Mobius: Enabling Byzantine-Resilient Single Secret Leader Election with Uniquely Verifiable State

Hanyue Dou*†¶, Peifang Ni*§¶, Yingzi Gao‡ and Jing Xu*†§‖
*Institute of Software, Chinese Academy of Sciences
†School of Computer Science and Technology,
University of Chinese Academy of Sciences
‡Shandong University
§Zhongguancun Laboratory, Beijing, P.R.China
Email: {hanyue2021, peifang2020, xujing}@iscas.ac.cn
gaoyingzi@sdu.edu.cn

***Abstract*—Single Secret Leader Election (SSLE) protocol facilitates the election of a single leader per round among a group of registered nodes while ensuring unpredictability. Ethereum has identified SSLE as an essential component in its development roadmap and has adopted it as a potential solution to counteract potential attacks. However, we identify a new form of attack termed the *state uniqueness* attack that is caused by malicious leaders proposing multiple publicly verifiable states. This attack undermines the property of *uniqueness* in subsequent leader elections and, with high probability, leads to violations of fundamental security properties of the over-layer protocol such as liveness. The vulnerability stems inherently from the designs reducing the uniqueness guarantee to a unique state per election, and can be generalized to the existing SSLE constructions. We further quantify the severity of this attack based on theoretical analysis and real-world executions on Ethereum, highlighting the critical challenges in designing provably secure SSLE protocols.**

To address the *state uniqueness* attack while ensuring both security and practical performance, we present a universal SSLE protocol called Mobius that does not rely on extra trust assumptions. Specifically, Mobius prevents the generation of multiple verifiable states for each election and achieves a unique state across consecutive executions through an innovative *approximately-unique randomization* mechanism. In addition to providing a comprehensive security analysis in the Universal Composability framework, we develop a proof-of-concept implementation of Mobius, and conduct extensive experiments to evaluate the security and overhead. The experimental results show that Mobius exhibits enhanced security while significantly reducing communication complexity throughout the protocol execution, achieving over 80% reduction in the registration phase.

I. Introduction

Leader election serves as a fundamental component of distribute systems, designed to select one or more nodes to perform tasks such as distributing proposals or aggregating cryptographic certificates, thereby ensuring both system security and efficiency. With the rapid rise of decentralized distributed systems (e.g., blockchains), new security challenges have prompted an increased focus on research related to leader election. Traditional leader election methods (e.g., round-robin [1]–[5]) guarantee leader uniqueness and public verification but suffer from pre-exposure of leaders, enabling the Byzantine adversary to execute preemptive corruptions and gain an additional attack advantage. Furthermore, subsequent efforts seek to realize unpredictability through randomized approaches (e.g., verifiable random functions (VRFs) [6]), and these approaches facilitate a private and non-interactive leader election, thereby mitigating risks such as denial-of-service (DoS) attacks and enhancing protocol security [7]–[10].

However, existing unpredictable or secret leader election schemes [7]–[11] may select an indeterminate number of potential leaders in each instance, providing only a probabilistic expectation regarding the total number of leaders over an extended period. The simultaneous election of multiple leaders can lead to significant resource wastage (e.g., computational power in Proof-of-Work systems). More critically, this issue considerably diminishes overall efficiency and degrades fault tolerance of distributed systems [12], [13]. For instance, blockchain systems that utilize probabilistic multi-leader election mechanisms not only experience higher confirmation latency (e.g., approximately one hour in Bitcoin [14]) but also exhibit weakened resilience against attacks (e.g., grinding attacks in Proof-of-Stake blockchains [11]), dramatically compromising system practicality.

This state of affairs has spurred research into the security property of *uniqueness* (or *singleness*) in the secret leader election process. Specifically, *uniqueness* requires that at any given instance, exactly one node is elected while preserving its secrecy and public verifiability. In pursuit of this design goal, Boneh et al. [15] formalize the concept of *Single Secret Leader Election* (SSLE), which enables a group of registered nodes to engage in continuous elections and deterministically yields a single secretly elected node per election.

Fortunately, contemporary distributed systems, such as Ethereum, are increasingly recognizing the importance of leader uniqueness in addition to the established properties of unpredictability and fairness during leader election. In fact,

¶ Hanyue Dou and Peifang Ni are co-first authors.
‖ Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.242407
www.ndss-symposium.org

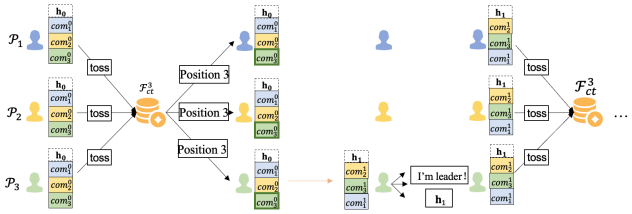


Fig. 1: The framework of shuffle-based SSLE, where **h** denotes a node list consisting of commitments.

Ethereum leverages leader uniqueness to implement an efficient optimization (i.e., proposer boosting [16]), which aims to mitigate attacks exemplified by the balance attack [17]. However, Ethereum simply deploys a leader election mechanism that depends on a chain-embedded random value to achieve the favorable *uniqueness* property with a weak unpredictability. This approach enables the adversary to undermine *uniqueness* via strategically inducing network partitions, particularly withholding or selectively broadcasting messages. Even worse, the collapse of *uniqueness* can revive the attacks that compromise liveness by preventing finalization (cf. Section III).

Beyond that, numerous efforts have been dedicated to the implementation of SSLE [15], [18]–[21], and the shuffle-based approach [15], [18], [21] stands out for its computational efficiency and dynamic adaptability. Informally, this framework organizes the participating nodes (indexed as $i \in [1, n]$) into a commitment list, where each entry uniquely corresponds to a node (cf. Fig. 1). During the election process, the nodes query a decentralized random beacon to obtain a publicly verifiable random index $\gamma \in [1, n]$. Subsequently, the node owning the γ-th position in the list is elected as leader, which can assert its leadership by generating a zero-knowledge proof demonstrating its ownership of the corresponding commitment. Delving into the essence of the shuffle-based approach, the committed node list preserves node identity secrecy, while the random beacon ensures both the singleness and unpredictability of the elected leader from this list, thereby facilitating the *secret single* leader election. However, since each node's identity is deterministically mapped to its commitment (i.e., a specific position in the list), the self-revelation of the current leader compromises secrecy in subsequent elections. To maintain secrecy across continuous elections, it becomes necessary to periodically randomize the node list[1] after each election.

Nevertheless, existing approaches [15], [21] remain conceptual and fail to efficiently address the randomization of state (i.e., the node list) while maintaining leader singleness. Even though nodes broadcast the randomized state and post it on the blockchain, the current shuffle-based solutions neglect the consensus process as well as the inherent confirmation latency of it. As a result, the nodes may continue to participate in the election process using a randomized yet unstable state derived from the previous election. Additionally, a malicious leader can generate multiple randomized states and strategically send them to honest nodes, thereby fundamentally undermining the property of *uniqueness* and subsequently facilitating further attacks on the over-layer protocol (e.g., the balance attack).

By leveraging the aforementioned observations, to ensure the secure and consecutive execution of SSLE protocol within distributed systems, two critical technical challenges must be addressed: the consecutive randomness generation and the state uniqueness enforcement. In slightly more detail, the consecutive generation of randomness after each election effectively randomizes the state that serves as the foundation for ensuring both leader secrecy and unpredictability. Additionally, maintaining state uniqueness during each randomization is vital for guaranteeing leader uniqueness among honest nodes.

Therefore, the issue of state uniqueness stemming from adversarial message propagation can extend to other frameworks of SSLE constructions, revealing the fundamental weaknesses inherent in current SSLE designs. This presents a foundational challenge in the design of SSLE: each leader election must simultaneously generate a state that is randomized, publicly verifiable, and globally unique. Hence, this tripartite requirement, i.e., *secrecy*, *verifiability*, and *uniqueness*, naturally raises our core research question:

Can we develop a state uniqueness scheme that enables a provably secure SSLE protocol while simultaneously achieving high performance?

A. Our Contribution

In this paper, we contribute to the rigorous understanding of SSLE and affirmatively answer the aforementioned question by introducing a novel shuffle-based SSLE protocol named Mobius. The specific contributions are outlined as follows:

- A systematic analysis of state uniqueness. We investigate the security of existing SSLE constructions [15], [18], [21], identifying a new form of attack termed the *state uniqueness* attack. This attack specifically targets the essential assurance for achieving the uniqueness property, leading honest nodes to recognize different elected leaders, even when supported by a globally unique random beacon. To quantify the effectiveness of *state uniqueness* attack, we use Ethereum [22] as a case study of leader-based consensus protocols to evaluate the implications. Specifically, we simulate each election process using multiple publicly verifiable node lists, and the results demonstrate that such violations can further exacerbate balance attacks even when employing the boosting mechanism, thereby undermining fundamental security properties of the over-layer protocol (e.g., liveness). More critically, the *state uniqueness* attack is generalizable, as it stems from an inherent weakness in existing SSLE architectural paradigms [15], [18]–[21].
- Approximately-Unique Randomization: a comprehensive solution to state uniqueness. We introduce a novel paradigm to realize the publicly verifiable randomization with the final guarantee of global uniqueness. Within the randomization process, each leader not only performs the current

[1]The process of randomizing the node list consists of two phases: (1) updating the commitment values to obtain refreshed commitments (commonly called re-randomization), and (2) shuffling the list to disrupt the original order.

TABLE I: Comparisons with other protocols.

