brings eligibility verification without exposing the identity of the future leader until the block is produced and the commitment is opened.

F. Verification of Secret Variables

When a validator produces a block, it opens the commitment used for the corresponding slot by revealing its public key A, the secret value b, and the auxiliary signature $sigSec$ from which b was deterministically derived. We refer to $sigSec$ as a *hidden* (or secret) signature because it is not published during the commitment phase and is only revealed together with the block, which preserves the unlinkability of the commitments to the validators before the revelation of the leader. Upon receiving a block, a node verifies that $sigSec$ is a valid EdDSA signature under A in the epoch-specific message and then checks that the revealed b matches the deterministic derivation of $sigSec$. Recomputing $m = MiMC(pubMsg||b)$ produces the message bound by the public signature of the commitment $sigMsg$. Together with prior zk-SNARK verification of the commitment, these checks ensure that the block producer is the original author of the opened commitment and that the revealed secret is consistent with the committed witness.

G. Shuffling Details and DAG Structure

Within each epoch, the protocol arranges blocks in a fixed-size grid that follows a binary-tree style branching rule, which enables systematic branch splitting and merging, thus forming DAG structure. Blocks are aligned by the round in which they are produced and are enforced by round-level time synchronization. At the beginning of each round, the nodes derive the set of active slots from the previous round's branching decisions, namely whether a branch continues, splits into two, or whether two branches merge into one. Each node then checks whether any of its commitment positions corresponds to the currently active slots.

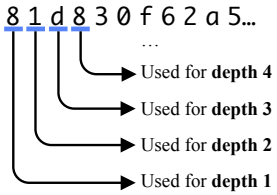

Figure 5: Hash mask for transaction. Each individual hexadecimal number is used to determine the depth of the slot.

After shuffling, the commitment bucket is interpreted as a two-dimensional matrix and mapped to grid coordinates in column-major order, so that each commitment deterministically selects a unique `(round, branch)` position. Let c denote the round index (column), let r denote the branch index (row), and let $B_{\max}$ denote the maximum number of parallel branches. The corresponding linearized position is

$$i = c \cdot B_{\max} + r \tag{1}$$

In our implementation, $B_{\max} = 8$ by default and the prototype supports up to 64 branches. Increasing $B_{\max}$ or the number of rounds per epoch increases the required number of commitments and thus increases per-epoch processing cost. During each round window, nodes concurrently (i) produce and broadcast blocks when they control commitments for active slots, (ii) validate and process incoming blocks, (iii) process commitments for the upcoming epoch, and (iv) process transactions. The interpretation of the epoch grid is illustrated in Figure 2, and the shuffling-to-slot mapping is in Figure 1.

A branch split is triggered when a block exceeds a configured transaction-count threshold. Upon splitting, transactions in the affected branch are sorted by hash, inspired by Sycomore [2], and then deterministically assigned to one of the two successor branches. The process is as follows. The node inspects the hexadecimal digit of the transaction hash at the current branching depth, as illustrated in Figure 4 and Figure 5. The digits in the range 0–7 are assigned to the first successor (slot label 0), while the digits in the range 8–15 are assigned to the second successor (slot label 1). The split operation always keeps one successor as the direct continuation of the original branch, while the second successor is placed by a half-step increase in nesting depth. Because leader assignments follow the commitment-to-slot mapping in Equation 1, validators are not guaranteed to produce blocks in deeper branches. Under an unbiased shuffling and randomness procedure, all validators remain equally likely to be assigned to any slot in the next epoch. A merge is triggered when both candidate branches fall below the merge threshold, which indicates that continuing along a single merged path is more efficient. The selection rule mirrors the splitting logic but is applied in reverse order, as shown in Figure 2.