Methodology	Construction	Assumption	Communication Cost (off-chain)	On-chain Cost		Round	Byzantine Adversary	Reconfiguration -free
				Election	Registration			
iO-based	BEH+20 [15]	iO	$O(1)$	$O(1)$	O(1)	1	□	✗
TFHE-based	BEH+20 [15]	-	$O(t)$	$O(t)$	$O(N)$	1	□	✗
PEKS-based*	CFG23+ [19]	random beacon	$O(N^2)$	$O(1)$	$O(\log^2 N)$	3	✗	✗
MPC-based	BPL23 [20]	ROM	$O(n^2)$	$O(1)$	$O(1)$	$O(\log n)$	✓**	✗
Shuffle-based	BEH+20 [15]	random beacon	$O(n^2)$	$O(n)$	$O(n^2)$	1	□	✓
	CFG22 [21]	random beacon	$O(n^2)$	$O(n)$	$O(n^2)$	1	□	✓
	BBH+22 [18]	random oracle	$O(n^2)$	$O(n)$	$O(n^2)$	1	□	✓
Shuffle-based	**Ours**	random beacon	$O(n^2)$	$O(n)$	$O(n)$	1	✓	✓

[a] In this table, the notation n denotes the number of registered nodes that participate in the election process, while N denotes the number of all nodes in the system. The notation t denotes the threshold of the threshold fully homomorphic encryption (TFHE) scheme.
[b] For the analysis of Byzantine adversary model, □ denotes the work does not consider the adversarial behaviors (e.g., selective message propagation) across the constructions, ✗ denotes that the work considers the Byzantine behaviors but fails to deal with it, and ✓ denotes that the work considers the conditions of the existence of Byzantine adversary and is Byzantine-resilient.
[c] The metric of on-chain cost quantifies the on-chain storage requirements when deploying the above SSLE protocols in a blockchain environment. Since this critical performance indicator primarily comprises two parts, i.e., the declaration of election outcome and generation of state transitions for subsequent elections, the metric can also be extended to other applications of SSLE protocols to measure the cost of the leadership declaration.
* Here, PEKS is abbreviation of *Public Key Encryption with Keyword Search*.
** This work considers the complex model but ignores the issue of state uniqueness by operating under the assumption of a globally unique public state.

randomization but also determines and commits to the randomness used in its subsequent election (i.e., for state randomization). This mechanism ensures that, after each leader election, exactly one publicly verifiable state and approximately unique randomness are accepted by honest nodes, thereby enforcing uniqueness in leader succession. As a byproduct, the proposed randomization framework can be extended to other privacy-preserving applications.

- Mobius: an efficient Byzantine-resilient SSLE protocol. Building upon the *Approximately-Unique Randomization*, we present Mobius, a SSLE protocol that continuously outputs the desired leaders with a latency of 2Δ. Crucially, any attempt to deviate from this protocol would inherently reveal the malicious leader's identity, as its committed identity in the latest state is already publicly exposed. Moreover, we formally prove that Mobius is UC-secure without relying on extra trust assumptions. A comparison with prior approaches is presented in Table I.
- Implementation. We develop a proof-of-concept implementation of Mobius to evaluate its security and performance overhead. The results demonstrate that our design achieves both enhanced security and improved efficiency. In particular, compared to state-of-the-art shuffle-based schemes, Mobius reduces both the on-chain and off-chain message complexity from $O(n^2)$ to $O(n)$ during the registration phase. Experimentally, for a system initialized with 10 nodes, the on-chain communication cost is reduced by 81%. These efficiency gains scale favorably with system size due to Mobius's linear complexity advantage. Moreover, we also conduct an analysis between the shuffle-based scheme and the MPC-based framework, revealing that our shuffle-based scheme yields better performance under the same condition.

B. Technical Overview

Recall that in general shuffle-based SSLE protocols [15], [21], the uniqueness property of SSLE fundamentally depends on a unique state of the currently participating nodes. Therefore, our key idea to resist *state uniqueness* attack is to enforce that each leader can only generate one publicly verifiable state for use in the next election consecutively. Any deviation from this rule will risk compromising the leader's own security, as the next election will be conduced based on the state that has revealed its committed identity. We elaborate on technical contributions as detailed below.

- A committed randomness better prepared for the uniqueness property. To ensure the latest state can be randomized into a unique and publicly verifiable state, a critical step is to ensure both randomness and pre-determination (or immutability) of the value used in each randomization operation. For this purpose, the leader is required to not only generate a randomized state but also prepare a random value for its subsequent election, which will be consistently accepted or rejected by the honest nodes through a graded delivery protocol. Specifically, the commitment refresh mechanism mandates that each leader generates binding and hiding commitments for two random values (for node list re-randomization and permutation, respectively), while the graded delivery protocol ensures approximate consistency that contributes to the eventual uniqueness of randomization operations. The resulting architecture exhibits a tightly coupled design, wherein public verification of leader's current randomization is intrinsically linked to the uniqueness of its next randomization, providing both immediate security guarantees and forward-looking validation preparation.

- A publicly verifiable state ensuring uniqueness for each election. The newly introduced randomization architecture inspires us to integrate the prepared randomness into consecutive randomization operations, thereby establishing the uniqueness property of each election. Leveraging the hiding committed random values, the leader can utilize its two pre-determined random values to randomize its committed identity into the node list, ensuring the *unpredictability* of the next election. In addition, the binding committed random values impose a strict limitation that exactly one updated state can be generated and publicly verified. Importantly, malicious leaders attempting to propose multiple states or launch denial-of-service attacks will not compromise security. In such cases, the previously used state will be reused, thus only undermining the adversary's unpredictability (i.e., leaking its identity in an outdated node list).
- A pipelined paradigm guaranteeing high performance and compatibility. Building upon the strong guarantee of uniqueness property, the honest nodes can determine a single leader by relying on the same state along with the same position index. Our further efforts focus on performance optimization, particularly addressing the performance degradation caused by the delivery process. We develop a pipelined paradigm featuring two fundamental architectural advances: a *decoupling mechanism* for graded delivery that separates the state randomization from the dissemination of corresponding committed random values, and a *transient lock mechanism* for refreshing committed random values. This design enables the leader to concurrently complete the state randomization and temporarily lock the newly generated random values, which will remain locked until the leader's subsequent election. This pipelined design ensures our protocol delivers outputs within 2Δ, thereby achieving compatibility with nearly all real-world systems (e.g., Ethereum).

II. Preliminary

Notations. We denote by λ the security parameter. We write $x \leftarrow^{\$} \mathcal{X}$ to denote uniformly sampling a value x from a set $\mathcal{X}$ and $\{x_s\}_{s=0}^{s'}$ to denote a set of elements that $\{x_0, \cdots, x_{s'}\}$. For an integer n, we use $[n]$ to denote a set of natural numbers smaller than n, i.e., $\{0, 1, \cdots, n-1\}$, and S_n to denote the random permutation family, i.e., the set of all bijections $\eta : [n] \rightarrow [n]$. Assuming that $\mathbb{G}$ is a multiplicative group of prime order q with generator g, we denote by $\mathbf{h}$ a set of group elements $\{\mathrm{g}^{x_1}, \cdots, \mathrm{g}^{x_n}\}$, $\mathbf{h}^r$ represents the set $\{\mathrm{g}^{x_1 r}, \cdots, \mathrm{g}^{x_n r}\}$, and h_s denotes the s-th element in the set $\mathbf{h}$. Additionally, the single lowercase letters g, h represent the elements of the group.

A. Cryptographic Building Blocks

We briefly introduce the building blocks used in our protocol, while deferring the formal definitions to Appendix B.

Commitment scheme. A commitment scheme consists of two phases (Commit, Open). In the commit phase, a message m is committed as $com_m \leftarrow \mathsf{Commit}(m, r)$, that involves a secret value $r \leftarrow^{\$} \{0,1\}^{\lambda}$. In the open phase, the commitment com_m is opened by $\mathsf{Open}(com_m, r)$ and its correctness is verified by $\mathsf{Ver}(com_m, m)$. In this paper, we adopt the Pedersen commitment scheme [23], which is proven to be perfectly hiding and computationally binding [24].

Non-Interactive Zero-knowledge Proof (NIZK). An NIZK proof for an NP relation $\mathcal{R}$ is a tuple of PPT algorithms $(\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ such that: $\mathsf{NIZK.Setup}$ initializes the common reference string crs; for any $(x, \omega) \in \mathcal{R}$, the prover produces a proof $\pi \leftarrow \mathsf{NIZK.Prove}(crs, x, \omega)$; and for any statement x and proof π, the verifier outputs 1 via $\mathsf{NIZK.Verify}(crs, x, \pi)$, meaning that the proof π is accepted, otherwise outputs 0.

While requiring the NIZK to satisfy the properties of soundness and zero-knowledge [25], we consider the following relations in this work: $\mathcal{R}_{DH}$ refers to that a given tuple of group elements (g, h) is a Diffe-Hellman tuple, i.e. that there exists an x such that $h = \mathrm{g}^x$, and $\mathcal{R}_{sh}$ associates two vectors in $\mathbb{G}^n$ such that the second one (e.g., $\widetilde{\mathbf{h}}$) is obtained by re-randomizing and shuffling the first one (e.g., $\mathbf{h}$).

$$\mathcal{R}_{DH} = \{(\mathrm{g}, h), x) : \mathrm{g}, h \in \mathbb{G}, \mathrm{g}^x = h\}$$

$$\mathcal{R}_{sh} = \{((g, \mathbf{h}, \tilde{g}, \widetilde{\mathbf{h}}, (r, rs, \eta)) : r, rs \in \mathbb{F}_q, \eta \in S_n[rs], \tilde{g} = g^r, \widetilde{h}_i = h^r_{\eta(i)}\}$$

Random beacon. Analogous to the existing shuffle-based constructions [15], [21], we deploy the random beacon that produces publicly verifiable and unbiased random values as a building block. The recent advancements in threshold cryptography [26], [27] and verifiable delay functions [28], [29] have provided practical implementations [30], [31]. Rather than specifying a particular scheme, we abstract it as a black-box service, decoupling security analysis from our protocol.

Graded broadcast. A graded broadcast protocol [32] (a.k.a *gradecast*) enables a sender $\mathcal{P}$ to broadcast a message m among n recipients with a graded consistency guarantee. In this protocol, each recipient outputs a pair of (m, c), where c quantifies the local confidence grade regarding agreement on message m (cf. Section V-A for a specific construction).

Definition II.1. A gradecast protocol with a designated sender $\mathcal{P}$ is secure if the following properties hold:

- **Validity:** If the sender $\mathcal{P}$ is honest, every honest node $\mathcal{P}_i$ will output (m, c), where $c = 2$.
- **Graded Consistency:** If an honest node $\mathcal{P}_i$ outputs (m_i, c_i), then every honest node $\mathcal{P}_j$ outputs (m_j, c_j) such that $m_i = m_j$ and $|c_i - c_j| \leq 1$.
- **Termination:** Every honest node will terminate.

B. The Single Secret Leader Election

The SSLE protocol enables n registered nodes $\mathcal{P} := (\mathcal{P}_1, \cdots, \mathcal{P}_n)$ to collaboratively conduct a single leader election, while guaranteeing secrecy of the elected leader $\mathcal{P}_i$ from all other nodes until it actively reveals itself. We now informally review the syntax of SSLE as defined in [15].

Definition II.2 (Single Secret Leader Election (SSLE))**.** An SSLE scheme is defined as a tuple of PPT algorithms $\mathsf{SSLE} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{RegisterVerify}, \mathsf{Elect}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^{\lambda}, N) \rightarrow (pp, \{(sk_1, pk_1), \cdots, (sk_N, pk_N)\})$. The setup algorithm outputs public parameters pp, and the key

pair for each node in $\mathcal{P}$. It is a one-time setup process before initiating a series of elections.

- $\mathsf{Register}(pp, pk_i, \mathcal{L}') \rightarrow \mathcal{L}$. The registration algorithm enables node $\mathcal{P}_i \in \mathcal{P}$ with identity pk_i to be added into registered node list $\mathcal{L}$, where $\mathcal{L} = \mathcal{L}' \cup \mathcal{P}_i$.
- $\mathsf{RegisterVerify}(pp, pk_i, \mathcal{L}) \rightarrow 0/1$. The verification algorithm is conducted by the registered nodes to verify that node $\mathcal{P}_i$ has been registered correctly.
- $\mathsf{Elect}(pp, eid, sk_i, \mathcal{L}) \rightarrow (0, \perp)/(1, \pi_i)$. The election algorithm enables each registered node $\mathcal{P}_i$ with secret sk_i to locally determine whether it is elected, and outputs 1 along with a proof π_i for being elected and $(0, \perp)$ otherwise.
- $\mathsf{Verify}(pp, \mathcal{L}, pk_i, \pi_i) \rightarrow 0/1$. The verification algorithm enables a public verification of claimed leader $\mathcal{P}_i$'s correctness.

Correspondingly, SSLE satisfies the following security properties of uniqueness, unpredictability, and fairness:

- **Uniqueness:** One single leader per election is elected.
- **Unpredictability:** No participant should be able to guess the next leader better than at random.
- **Fairness:** For the n registered nodes, each of them is elected with the probability of $1/n$.

III. The State Uniqueness Attack – From Uniqueness Violation to Practical Attack

In this section, we present a comprehensive analysis of the state uniqueness attack, which can be leveraged to undermine the uniqueness guarantees of SSLE.

A. Attack Description

Given the prolonged execution of the distribution system (e.g., Ethereum), the continuous leader generation is imperative. In essence, within existing SSLE constructions [15], [18]–[21], [33], the unpredictability and uniqueness for each election rely entirely on a publicly verified randomized state (e.g., a randomized node list), however, the uniqueness of the state cannot be guaranteed. This naturally compels honest nodes to depend on distinct yet respectively verified states when electing the next leader. Based on the two most representative methodologies to guarantee unpredictability and uniqueness, we systematically analyze the attack methods as follows.

- **Public List With Encrypted Randomness (PL-ER) [15], [19], [20], [33].** The PL-ER paradigm utilizes a transparent state comprising both a public node list and a set of partial ciphertexts. These ciphertexts are specifically designed to construct an aggregated encrypted random identity index, which can only be decrypted by the corresponding node. In this framework, an adversarial node eligible to participate in the generation of randomized ciphertexts can launch a state uniqueness attack by selectively transmitting its partial ciphertext to specific honest nodes. Consider three nodes $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, all responsible for generating random ciphertexts while awaiting an election, where $\mathcal{P}_3$ is the adversarial node. During this process, $\mathcal{P}_3$ can strategically transmit its partial ciphertext only to $\mathcal{P}_1$ while deliberately excluding $\mathcal{P}_2$. This selective transmission leads $\mathcal{P}_1$ and $\mathcal{P}_2$ to aggregate distinct random indices, causing divergence in leader identity determination.
- **Blinded List with Public Randomness (BL-PR) [15], [18], [21].** In contrast to PL-ER, BL-PR utilizes a cryptographically blinded node list as the state, typically represented as commitments. The leader is elected based on publicly verified randomness that maps to a specific position in the node list. To ensure secure and continuous leader elections, the newly elected leader is required to randomize the node list using its own chosen random values. This randomization obscures the leader's own position (i.e., ensuring unpredictability for its future election) before distributing the state for subsequent elections. In the aforementioned three-party system, the elected adversary $\mathcal{P}_3$ can launch the state uniqueness attack as follows: $\mathcal{P}_3$ sends list $\mathcal{L}_1$ to $\mathcal{P}_1$ and a distinct randomized list $\mathcal{L}_2$ to $\mathcal{P}_2$, where both the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ are derived from $\mathcal{L}$ via different random values. As a result, both lists are publicly verifiable and separately guide honest nodes $\mathcal{P}_1$ and $\mathcal{P}_2$ to determine the next leader.

B. State Uniqueness Attack in Real-World Ethereum System

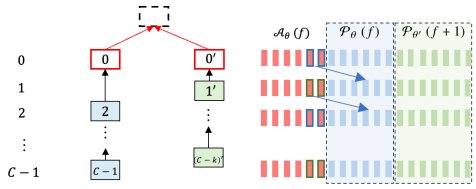
The Ethereum consensus protocol [22], named Gasper, is a committee-based Proof-of-Stake protocol that adopts a slot-epoch execution paradigm. Within this framework, an epoch is divided into C slots, and all n nodes (a.k.a. validators) are partitioned into C corresponding committees. For each epoch, Gasper allocates the committees to slots by subdividing a node list, and selects a single leader via sampling from a publicly known node distribution using a random value[2]. Subsequently, the protocol is executed through the collaborative efforts of the leader and committee validators, who respectively perform proposal and voting operations.

To enhance security and mitigate balance attacks [17], Ethereum combines single leader election [34] with a proposer boosting mechanism [16]. However, no effective method is provided to ensure a consistent node distribution across honest nodes, leaving the protocol vulnerable to state uniqueness attacks and thereby undermining the goal of selecting a single leader per slot. Even worse, this attack becomes particularly severe as the resulted leader conflicts among honest nodes can be exploited by the adversary to neutralize the proposer boosting mechanism, ultimately exacerbating the balance attacks.

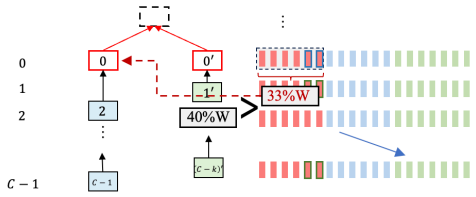
We now demonstrate the effectiveness of the state uniqueness attack against the Ethereum consensus protocol. Prior to delving into the detailed execution of our attack strategy, we first establish the necessary knowledge of the general *balance attacks* paradigm [17] and the *proposer boosting* mechanism [16] within the Ethereum consensus protocol.

Balance attack. As depicted in Fig. 2(a), the balance attack targets the Ethereum consensus layer to prevent block finalization, ultimately compromising the protocol's liveness. Specifically, the Byzantine adversary $\mathcal{A}_\theta$ of slot θ can control up to f validators, where $W = 3f + 1$ and $W = n/C$. At slot 0, when the adversary $\mathcal{A}_0$ is elected as block proposer (i.e.,

[2]cf. Appendix C-A for more details.



(a) Balance attack.



(b) Proposer boosting mechanism.

Fig. 2: Balance attack and proposer boosting mechanism.

leader), it first partitions the honest validators into two disjoint subsets $\mathcal{P}_0$ ($|\mathcal{P}_0| = f$) and $\mathcal{P}_{0'}$ ($|\mathcal{P}_{0'}| = f+1$), and prepares two conflicting blocks B_0 and $B_{0'}$ to launch the attack as follows:

- Slot 0: $\mathcal{A}_0$ transmits block B_0 to $\mathcal{P}_0$ and block $B_{0'}$ to $\mathcal{P}_{0'}$, respectively. Upon receiving these blocks, validators in $\mathcal{P}_0$ vote for B_0, while those in $\mathcal{P}_{0'}$ vote for $B_{0'}$.
- Slot 1: If $B_{0'}$ is extended by $B_{1'}$ in slot 1, then $\mathcal{A}_0$ discloses two previously withheld votes for B_0 to $\mathcal{P}_1$ ($|\mathcal{P}_1| = f$) before the voting phase. This disclosure ensures that $\mathcal{P}_1$ continue to support B_0, while $\mathcal{P}_{1'}$ ($|\mathcal{P}_{1'}| = f+1$) endorse $B_{1'}$. As a result, at the end of slot 1, the two competing chains maintain an approximately equal number of votes (i.e. $33.3\%W$).
- Slot 2 and beyond: Following slot 1, the adversary can persist in its vote-withholding behavior across subsequent slots, resulting in a sustained state of equilibrium between the two competing chains, where each chain collects approximately 33.3% of total validator votes (n) at the end of each epoch.

Consequently, neither chain can accumulate the required 66.7% total validator votes (n) to achieve block finalization, undermining the fundamental liveness property of the Ethereum consensus protocol.

Proposer boosting. To mitigate the balance attack, the proposer boosting mechanism is specifically designed to diminish the impact of adversarial votes on the chain-selection decisions rendered by honest validators. Specifically, this mechanism incentives the timely broadcast blocks by conferring a temporary 40% voting power advantage[3], thereby effectively countering adversarial withholding strategies. As depicted in Fig. 2(b), we consider a slot configuration with a total of $W = 3f+1$) votes. If block $B_{1'}$ is timely broadcast, it will receive an additional $40\%W$ votes. In this case, at slot 1, the honest nodes $\mathcal{P}_1$ cannot be induced to maintain their support for block B_0, as $\mathcal{A}_0$ only controls $33\%W$ votes in slot 0. Consequently, this mechanism robustly precludes the adversary from fragmenting the voting power of honest validators across competing chains, thereby achieving effective resistance against balance attacks.

Reactivation of balance attack. Nevertheless, when the prerequisite of having a single leader elected per slot cannot be guaranteed (i.e., in cases where a malicious leader initiates the state uniqueness attack), the protective effectiveness of the proposer boosting mechanism will be compromised, consequently reviving balance attacks. Building on this observation, the core attack strategy is outlined as follows (cf. Fig. 3).

Without loss of generality, we assume the first epoch and slot following the network reaching Global Stabilization Time[4] are epoch 0 and slot 0, respectively. The malicious leader initiates the attack by exploiting network propagation delays prior to epoch 0 to establish two competing chains in exact weight equilibrium by slot 0 (i.e., ■ in Fig. 3). Since different honest nodes support divergent chains embedded with distinct random values, the adversary can execute the selective message propagation, i.e., strategically withholding votes while selectively revealing them to the targeted honest nodes. This manipulation further misleads the honest leaders (i.e., L_0 and $L_{0'}$ in step ①, derived from the distinct random values r_{-2} and r'_{-2}) into adopting different canonical chains and extend them respectively (i.e., B_0 and $B_{0'}$). Consequently, the boosting weights of these conflicting blocks offset each other (i.e., $w(B_0) = w(B_{0'}) = 40\%W$), thereby empowering the adversary to reinitiate balance attacks and establish ideal conditions for state uniqueness attack to result in the sustainable coexistence of multiple leaders (step ③ in Fig. 3). More critically, the chain split can be maintained indefinitely, ultimately undermining the system's liveness guarantees (cf. Appendix C-B for a detailed description).