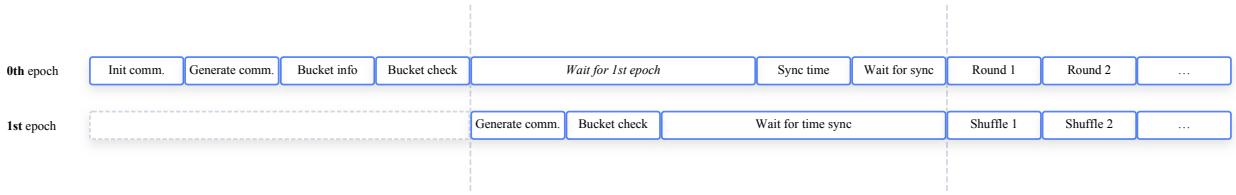

Figure 6: Initial SSLE-DAG protocol flow.

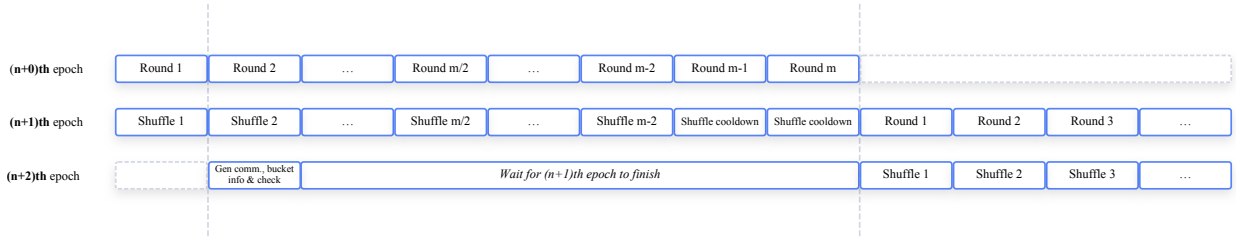

Figure 7: SSLE-DAG protocol flow for the running epochs.

V. Evaluation

We evaluate the proposed SSLE-DAG protocol using a containerized simulator that deploys the full implementation as a set of Docker services. We first summarize the cost of zk-SNARK commitments, then we describe the simulation environment and network model, and finally summarize results from the two experiments.

A. ZKP Circuit Performance Analysis

The zk-SNARK circuit is designed to keep proving costs practical while retaining the functionality required for SSLE commitments. The dominant components are two EdDSA signature verifications and a Merkle membership proof, while MiMC hashing contributes a smaller fraction due to its low multiplicative complexity. In the current prototype, the circuit comprises roughly 21k constraints, about 14k for two EdDSA verifications, about 6k for a Merkle proof (depth 9 for 512 public keys), and about 1k for MiMC. On common hardware (desktop computer – AMD Ryzen 9 7900X3D and 64GB of RAM), the average proof generation time is 600–1,100 ms. Verification time is effectively constant with respect to circuit size and remains negligible for validators (thanks to succintness of zk-SNARKs). Proving is the main performance bottleneck, but it is incurred off-chain and only once per epoch per validator, so it does not directly affect block production latency. Protocol-level optimizations such as alternative proving systems or even GPU-accelerated proving can be incorporated without changing the commitment semantics.

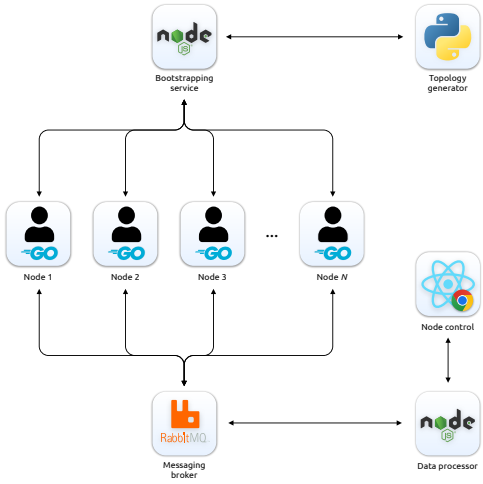

Figure 8: Simulation environment architecture. Components represent containers inside a Docker environment.

B. Simulation Environment and Network Model

The implementation is packaged for execution in a containerized environment (see Figure 8), which enables reproducible deployment with minimal configuration. The software stack requires Python, Docker, and Docker Compose, and an initialization script builds images, generates a Compose configuration for the requested number of nodes, and launches all services. Upon acceptation of the publication we will disclose the repository containing full implementation.