C. Quantitative Analysis of State Uniqueness Attack

To formally illustrate the effectiveness of the state uniqueness attack, we first simulate the attack to verify our theoretical analysis. Then, we implement it respectively in a vote-based consensus protocol and a heaviest-chain protocol to show its impact on the over-layer protocol, and perform an augmented evaluation of the attack on the real-world Ethereum system.

We conduct a simulation of the state uniqueness attack based on the state-of-the-art shuffle-based SSLE frameworks [15], [21]. Specifically, we perform this simulation under the following configurations: two fault tolerance thresholds ($f < n/2$ and $f < n/3$) that encompass mainstream scenarios, two prevalent message handling paradigms: the time-sensitive message handling mechanism[5] and lexicographical-order message handling mechanism[6], and two network configurations with $n = 5$ (with $f = 2$, satisfying $f < n/2$) and $n = 7$ (with

[3]40% is the boosting factor specified in Ethereum [35], which takes effect only in the current slot θ.

[4]Global Stabilization Time (GST) represents a critical time boundary in Ethereum's network model, demarcating a phase transition: the network is in a delay-unbounded and asynchronous regime before GST, while exhibiting synchronous properties thereafter.

[5]It operates under a "first-valid-message" acceptance rule, whereby parties accept the first valid message they receive.

[6]It deterministically sequences messages based on their hash values and selects either the maximum or minimum among them.

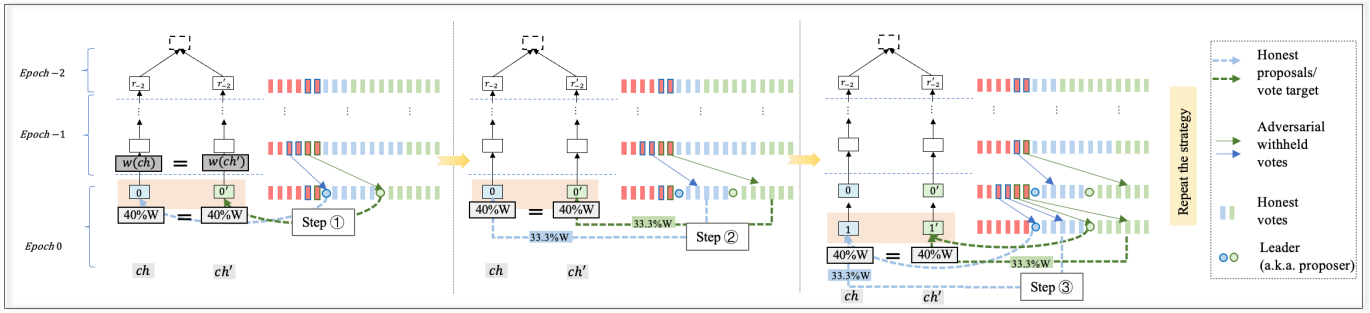
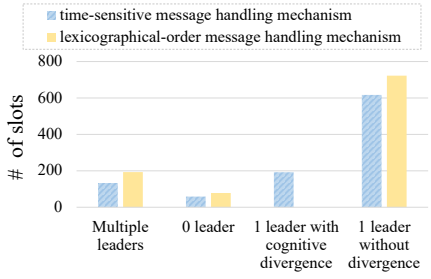
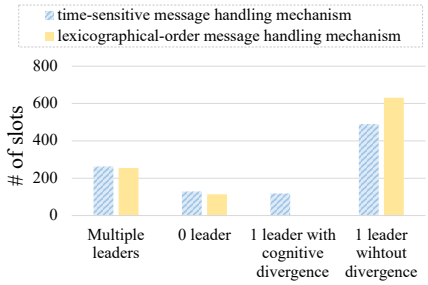


Fig. 3: The brief attack workflow caused by multiple leaders, wherein ch and ch' represent two competing chains that both are prevented from finalization. Additionally, the blocks in ch and ch' are proposed by leader L_θ and $L_{\theta'}$ and supported by two equal-sized node groups $\mathcal{P}_\theta$ and $\mathcal{P}_{\theta'}$, where θ represents the corresponding slot number.

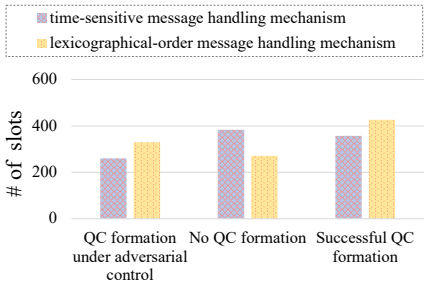


(a) SSLE with $n = 7, f = 2$ $(f < n/3)$

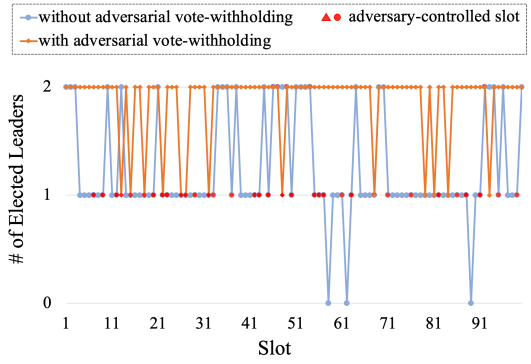


(b) SSLE with $n = 5, f = 2$ $(f < n/2)$

Fig. 4: The number of slots under varying conditions.



(a) BFT-like protocol.



(b) Gasper-like protocol.

Fig. 5: Evaluation of the over-layer protocol with $n = 7, f = 2$.

$f = 2$, satisfying $f < n/3$). For each configuration, We respectively run 1000 consecutive elections under the Byzantine adversary, where the adversary can arbitrarily deviate from the protocol specification, including propagating conflicting randomization messages and selectively withholding messages from subsets of honest nodes. Notably, our simulated results can be generalized to larger-scale systems, provided that the network conditions remain synchronous and there is adequate message processing capacity, as increase in committee size would only exacerbate the attack severity.

We present the results in Fig. 4 to clearly demonstrate that existing SSLE constructions exhibit significant vulnerabilities to state uniqueness attacks across all evaluated configurations. Specifically, with $f < n/3$, the probability of violating the uniqueness property reaches 13.3%-19.3% (i.e., 133-193 multi-leader slots, cf. Fig. 4(a)), which escalates to 25.5%-26.3% (i.e., 255-263 slots) when the condition is relaxed to $f < n/2$ (cf. Fig. 4(b)). Moreover, 12%-19% of slots exhibit divergent leader recognition, where only subsets of nodes acknowledge the elected leader, while 6%-13% of slots are identified as leaderless slots. Additionally, the time-sensitive message processing paradigm significantly exacerbates attack vulnerability, exhibiting success rates approximately 1.4 times higher than those observed with lexicographical ordering across both fault tolerance thresholds. This can be attributed to its inherent susceptibility to network latency manipulation and its adherence to the first-valid-message acceptance rule.

We further evaluate the attack's impact on the over-layer protocol by implementing it in a blockchain scenario. Specif-

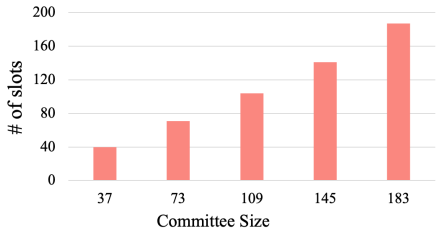


Fig. 6: Attack on the Ethereum consensus protocol.

ically, we respectively integrate this attack with a vote-based consensus protocol (e.g., BFT-like protocol PBFT [2]) and a heaviest-chain protocol (e.g., Ethereum Gasper [22]) under $f < n/3$ (cf. Fig. 5). For BFT-like protocols, we track quorum certificate (QC) formation as a key metric, which serves as an essential data structure in these protocols. The results (cf. Fig. 5(a)) show that in approximately 26%-30.3% of 1000 slots, the adversary is elected as the single leader, enabling it to manipulate QC formation and consequently compromise the over-layer protocol. Meanwhile, 35.7%-42.7% of slots under honest control successfully construct QCs. For the remaining slots, the adversary can exploit leadership divergence to stall protocol progress. Specifically, by partitioning honest parties into two subsets: $\mathcal{P}_1$ ($|\mathcal{P}_1| = f + 1$) and $\mathcal{P}_{1'}$ ($|\mathcal{P}_{1'}| = f$), the adversary can split honest votes between conflicting proposals, thereby preventing QC formation. In contrast, heaviest-chain protocols strictly enforce lexicographical ordering in their construction and recover to a secure state when an honest node becomes the sole leader. For this scenario, we examine the first 100 slots with $n = 7$. The results (cf. Fig. 5(b)) show that during the initial 100-slot period, chain equilibrium persists for 68 slots when the adversary employs a vote-withholding strategy. More critically, even without adversarial interference, these protocols can enter weight equilibrium states lasting for 14 slots in the worst case (i.e., slots 44 to 58 in Fig. 5(b)), which both lowers the attack barrier and introduces additional potential attack surfaces in protocol design.

We further proceed to evaluate the state uniqueness attack in a real-world Ethereum system (cf. Fig. 6). Specifically, we establish a local testnet based on Prysm [36] with the committee size of $n = 39$ to 183. The experimental results show that this attack remains sustainable for over 180 slots, even at an experimental scale of fewer than 200 nodes. Furthermore, as the number of nodes increased, the duration of the attack continued to grow. This indicates that, for the Ethereum mainnet where $n \approx 10^6$, the attack enables a prolonged chain divergence, significantly impairing the system's liveness and ultimately resulting in a denial-of-service (DoS) condition.

IV. System Model and Formal Security Definition

A. Modeling the System and Threats

We now discuss the security model of the generalized SSLE protocol, which exactly follows the previous works in this field [19], [21]. In order to formally model the security of SSLE, we adopt the Universal Composability (UC) [37] framework to define an SSLE model that involves N nodes $\mathcal{P} := \{\mathcal{P}_1, \cdots, \mathcal{P}_N\}$. The UC model specifies two worlds, a protocol Π is executed in the real world by the nodes that interacts with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, while a functionality $\mathcal{F}$ is executed in the ideal world and interacts with a simulator $\mathcal{S}$ and an environment $\mathcal{Z}$. We denote the ensemble corresponding to real world execution as $\text{EXEC}(\lambda)_{\Pi,\mathcal{Z},\mathcal{A}}$, while that for ideal world execution is represented as $\text{EXEC}(\lambda)_{\mathcal{F},\mathcal{Z},\mathcal{S}}$.

UC security. The protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\text{EXEC}(\lambda)_{\Pi,\mathcal{Z},\mathcal{A}} \approx \text{EXEC}(\lambda)_{\mathcal{F},\mathcal{Z},\mathcal{S}}$, where $\approx$ denotes the computational indistinguishability.

The known parties and trusted setup. There are N designed nodes (i.e., $\mathcal{P} := \{\mathcal{P}_1, \cdots, \mathcal{P}_N\}$), each of which has a unique public-private key pair (sk_i, pk_i) and a unique identity denoted by the public key pk_i known by everyone else. In addition, during protocol execution, there are n registered nodes $\mathcal{P}_n$, i.e., $\{\mathcal{P}_{i_1}, \cdots, \mathcal{P}_{i_n}\} \subseteq \{\mathcal{P}_1, \cdots, \mathcal{P}_N\}$ that take part in the election procedure, and the other nodes can dynamically register through a registration mechanism.

The adversary. The Byzantine adversary $\mathcal{A}$, which can fully control $f < n/2$ registered nodes, is able to arbitrarily deviate from the protocol specification, including sending conflicting messages and launching the denial-of-service attack.

The communication network. We consider the synchronous communication among the participating nodes, implying that we assume a global clock functionality $\mathcal{F}_{clock}$ [38]. The SSLE protocol executes in rounds (and slots), ensuring that each node is aware of the current round (and slot) and can expect that the message m sent by an honest node will be received by all the honest nodes within time Δ. There is also a secure authenticated message transmission channel between the nodes modeled by the ideal functionality $\mathcal{F}_{smt}$ [37], [38].

B. Ideal Functionality of SSLE

We formalize the security of SSLE as an ideal functionality $\mathcal{F}_{SSLE}$ (cf. Fig. 7), which interacts with the nodes in set $\mathcal{P}$, environment $\mathcal{Z}$, ideal adversary $\mathcal{S}$. Specifically, it consists of the following three procedures, each triggered by request messages with a session identifier eid from node $\mathcal{P}_i \in \mathcal{P}$:

- Register: The unregistered node $\mathcal{P}_i$ requests to participate in the election and is added to the registered node set $\mathcal{L}$.
- Elect: At each new election eid, one registered node in $\mathcal{L}$ is randomly selected to serve as the leader, and the election result is secretly sent to the designated node.
- Verify: When the elected node chooses to disclose its identity, the correctness of the claim can be verified publicly.

Security analysis. The ideal functionality satisfies that, for each election, there is exactly one node uniformly sampled from the set of registered nodes $\mathcal{L}$ (*uniqueness* and *fairness*). Each node is only allowed to access its own election result, unless others actively disclose their identities themselves (*unpredictability*), which can be publicly verified by others.

Initialize $E, \mathcal{L} \leftarrow \emptyset$, $n \leftarrow 0$.

Register: Upon receiving $(\mathsf{register}, pk_i) \hookleftarrow \mathcal{P}_i$:

- If $(pk_i, \cdot) \notin \mathcal{L}$, update $\mathcal{L} := \mathcal{L} \cup (pk_i, n)$, broadcast $(\mathsf{registered}, pk_i)$ and set $n := n + 1$.
- Otherwise, abort.

Elect: Upon receiving (elect, eid) from all honest nodes, if $\mathcal{L} \neq \emptyset$ and eid was not requested before, do the following:

- Sample $(pk_i, \delta) \leftarrow^{\$} \mathcal{L}$, where $\delta \in [n]$.
- Send $(\mathsf{outcome}, eid, 1) \hookrightarrow \mathcal{P}_i$ and $(\mathsf{outcome}, eid, 0) \hookrightarrow \mathcal{P}_{j \neq i} \in \mathcal{L}$, update $E := E \cup \{eid, pk_i, \cdot\}$.
- Upon receiving $(\mathsf{proof}, eid, pk_i, \pi_{i,eid}) \hookleftarrow \mathcal{P}_i$, if $(eid, pk_i, \cdot) \in E$, update $(eid, pk_i, \cdot) := (eid, pk_i, \pi_{i,eid})$. Otherwise, abort.

Verify: Upon receiving $(\mathsf{reveal}, eid, pk_i, \pi_{i,eid}) \hookleftarrow \mathcal{P}_i$, and $(\mathsf{verify}, eid, pk_i, \pi_{i,eid}) \hookleftarrow \mathcal{P}_{j \neq i} \in \mathcal{L}$

- If $(eid, pk_i, \pi_{i,eid}) \in E$, $(\mathsf{verified}, eid, pk_i, 1) \hookrightarrow \mathcal{P}_j$.
- Otherwise, $(\mathsf{verified}, eid, pk_i, 0) \hookrightarrow \mathcal{P}_j$.

Fig. 7: Ideal functionality that models SSLE.

V. Mobius: a Byzantine-resilient SSLE protocol with Enhanced Uniqueness

Solution intuition. As previously elaborated, the core challenge posed by the state uniqueness attack stems from the fact that a malicious node is able to generate distinct publicly verifiable states (e.g., the node lists in shuffled-based schemes) and strategically disseminate divergent versions to honest nodes, thereby compromising the uniqueness of SSLE.

Accordingly, our solution to this vulnerability centers on preventing the adversaries from accessing multiple valid random values per randomization operation. Specifically, only one valid state and random value can be utilized for each randomization operation (i.e., the re-randomization and shuffle), without leaking any information of the corresponding random factors. To achieve this, we introduce a method of "*Approximately-Unique Randomization*", compelling participating nodes to continuously pre-determine random values with approximate uniqueness, and integrate it into the *"Elect→Randomize"* workflow to provide perfect uniqueness guarantees. Consequently, each new state can be uniquely derived from the previous state using these unique and secret random values while enabling public verification of it.

A. Approximately Unique Randomization

We formally define *approximate uniqueness* as the properties of *gradecast* (cf. Definition II.1): the honestly pre-determined random values can be accepted by all the honest nodes with full confidence (i.e., $c = 2$), whereas the confidences of maliciously pre-determined random values held by honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ satisfy that $|c_i - c_j| \leq 1$. We now walk through how to realize the "*Approximately Unique Randomization*" (abbr. AU-Randomization).

Consecutively-verifiable randomization path. To enable the consecutive randomization operations, we first develop a commitment-based mechanism that generates and refreshes the random parameters (cf. Algorithm 1). With a node $\mathcal{P}$ as the shuffler to randomize the state, the key points are as follows:

- The shuffler $\mathcal{P}$ is required to commit two random values $r, rs \leftarrow^{\$} \mathbb{F}_q$ in advance, which is denoted as com_r and com_{rs}, along with an NIZK proof π_{com} for their correctness.
- $\mathcal{P}$ then re-randomizes the state tuple $(g, \mathbf{h})$ as $(g^r, \mathbf{h}^r)$, followed by performing a shuffle on $\mathbf{h}^r$ defined by $\widetilde{h}_\tau := h'_{\eta(\tau)}$, thereby yielding the updated state $(\widetilde{g}, \widetilde{\mathbf{h}})$, where $\eta \leftarrow S_n[rs]$.
- Finally, $\mathcal{P}$ computes an NIZK proof π_{sh} to demonstrate that the resulting state $(\widetilde{g}, \widetilde{\mathbf{h}})$ has been correctly re-randomized and shuffled from the original state $(g, \mathbf{h})$, using the committed random values r and rs. Additionally, $\mathcal{P}$ is also required to reconfigure the random values that are utilized for its subsequent randomization operation.

Algorithm 1 Consecutive Randomness Generation (of $\mathcal{P}$)

1: **function** RandGen($Com_1, g, \mathbf{h}$)
2: **Parse** $Com_1 =: \{com_r, com_{rs}\}$
3: $\tilde{g} \leftarrow g^r$, $\mathbf{h'} \leftarrow \mathbf{h}^r$
4: $\eta \leftarrow S_n[rs]$, $\widetilde{\mathbf{h}} \leftarrow \{\tilde{h}_\tau := h'_{\eta(\tau)} | \tau \in [n]\}$
5: $\pi_{sh} \leftarrow \mathsf{NIZK.Prove}(Com_1, g, \tilde{g}, \mathbf{h}, \widetilde{\mathbf{h}}; (r, rs, \eta))$
6: $r', rs' \leftarrow^{\$} \mathbb{F}_q$
7: $\overline{com_{r'}} \leftarrow \mathsf{Commit}(r')$, $com_{rs'} \leftarrow \mathsf{Commit}(rs')$
8: $\overline{Com} \leftarrow (com_{r'}, com_{rs'})$
9: $\pi_{\overline{com}} \leftarrow \mathsf{NIZK.Prove}(\overline{Com}; (r', rs'))$
10: **return** $m_{sh} := (\mathsf{shuffle}, \tilde{g}, \widetilde{\mathbf{h}}, \pi_{sh}, \overline{Com}, \pi_{\overline{com}})$

Approximately-unique randomization path. Following the consecutive reconfiguration of randomness, we develop a gradecast protocol, namely GradeDelivery, as a building block to deliver the generated randomness. By this protocol, the node can output a message m with an agreement confidence c, which serves as links across the complete consecutive randomization process. Informally, the protocol consists of four steps described as follows (cf. Algorithm 2):

- Message approval (line 2-9). The function initializes a timer $t = 0$ when called, and then increments t monotonically during the execution. The sender $\mathcal{P}_i$, hereafter a shuffler, firstly broadcasts the shuffle message m_i to the nodes in $\mathcal{P}_n$. At $t = \Delta$, if $\mathcal{P}_j$ receives the shuffle message m_i, it invokes $\mathsf{ValidShuffle}(m_i, Com_{i,1})$, where $Com_{i,1}$ represents the latest commitment tuple generated by $\mathcal{P}_i$ and recorded in $\mathcal{P}_j$'s local storage. Upon receiving True and confirming that no conflicting shuffle message m'_i has been received, it signs m_i and sends the signature σ_j back to $\mathcal{P}_i$ (line 9).
- Certificate aggregation (line 10-15). At $t = 2\Delta$, if the shuffler $\mathcal{P}_i$ has received more than $n/2 + 1$ valid signatures, it aggregates these signatures into a certificate denoted as Ω_i and subsequently broadcasts Ω_i to the nodes in $\mathcal{P}_n$.
- Revocation (line 16-19). At $t = 3\Delta$, if $\mathcal{P}_j$, who has previously signed m_i ($\phi_{j,i} = 1$), does not receive Ω_i from $\mathcal{P}_i$, it then broadcasts a *revoke* message to invalidate σ_j.
- Update(line 21-28). At $t = 4\Delta$, if $\mathcal{P}_j$, who has previously signed m_i, receives a certificate Ω_i with $\geq n/2+1$ valid sig-

natures (line 24), and does not receive any conflicting shuffle messages, $\mathcal{P}_j$ then sets $c_i \leftarrow 2$. What's more, if it receives less than $n/2+1$ valid signatures but $|\Omega_i| \geq n/2+1$ (line 26), then it sets $c_i \leftarrow 1$. In this case, $\mathcal{P}_j$ outputs (m_i, c_i). Otherwise, if $\mathcal{P}_j$ receives no certificate or two conflicting messages, it outputs $(m_i, 0)$ or $(\perp, 0)$ respectively.