The simulator follows a multi-service architecture. Nodes communicate over a simulated peer-to-peer network, while experiment monitoring and control is performed by auxiliary services. Only the web interface and the data-processor are exposed externally, node control is performed through commands transmitted on the internal network via RabbitMQ.

To model realistic network conditions, we use a latency dataset derived from round-trip-time measurements between 246 globally distributed servers. The original measurements were published by WonderProxy[1] and collected repeatedly on December 20, 2024. For each location, we compute the mean RTT for every other server inbetween and store the resulting matrix for efficient lookup during simulation. Because fewer than 1% of the entries were missing, we filled the gaps with manual measurements taken on December 31, 2024. The resulting dataset contains 246 server records with geographic coordinates and mean latencies to the other 245 sites.

[1] https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/

TABLE I: Parameters used in the 60 nodes experiment.

Parameter	Value
Number of nodes	60
Time allocated per round	8 seconds
Maximum number of parallel chains (DAG)	8
Number of rounds per epoch	20
Maximum number of transactions per block	1000+1
Split threshold (min. transactions per block)	800
Merge threshold (max. transactions per block)	275
Minimum number of commitments per node	1
Maximum number of commitments per node	3
Number of entries in Merkle tree of public keys	512
Depth of Merkle tree of public keys	9

The ping packet size is not specified in the source dataset and typical ping payloads are on the order of tens of bytes, so raw RTT alone may underestimate block propagation delay. We address this in two ways. First, we aim to keep blocks small via the parallel DAG structure. Second, we add an extra per-node random delay (jitter) to model congestion, protocol overhead, and payload-dependent effects. Scenarios requiring substantially larger blocks, for example smart-contract execution, are left for future work.

C. Experiments and Results

We report results from two experiments that stress the protocol under sustained transaction load. Both experiments were executed on an Intel Xeon Gold 6548N (x86_64, 60 cores, 120 GB RAM) hosted on a cloud VM. All nodes start with 1M pre-generated transactions to ensure constant demand.

1) Experiment 1: 60-node network: The first experiment evaluates functional correctness, fairness, and throughput in a 60-node network (simulated on the one machine). The configuration (parameters) is listed in Table I. Immediately after launch, blocks fill, and the DAG expands until it reaches the configured maximum of eight parallel chains.

Fairness is evaluated by comparing the number of blocks produced by each node (see Figure 10). To illustrate temporal dynamics, Figure 9 reports blocks aggregated in 8-second intervals. During the experiment, the nodes produced 815 blocks over 104 rounds and processed 815,815 transactions, generating 990.07 TPS on average and 1001 TPS at peak. Commitment utilization was 88.89%, which means that when all parallel chains were active, 11.11% of commitments did not map to active slots.

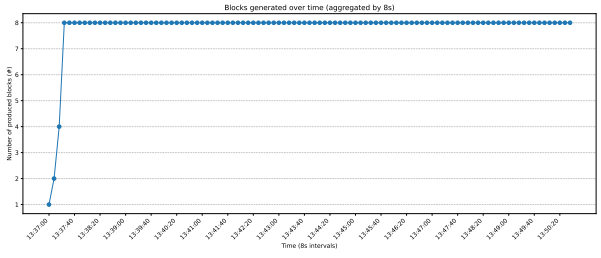

Figure 9: Blocks generated over time in the 60-node experiment, aggregated in 8-second intervals.

TABLE II: Results of the 40 node experiment.