Algorithm 2 Graded Broadcast (of $\mathcal{P}_i \in \mathcal{P}_n$)

1: **function** GRADEDDELIVERY(m_i)
2: **Set** timer $t \leftarrow 0$, $c_i \leftarrow 0$
3: **if** $\mathcal{P}_i$ is the sender: Broadcast m_i
▷ Message Approval
4: **For** each $\mathcal{P}_j \in \mathcal{P}_n$, at $t \in (0, \Delta]$
5: **Upon** receiving a shuffle message m_i from $\mathcal{P}_i$
6: **if** VALIDSHUFFLE($m_i, Com_{i,1}$) = True **then**
7: Compute $\sigma_j \leftarrow \mathsf{Sign}(sk_j, m_i)$ and set $\phi_{j,i} = 1$
8: **if** no m_i' has been received at $t = \Delta$ **then**
9: send σ_j to $\mathcal{P}_i$
▷ Certificate Aggregation
10: **For** shuffler $\mathcal{P}_i$, at $t \in [\Delta, 2\Delta]$
11: **Upon** receiving σ_j from $\mathcal{P}_j$
12: **if** $\mathsf{Verify}(pk_j, \sigma_j, m_i) = 1$ **then**
13: Update $\Omega_i := \Omega_i \cup \sigma_j$
14: **At** $t = 2\Delta$, **if** $|\Omega_i| \geq n/2+1$ **then**
15: Broadcast Ω_i
▷ Revocation
16: **For** each $\mathcal{P}_j \in \mathcal{P}_n$, at $t = 3\Delta$
17: **if** $(\phi_{j,i} = 1) \wedge$ (no Ω_i has been received $\vee$ $|\Omega_i| < n/2+1$)
18: Broadcast $(\mathsf{revoke}, \sigma_j)$
19: **else**: Forward Ω_i
20: **For** each $\mathcal{P}_j \in \mathcal{P}_n$, at $t \in [3\Delta, 4\Delta]$
21: **Upon** receiving $(\mathsf{revoke}, \sigma_k)$ from $\mathcal{P}_k \in \mathcal{P}_n$
22: Update $\overline{\Omega}_i := \overline{\Omega}_i \cup \sigma_k$
▷ Update
23: **At** $t = 4\Delta$ and no conflicting m_i' received
24: **if** $(|\Omega_i / \overline{\Omega}_i| \geq n/2+1)$:
25: $c_i \leftarrow 2$
26: **else if** $(|\Omega_i / \overline{\Omega}_i| < n/2+1) \wedge (|\Omega_i| \geq n/2+1)$:
27: $c_i \leftarrow 1$
28: **return** (m_i, c_i)

function VALIDSHUFFLE($m_i, Com_{i,1}$)
Parse $m_i =: (\mathsf{shuffle}, \tilde{g}, \tilde{\mathbf{h}}, \pi_{i,sh}, \overline{Com}_i, \pi_{\overline{com}})$
▷ $(g, \mathbf{h})$ represents the latest state stored by $\mathcal{P}_j$
$b_{i,1} \leftarrow$ NIZK.VERIFY($Com_{i,1}, g, \tilde{g}, \mathbf{h}, \tilde{\mathbf{h}}, \pi_{i,sh}$)
$b_{i,2} \leftarrow$ NIZK.VERIFY($\overline{Com}_i, \pi_{\overline{com}}$)
if $(b_{i,1} = 1) \wedge (b_{i,2} = 1)$ **then**:
return True
else: return False

With the rigorous construction, the protocol satisfies the properties defined in Definition II.1. We defer the detailed proof to Appendix E in the full version [39] and provide the security intuition below.

Security intuition. The execution can naturally ensure the validity and termination properties of the protocol, thereby providing security against the denial-of-service attack. Regarding graded consistency, the grade $c = 2$ for message m is achieved upon collecting $n/2$ signatures, guaranteeing that all honest nodes will output $c \geq 1$ for m. During this process, if a malicious sender disseminates two conflicting messages to different honest nodes to obtain $c = 2$, all the honest nodes will detect the conflicts and refuse to approve the conflicting messages, resulting in no message being able to achieve $c = 2$.

Approximately-Unique Randomization. Building on the randomness refreshing and the gradecast, the nodes in $\mathcal{P}_n$ can achieve an approximate uniqueness by Algorithm 3 as follows.

- Randomness Generation: The designated shuffler $\mathcal{P}_i$ executes the randomness generation procedure (Algorithm 1) to output shuffle message m_i.
- Randomness Gradecast: The shuffler $\mathcal{P}_i$ gradecasts the message m_i while the nodes in $\mathcal{P}_n$ acknowledge the message m_i (Algorithm 2) and deliver it with a confidence tag.

Algorithm 3 AU-Randomization (of $\mathcal{P}_i$)

1: **function** AU-RANDOMIZE($Com_{i,1}, g, \mathbf{h}$)
2: $m_i \leftarrow$ RANDGEN($Com_{i,1}, g, \mathbf{h}$)
3: GRADEDDELIVERY(m_i)

So far, an approximately unique and publicly verifiable randomization (i.e., $(\tilde{g}, \tilde{\mathbf{h}})$ and $\overline{Com}$) can be realized among the participating nodes in $\mathcal{P}_n$,

Security intuition. With a pre-committed randomness commitment $Com_{i,1}$, the state can only be appropriately randomized through a designated yet secret way, while the refreshed commitments that are transmitted by the graded delivery naturally satisfy the property of graded consistency defined in Definition II.1. We defer the detailed proof to Appendix E in our online full version [39].

B. Mobius: Protocol Description

We can now securely realize the SSLE via simply executing the *AU-Randomization*. Informally, by integrating the graded output into the query-driven workflow, the honest nodes can dynamically refresh their confidence parameters and determine whether to update the randomization message, including the randomized state and refreshed commitment, to their local storage. Subsequently, through applying the newly tossed coin γ on the local newest node list $\mathbf{h}$, the honest nodes can determine a single leader while ensuring unpredictability. However, the inherent latency of 4Δ associated with *AU-Randomization* still renders the trivial construction impractical for deployment (cf. Appendix D for details). Therefore, our further efforts concentrate on decoupling the update of the randomization message m_i into the node list $\mathbf{h}$ and committed random values. This approach ensures that the newly updated $\mathbf{h}$ can be utilized for subsequent election, while the committed randomnesses can be employed in $\mathcal{P}_i$'s next operation.

Protocol details. For ease of understanding, we depict Mobius in Fig. 8 and provide a detailed description in Algorithm 4, summarizing the key points of each phase as follows:

(A1) Registration Phase: Setup. In this phase, the initial n nodes in set $\mathcal{P}_N$ jointly generate an initial registered nodes set, denoted as $\mathcal{P}_n$ which includes their respective public keys pk and the committed random values utilized in subsequent randomization operations. With the established nodes set $\mathcal{P}_n$, these n nodes deterministically serialize $\mathcal{P}_n$ in accordance

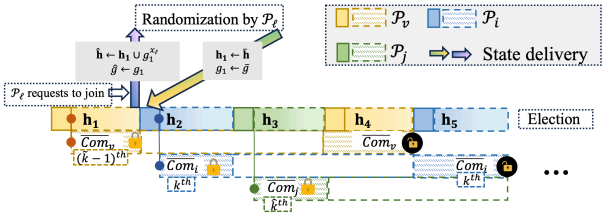


Fig. 8: Overview of the Mobius protocol, which consists of the registration and election phase. For conciseness, we abbreviate the state tuple $(g_\theta, \mathbf{h}_\theta)$ as $\mathbf{h}_\theta$ for $\theta \in \mathrm{N}^+$ to represent the randomization states, while $(\hat{g}, \hat{\mathbf{h}})$ and $(\overline{g}, \overline{\mathbf{h}})$ denote the randomization states that are input or output by a randomization process in registration phase.

with a strict lexicographic ordering and elect the first $\lfloor \frac{n}{2} \rfloor + 1$ nodes to sequentially complete their respective randomization operation At the end, they generate the initial state $(g, \mathbf{h})$, which is randomized and becomes the basis of the election.

(A2) Registration Phase: Joining Request. This phase facilitates the secure integration of the unregistered node, denoted as $\mathcal{P}_\ell$, into the protocol execution. Specifically, the new node $\mathcal{P}_\ell$ first executes step (**A1**)-01 to join the node set $\mathcal{P}_n$ maintained by all registered nodes. Subsequently, $\mathcal{P}_\ell$ incorporates itself into the most recent joining state $(\hat{g}, \hat{\mathbf{h}})$ and completes its randomization operation, specifically re-randomizing and shuffling, to randomize $(\hat{g}, \hat{\mathbf{h}})$. Once this updated joining state $(\hat{g}, \hat{\mathbf{h}})$ is accepted by a leader, that leader will confirm the randomization operation that generates the joining state and update the state to become the latest state $(g, \mathbf{h})$. Notably, several unregistered nodes can collaboratively generate the updated joining state and successfully participate in protocol execution once the state is adopted by a leader.

(B) Election Phase. This phase starts when $n \geq 3$. At each slot $\theta \geq 1$, based on the response γ_θ from $\mathcal{F}_{ct}^n$, the node owning the γ_θ-th position of the latest node list $\mathbf{h}$ is designated as the leader for slot θ. To enhance the protocol efficiency, Mobius introduces a transient lock mechanism (cf. Fig. 8) that strategically decouples the k-th randomization message $m_{j,k}$ of leader $\mathcal{P}_j$ into the state $(g, \mathbf{h})$ and the commitment $\overline{Com_j}$ of the refreshed random values. Therefore, the valid updated state $(g, \mathbf{h})$ can be used for the leader election of slot $\theta + 1$ as soon as possible, while ensuring the refreshed random values to be updated in the $(k+1)$-th election of $\mathcal{P}_j$. Moreover, to mitigate potential malicious behaviors by leaders, Mobius further combines the updates of state $(g, \mathbf{h}) \in m_{j,k}$ and commitment $\overline{Com_j} \in m_{j,k-1}$. The state will only be updated according to $\mathcal{P}_j$ if both the state $(g, \mathbf{h}) \in m_{j,k}$ is valid and the grade output by GradeDelivery$(m_{j,k-1})$ is $c = 2$. Otherwise, the current valid state will be maintained.

Security intuitions. Mobius efficiently and securely realizes the functionality defined in Fig. 7. We now provide the brief security intuitions and defer the detailed proofs to Appendix F in the full version [39].

Honest leader $\mathcal{P}_j$. The honest nodes will successfully update local state according to the randomization message $m_{j,k}$ proposed by leader $\mathcal{P}_j$. Therefore, all the honest nodes will determine the next leader as the owner of the γ-th position within the same state $\mathbf{h}$, thereby ensuring *uniqueness* and *fairness*. Given that leader $\mathcal{P}_j$ has re-randomized itself in state $\mathbf{h}$, this guarantees *unpredictability* of its $(k+1)$-th election.

Malicious leader $\mathcal{P}_j$. If leader $\mathcal{P}_j$ initiates the state uniqueness attack, Mobuis will protect the honest nodes by allowing them to maintain their current local state $(g, \mathbf{h})$ rather than adopting the state generated by $\mathcal{P}_j$. As a result, the security properties of *uniqueness* and *fairness* are guaranteed, while the *unpredictability* can only be partially guaranteed, given that its position in state $\mathbf{h}$ has been revealed.

Overall, in each leader election, if at least one honest node adopts the state updated by leader $\mathcal{P}_j$, all honest nodes will update their local states consistently. In situations where any honest node detects dishonesty from leader $\mathcal{P}_j$, all honest nodes will abort their updates related to $\mathcal{P}_j$, thereby compromising the security of $\mathcal{P}_j$ in its subsequent election. Consequently, Mobius enhances security against state uniqueness attack.

A concrete instance of decoupling and lock. Here, we present a concrete example to elaborate on the aforementioned *update-decoupling* and *transient lock* of node $\mathcal{P}_i$, whose instances are represented as ▮ and ▨ in Fig. 8.

At slot θ', we assume this is the k-th election of node $\mathcal{P}_i$. Initially, the nodes execute the first two steps of GradeDelivery to confirm the update of state $\mathbf{h}_1 \in m_{i,k}$, unlock $\overline{Com_i} \in m_{i,k-1}$, and generate the lock for $\overline{Com_i} \in m_{i,k}$. At slot θ, node $\mathcal{P}_i$ commences its $(k+1)$-th election. Specifically, the nodes perform GradeDelivery to generate the lock for $\overline{Com_i} \in m_{i,k+1}$ while simultaneously confirming both the validation of state $\mathbf{h}_4 \in m_{i,k+1}$ (i.e., Condition 1) and unlocking $\overline{Com_i} \in m_{i,k}$ (i.e., Condition 2). Only when both conditions are confirmed by the honest nodes (i.e., $c = 2$), will state $\mathbf{h}_4$ be utilized in the next election, and the stored committed random values Com_i of node $\mathcal{P}_i$ will be updated to $\overline{Com_i} \in m_{i,k}$.

Therefore, the proposed *update-decoupling* and *transient lock* mechanisms facilitate an efficient pipelined execution while ensuring both security and compatibility with existing protocols, where 1 slot =2Δ.

VI. Evaluation

In this section, we implement Mobius and compare it against the state-of-the-art shuffle-based SSLE schemes [15], [21] and the MPC-based scheme [20] across two key dimensions, i.e., Byzantine-resilience and communication cost. All tests are conducted on a Ubuntu 22.04 system in a Thinkpad X1 Carbon laptop, and the details of the evaluations are as follows.

Attack simulation. We conduct a simulation of various Byzantine attack strategies to evaluate the security of Mobius, with particular emphasis on *uniqueness*. In the simulation, we set $n = 5$ and $f = 2$, executing the election process for 1000 slots. As illustrated in Table II, Mobius exhibits an enhanced guarantee of *uniqueness* compared to the related works [15], [21], respectively under both the time-sensitive and lexicographical-order message handling mechanisms. Specifically, under the same system scale and Byzantine fault

Suppose each participating node $\mathcal{P}_i$ maintains a table $\mathrm{T}_i := \{\mathcal{P}_n, (g, \mathbf{h})\}$, which records the set $\mathcal{P}_n$ of registered nodes' identities (i.e., the public keys) and the corresponding commitments, and the latest state $(g, \mathbf{h})$. Global parameters are $(\mathbb{G}, q, \mathrm{g})$, $\mathcal{P}_n \leftarrow \emptyset, g \leftarrow \mathrm{g}, \mathbf{h} \leftarrow \emptyset, N, n \leftarrow 0$, and confidence grade $c \leftarrow 0$.

(A1) Registration Phase: Setup

01 Node $\mathcal{P}_i \in \mathcal{P}_N$ completes the Setup:

1) Sample $x_i, \{r_{i,s}, rs_{i,s}\}_{s=0}^{2} \xleftarrow{\$} \mathbb{F}_q$, computes $pk_i := \mathrm{g}^{x_i}$, $\{Com_{i,s} := (com_{r_s} := \mathrm{g}^{r_{i,s}}, com_{rs_s} := \mathrm{g}^{rs_{i,s}})\}_{s=0}^{2}$.
2) Set $\mathcal{P}_n := \mathcal{P}_n \cup \{(pk_i, Com_i)\}$, where $Com_i := \{Com_{i,1}, Com_{i,2}\}$.
3) Send $\big(\mathsf{prove}, \mathrm{g}, pk_i, \{Com_{i,s}\}_{s=0}^{2}; (x_i, \{r_{i,s}, rs_{i,s}\}_{s=0}^{2})\big) \hookrightarrow \mathcal{F}_{NIZK}^{DH}$.

Upon receiving $\big(\mathsf{proof}, \mathrm{g}, pk_j, \{Com_{j,s}\}_{s=0}^{2}; \pi_j\big) \hookleftarrow \mathcal{F}_{NIZK}^{DH}$, update $\mathcal{P}_n := \mathcal{P}_n \cup (pk_j, Com_j)$, $n \leftarrow |\mathcal{P}_n|$.

02 Node $\mathcal{P}_i \in \mathcal{P}_n$ completes the Randomization:

1) Serialize the registered node list $\mathcal{P}_n$ deterministically as $\mathcal{P}_n := \{(pk_{i_\kappa}, Com_{i_\kappa})\}_{i_{\kappa=1}}^{n}$, where $i_1 < i_2 < \cdots < i_n$.
2) Set $g \leftarrow \mathrm{g}$, $\mathbf{h} \leftarrow \{pk_{i_\kappa}\}_{\kappa=1}^{n}$.
3) The first $\lfloor \frac{n}{2} \rfloor + 1$ nodes $\mathcal{P}_i \in \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_{\lfloor \frac{n}{2} \rfloor + 1}}\}$ sequentially do the following:
   - Compute $\tilde{g} := g^{r_{i,0}}$, $\widetilde{\mathbf{h}} := \mathbf{h}^{r_{i,0}}$, $\eta \leftarrow S_n[rs_{i,0}]$, and $\widetilde{\mathbf{h}} := \{\tilde{h}_\tau := \tilde{h}_{\eta(\tau)} | \tau \in [n]\}$
   - Send $\big(\mathsf{prove}, g, \mathbf{h}, \tilde{g}, \widetilde{\mathbf{h}}, Com_{i,0}; (r_{i,0}, rs_{i,0}, \eta)\big) \hookrightarrow \mathcal{F}_{NIZK}^{sh}$.

   Upon receiving $\big(\mathsf{proof}, g, \mathbf{h}, \tilde{g}, \widetilde{\mathbf{h}}, Com_{i,0}; \pi_i\big) \hookleftarrow \mathcal{F}_{NIZK}^{sh}$, update $g \leftarrow \tilde{g}$ and $\mathbf{h} \leftarrow \widetilde{\mathbf{h}}$.

(A2) Registration Phase: Joining Request

01 Upon receiving a state $(g, \mathbf{h})$ from the latest leader and joining request $(\hat{g}, \hat{\mathbf{h}})$, the joining node $\mathcal{P}_\ell \in \mathcal{P}_N \wedge \mathcal{P}_i \notin \mathcal{P}_n$ does the following:

1) Executes step (**A1**)-01 to complete the Setup.
2) If $|\hat{\mathbf{h}}| \leq |\mathbf{h}|$, then set $(\hat{g}, \hat{\mathbf{h}}) := (g, \mathbf{h})$; otherwise, set $(\hat{g}, \hat{\mathbf{h}}) := (\hat{g}, \hat{\mathbf{h}})$.
3) Compute $\bar{g} = \hat{g}^{r_{\ell,0}}$, $\bar{\mathbf{h}} := \hat{\mathbf{h}}^{r_{\ell,0}} \cup \bar{g}^{x_\ell}$, $\eta \leftarrow S_n[rs_{\ell,0}]$, $\bar{\mathbf{h}} := \{\bar{h}_\tau := \bar{h}_{\eta(\tau)} | \tau \in [n]\}$.
4) Send $\big(\mathsf{prove}, \mathrm{g}, pk_\ell, \{Com_{\ell,s}\}_{s=0}^{2}; (x_\ell, \{r_{\ell,s}, rs_{\ell,s}\}_{s=0}^{2})\big) \hookrightarrow \mathcal{F}_{NIZK}^{DH}$, and $\big(\mathsf{prove}, \hat{g}, \hat{\mathbf{h}}, \bar{g}, \bar{\mathbf{h}}, Com_{\ell,0}; (r_{\ell,0}, rs_{\ell,0}, \eta)\big) \hookrightarrow \mathcal{F}_{NIZK}^{sh}$.

02 Upon receiving $\big(\mathsf{proof}, \mathrm{g}, pk_\ell, \{Com_{\ell,s}\}_{s=0}^{2}; \pi_\ell\big) \hookleftarrow \mathcal{F}_{NIZK}^{DH}$, and $\big(\mathsf{proof}, \hat{g}, \hat{\mathbf{h}}, \bar{g}, \bar{\mathbf{h}}, Com_{\ell,0}; \pi_\ell\big)\big) \hookleftarrow \mathcal{F}_{NIZK}^{sh}$, node $\mathcal{P}_i \in \mathcal{P}_n$ updates $\mathcal{P}_n := \mathcal{P}_n \cup \{(pk_\ell, Com_\ell)\}$, and $\hat{g} := \bar{g}$, $\hat{\mathbf{h}} := \bar{\mathbf{h}}$.