Metric	Value
Number of completed rounds	161
Number of completed blocks	1,086
Minimum blocks produced per node	20
Maximum blocks produced per node	36
Mean blocks produced per node	27.15
Standard deviation of block production	3.60
Utilization of commitments	50%
Mean interval of block generation	0.59 seconds
Total number of processed transactions	1,024,403
Total time span of simulation	10 minutes 40 seconds
Average transaction throughput (TPS)	1600.63 transactions/second
Peak transaction throughput (TPS)	2002 transactions/second

2) Experiment 2: 40-node network: The second experiment reduces the network size, but targets higher block-production rates by shortening rounds to 4 seconds and increasing the epoch length to 40 rounds. The configuration was similar to the previous experiment, with a difference in maximum commitments per node to 16. This longer run exhibits visible branch merges, and after all transactions are processed, the DAG converges back toward a single branch. Commitment utilization is reduced to 50%, still yielding comparable fairness while allowing higher throughput (shown in Figure 11 and Figure 12). Across 161 completed rounds, the nodes produced 1,086 blocks and processed 1,024,403 transactions. This configuration achieves 1600.63 TPS on average and 2002 TPS at peak, with a mean inter-block interval of 0.59 seconds. The summarization of the results is shown in Table II.

VI. Discussion and Limitations

A. Commitment-Scheme Requirements

We discuss security at the level of the protocol model, where the commitment scheme satisfies the following requirements.

- **Eligibility enforcement**. Each commitment proves membership of the validator in the epoch-specific registry via a Merkle inclusion proof, so only registered validators can create acceptable commitments.
- **Uniqueness and bounded influence**. Deriving the secret value b from a deterministic auxiliary signature prevents validators from freely generating multiple independent

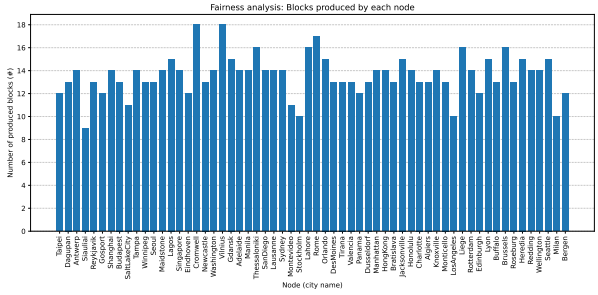

Figure 10: Histogram of produced blocks per node in the 60-node experiment (nodes are identified by a city, instead of the public key).

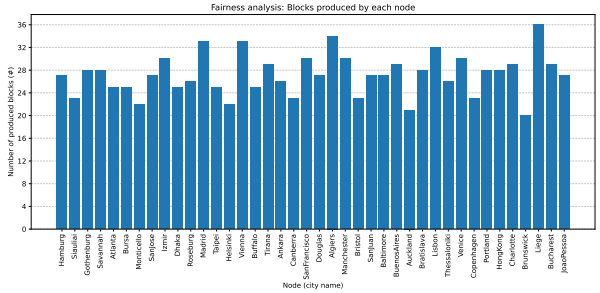

Figure 11: Histogram of producesd blocks per node in the 40-node experiment.

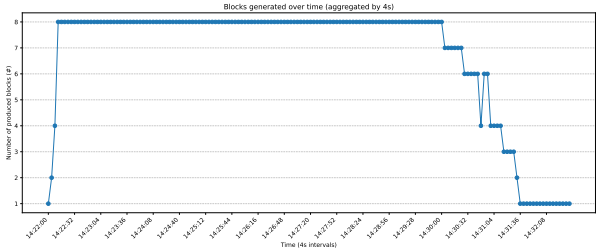

Figure 12: Blocks generated over time in the 40-node experiment, aggregated in 4-second intervals.

commitments, thereby limiting their ability to bias the selection of the leader. This limits grinding over leader assignments and aligns with SSLE requirements that aim to restrict manipulation of election outcomes. [5], [13]

- **Unpredictability and anonymity of the leader until revealed**. The validator's public key remains hidden inside the zk-SNARK proof until the commitment is opened. This property is the key defense against targeted DoS on upcoming leaders, and it matches the SSLE goal that only the winner learns the outcome before revealing it.
- **Public verifiability and noninteractivity**. Any node can verify commitments and openings using only public inputs, with no interaction and no trusted online party.
- **Compatibility with parallel slots in the structured DAG**. The deterministic mapping from shuffled commitments to epoch slots yields a unique leader assignment per slot, which supports parallel block production while keeping the election rule consistent across nodes [2].

B. Alternative Design Without Leader Anonymity

A simpler alternative would reveal the public key of the validator in the commitment and prove only eligibility and slot ownership using signatures plus a registry check. This reduces circuit complexity and eliminates the privacy layer, but it also makes leaders predictable within the epoch, reintroducing the primary DoS vector to mitigate that SSLE is designed to mitigate. In contrast, the zk-SNARK-based design decouples eligibility verification from identity revelation and therefore preserves the intended DoS resistance at the cost of higher proving complexity.

C. Limitations and Future Work

- **Simulator scalability and system-level constraints**. The containerized simulator was stable up to about 99 nodes, after which connections failed due to host network limits, so we report results for 40 and 60 nodes. Scaling further likely requires system-level tuning or a different networking backend and validation on distributed infrastructure instead of a single host. Our latency model emulates geo-distribution with added jitter, but we do not evaluate adversarial settings such as targeted DoS, eclipse attacks, or coordinated multi-slot censorship.
- **Abstracted transaction validation**. We abstracted balance checks and double-spend validation, which reduces per-block processing costs and likely overestimates throughput relative to real full deployment.
- **Randomness beacon and bias resistance**. The shuffle seed is stubbed with fixed keys in the experiments. A real deployment requires an unbiased seed and also requires defenses against proposer influence over randomness, including last-revealer-style withholding attacks and related grinding. Integrating a publicly verifiable distributed beacon such as [12] or combining on-chain accumulation with strong accountability and slashing is possible.
- **Trusted setup and circuit completeness**. In our setup prototype, zk-SNARK used a single-machine-trusted setup, while a production deployment should rely on the MPC ceremony so that security holds if at least one participant is honest [25]. In addition, the public-key hash check inside the circuit is disabled due to an implementation issue in the current gnark workflow, so the circuit must be revalidated end-to-end after this dependency is resolved.
- **Incentive layer**. In the prototype, we focussed on functional correctness and throughput, while we did not implement an incentive layer, slashing for missed slots, censorship behavior, or equivocation in parallel branches.

VII. Conclusion

We presented SSLE-DAG, a PoS consensus protocol that integrates a single secret leader election with a structured adaptive DAG to support parallel block production while reducing exposure to targeted DoS against upcoming leaders. Leader eligibility is realized via an epoch commitment shuffle that assigns validators to slots without revealing proposers in advance, while deterministic split and merge rules adapt the number of active branches (parallel chains) based on transaction demand. In a geo-distributed simulation calibrated with measured latency, our prototype sustains throughput up to 1,600 TPS and maintains balanced block production across validators, which indicates that adaptive parallelism can raise capacity without altering the core SSLE workflow. Future work includes evaluating adversarial settings, implementation of the full protocol with an incentive layer, and smart contract support.

References

[1] Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Advances in Cryptology – ASIACRYPT 2016. Lecture Notes in Computer Science, vol. 10031, pp. 191–219. Springer (2016), https://pure.royalholloway.ac.uk/ws/files/26913721/mimc.pdf

[2] Anceaume, E., Guellier, A., Ludinard, R., Sericola, B.: Sycomore: A permissionless distributed ledger that self-adapts to transactions demand. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA). pp. 1–8 (2018). https://doi.org/10.1109/NCA.2018.8548053

[3] Baldimtsi, F., Chalkias, K.K., Ji, Y., Lindstrøm, J., Maram, D., Riva, B., Roy, A., Sedaghat, M., Wang, J.: zklogin: Privacy-preserving blockchain authentication with existing credentials. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. p. 3182–3196. CCS '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3658644.3690356, https://doi.org/10.1145/3658644.3690356

[4] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 781–796. USENIX Association (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[5] Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. Cryptology ePrint Archive, Paper 2020/025 (2020), https://eprint.iacr.org/2020/025

[6] Boneh, D., Partap, A., Rotem, L.: Post-quantum single secret leader election (ssle) from publicly re-randomizable commitments. Cryptology ePrint Archive, Paper 2023/1241 (2023), https://eprint.iacr.org/2023/1241

[7] Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-snark parameters in the random beacon model. IACR Cryptology ePrint Archive **2017**, 1050 (2017), https://eprint.iacr.org/2017/1050

[8] Brown, D.R.: Sec 1: Elliptic curve cryptography. Certicom Research **2** (2009)

[9] Burianová, T., Perešíni, M., Homoliak, I.: Secret Leader Election in Ethereum PoS: An Empirical Security Analysis of Whisk and Homomorphic Sortition under DoS on the Leader and Censorship Attacks. arXiv preprint arXiv:2509.24955 (2025)

[10] Buterin, V.: The meaning of decentralization. https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274 (2017), accessed: 2026-01-14

[11] Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems. p. 34–50. EuroSys '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3492321.3519594, https://doi.org/10.1145/3492321.3519594

[12] drand: drand explained, https://docs.drand.love/about/

[13] Eskandarian, S.: Single secret leader election (ssle) slides. PDF, https://crypto.stanford.edu/~saba/slides/SSLE.pdf, accessed: 2026-01-14

[14] Ethereum Improvement Proposals: Eip-7441: Slot-based proposer-builder separation. https://eips.ethereum.org/EIPS/eip-7441 (2024), accessed: 2026-01-12

[15] Freitas, L., Tonkikh, A., Bendoukha, A.A., Tucci-Piergiovanni, S., Sirdey, R., Stan, O., Kuznetsov, P.: Homomorphic sortition–secret leader election for pos blockchains. arXiv preprint arXiv:2206.11519 (2022)

[16] Gagol, A., Lesniak, D., Straszak, D., Swietek, M.: Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. p. 214–228. AFT '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3318041.3355467, https://doi.org/10.1145/3318041.3355467

[17] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: SOSP. ACM (2017)

[18] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems, p. 203–225. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3335741.3335750

[19] Groth, J.: On the size of pairing-based non-interactive arguments. IACR Cryptology ePrint Archive **2016**, 260 (2016), https://eprint.iacr.org/2016/260

[20] Kadianakis, G.: Whisk: A practical shuffle-based ssle protocol for ethereum. online (2022), https://hackmd.io/@asn-d6/HyD3Yjp2Y

[21] Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. p. 165–175. PODC'21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3465084.3467905, https://doi.org/10.1145/3465084.3467905

[22] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO'17. Springer (2017)

[23] Li, C., Li, P., Zhou, D., Yang, Z., Wu, M., Yang, G., Xu, W., Long, F., Yao, A.C.C.: A decentralized blockchain with high throughput and fast confirmation. In: 2020 USENIX Annual Technical Conference (USENIX ATC 20). pp. 515–528. USENIX Association (Jul 2020), https://www.usenix.org/conference/atc20/presentation/li-chenxing

[24] Perešíni, M., Benčić, F.M., Hrubý, M., Malinka, K., Homoliak, I.: Incentive Attacks on DAG-Based Blockchains with Random Transaction Selection. In: IEEE International Conference on Blockchain, Blockchain 2023, Hainan, China, December 17-21, 2023. IEEE (2023)

[25] Pruden, A., Matlala, A.: Setup ceremonies (2021), https://zkproof.org/2021/06/30/setup-ceremonies/

[26] Sompolinsky, Y., Lewenberg, Y., Zohar, A.: Spectre: A fast and scalable cryptocurrency protocol. IACR Cryptol. ePrint Arch. **2016**, 1159 (2016)

[27] Sompolinsky, Y., Wyborski, S., Zohar, A.: PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. In: ACM AFT '21. pp. 57–70. ACM, New York, NY, USA (2021). https://doi.org/10.1145/3479722.3480990, https://doi.org/10.1145/3479722.3480990

[28] Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2705–2718. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3559361, https://doi.org/10.1145/3548606.3559361

[29] Tornado Cash: Tornado cash: A privacy tool for ethereum. https://github.com/tornadocash/tornado-core (2021), accessed: 2025-02-03

[30] Westerkamp, M., Eberhardt, J.: zkrelay: Facilitating sidechains using zksnark-based chain-relays. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 378–386 (2020). https://doi.org/10.1109/EuroSPW51379.2020.00058