(B) Election Phase

01 At shot $\theta \geq 1$, node $\mathcal{P}_i \in \mathcal{P}_n$ does the following:

1) Invoke $(\mathsf{toss}, \theta) \hookrightarrow \mathcal{F}_{ct}^{n}$ if $\theta = 1$ or confidence grade $c_i = 2$.
2) Upon receiving $(\mathsf{tossed}, \theta, \gamma) \hookleftarrow \mathcal{F}_{ct}^{n}$, if $\gamma = \perp$, then set $\gamma_\theta := \gamma_{\theta-1} + 1 \bmod n$; otherwise, update $\gamma_\theta := \gamma$, $c_i \leftarrow 2$.
3) Upon the latest state $(g, \mathbf{h})$ and joining request $(\hat{g}, \hat{\mathbf{h}})$, if $\mathcal{P}_j$ is owner of the γ_θ-th position in $\mathbf{h}$:
   - Broadcast its k-th leadership claim $(\mathsf{claim}, j, \theta, k, g, \mathbf{h}, \pi_j)$, where $\pi \leftarrow \mathsf{NIZK.Prove}(\mathrm{g}, g, pk_j, h_{\gamma_\theta}; x_j)$.
   - If $\exists |\hat{\mathbf{h}}| > |\mathbf{h}|$, then set $(g, \mathbf{h}) := (\hat{g}, \hat{\mathbf{h}})$; otherwise, skip this step.
   - Run AU-RANDOMIZE$(Com_{j,1}, g, \mathbf{h})$ (cf. Algorithm V-A).
4) Upon receiving the shuffle message $m_{j,k}$, node $\mathcal{P}_i \in \mathcal{P}_n$ executes GradeDelivery process (Algorithm 2) to respectively respond to leader $\mathcal{P}_j$'s k-th shuffle message $m_{j,k}$ and the last two steps of $(k-1)$-th shuffle message $m_{j,k-1}$:
   - If VALIDSHUFFLE$(m_{j,k}, Com_{j,1})$ = True, then set confidence grade $c_i = 2$ and forward $m_{j,k}$; otherwise, set confidence grade $c_i = 1$ and skip to step **(B)**-02.
   - Update the confidence grade c_i as the output of GradeDelivery if $k > 1$.

02 Node $\mathcal{P}_i \in \mathcal{P}_n$ returns to step **(B)**-01 to complete the first two steps:

1) If $k = 1 \wedge c_i = 2$, then update $Com_j := (Com_{j,1} := Com_{j,2}, \perp)$.
2) If $k > 1 \wedge c_i = 2$, then update $(g, \mathbf{h}) := (m_{j,k}.g, m_{j,k}.\mathbf{h})$ and $Com_j := (Com_{j,1} := m_{j,k-1}.\overline{Com}_j, \perp)$
3) Return to the third step of **(B)**-01.

Algorithm 4: The Mobius protocol.

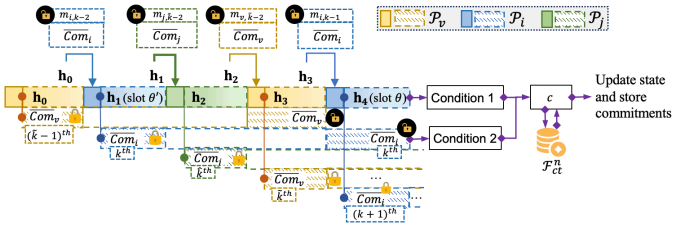


Fig. 8: The overview of the update-decouple with a transient lock, which enables a pipelined execution by decoupling the state delivery (e.g., $(g_1, \mathbf{h}_1)$) from the commitment delivery (e.g., the k^{th} commitment $\overline{Com}_i$ included in $m_{i,k}$). The boxes ■ and ▨ represent the state update and commitment delivery respectively. The arrow notation represents the relationship between data, where ⊷ indicates the data are included in the same message, and ↔ highlights that the delivery of the data will influence (or be influenced by) the confidence parameter.

TABLE II: The number of slots under different conditions.

Constructions	Multiple leaders	0 leader	1 leader with leadership divergence	1 leader without divergence+
time -sensitive*	263	129	118	490
lexicographical -order*	255	114	0	631
Mobius	0	0	0	1000

* The "time-sensitive" and "lexicographical-order" respectively represent the messages mechanisms deployed in prior constructions [15], [21].
+ The results include that the adversary is elected as the unique leader.

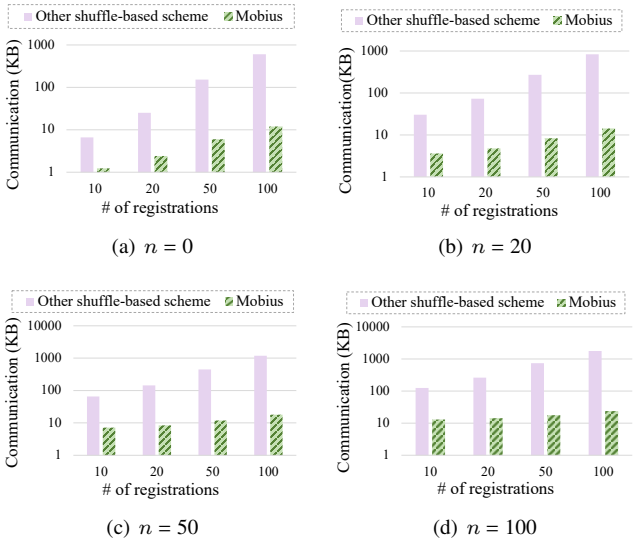


Fig. 9: On-chain communication cost in the registration phase.

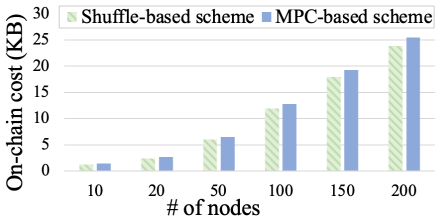


Fig. 10: Comparison with MPC-based schemes.

tolerance threshold, the works [15], [21] suffer from the leader identity divergence in 25.5%-38.2% of 1000 executions.In stark contrast, when subjected to uniqueness attack strategies in Section III-C, Mobius benefits from its *AU-Randomization* and the integration with the inherent workflow, effectively mitigating such divergence attack, achieving perfect *uniqueness* (i.e., 0% divergence rate). This empirical validation underscores Mobius' potential as a significant enhancement for distributed systems that expect leader uniqueness.

Communication cost. For the equitable comparison, we apply Mobius in blockchains and evaluate both off-chain and on-chain costs across the protocols (cf. Table I and Fig. 9).

In terms of off-chain communication cost, each node in the evaluated protocols, including Mobius and prior construction, sustains an inherent $O(n^2)$ communication overhead, necessitated by the broadcast of n-element nodes list. For $n = 2^{10}$, this translates to 126MB of off-chain data transmission.

As for on-chain cost, our analysis focuses on two phases, i.e., the election phase and registration phase. During the election phase, our protocol inherently ensures an $O(n)$ on-chain communication complexity by posting a node list as a leadership proof, which is in line with prior shuffle-based schemes [15], [21]. In the registration phase, assuming n_r registrations occur between two elections, we measure the communication overheads that is induced by the continuous registrations across different system scales. Introducing a novel shuffle mechanism, Mobius reduces the communication complexity from $O(n^2)$ to $O(n)$ by eliminating historical data storage requirements. Furthermore, when comparing with prior shuffle-based schemes, the systematic evaluations across varying registration request scales (n_r) and system sizes (n) further demonstrate Mobius' superior efficiency (cf. Fig. 9). Specifically, for an initial system ($n = 0$), the communication costs of Mobius decrease by 81.4%, 90.4%, 96% and 98% when $n_r = 10, 20, 50$, and 100 respectively. With $n_r = 10$ and varying n ($n = 0, 20, 50, 100$), the on-chain costs are reduced by 81.4%, 88.1%, 89.1% and 89.6% respectively. These results further indicate that our protocol is more efficient and lightweight than prior protocols, making it better suited for deployment as a functional module in a real-world system.

We also compare the communication cost of the shuffle-based scheme with the MPC-based scheme [20] (cf. Fig. 10). In the MPC approach, the node designated as leader must complete $O(n)$ MPC operations, whereas Mobius only requires it to send at most $O(n^2)$ messages. To support external public verifiability, the MPC scheme needs to store intermediate states generated during the protocol on-chain, while Mobius only needs to maintain a single list. Our measurements further show that the shuffle-based scheme reduces on-chain overhead by approximately 10% compared to the MPC-based scheme. Additionally, Table I presents a comparative summary of the

advantages of Mobius relative to the MPC-based scheme.

VII. Discussion and Future Work

We identify a new form of attack on SSLE that allows a malicious leader to propose multiple publicly verifiable states, thereby violating fundamental security properties including safety and liveness. To mitigate this threat, we introduce an approximately unique randomization mechanism and present Mobius, a universal SSLE protocol that maintains security without imposing additional trust assumptions.

An interesting direction for future work is to explore the potential risks inherent in the leader election process and their impact on the underlying protocols. We also believe that designing a leader election protocol with enhanced security and resilience against more complex adversarial models is another worthwhile direction. Additionally, the proposed randomization technique can be applied to improve other privacy-preserving protocols.

Acknowledgment

We would like to thank our shepherd and the anonymous reviewers for their valuable comments. This work was supported in part by the National Cryptologic Science Fund of China (No. 2025NCSF02034), in part by the National Natural Science Foundation of China (No. 62172396, No. 62572460), and in part by Zhongguancun Laboratory.

References

[1] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, 2019, pp. 347–356.

[2] M. Castro, B. Liskov et al., "Practical byzantine fault tolerance," in OsDI, vol. 99, no. 1999, 1999, pp. 173–186.

[3] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020, pp. 106–118.

[4] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains," in Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, 2020, pp. 1–11.

[5] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 31–42.

[6] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in 40th annual symposium on foundations of computer science (cat. No. 99CB37039). IEEE, 1999, pp. 120–130.

[7] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37. Springer, 2018, pp. 66–98.

[8] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 913–930.

[9] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." IACR Cryptol. ePrint Arch., vol. 2016, no. 919, 2016.

[10] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in Proceedings of the 26th symposium on operating systems principles, 2017, pp. 51–68.

[11] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in Annual international cryptology conference. Springer, 2017, pp. 357–388.

[12] J. Neu, S. Sridhar, L. Yang, D. Tse, and M. Alizadeh, "Longest chain consensus under bandwidth constraint," in Proceedings of the 4th ACM Conference on Advances in Financial Technologies, 2022, pp. 126–147.

[13] L. Kiffer, J. Neu, S. Sridhar, A. Zohar, and D. Tse, "Security–throughput tradeoff of nakamoto consensus under bandwidth constraints," Cryptology ePrint Archive, 2023.

[14] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," Journal of the ACM, vol. 71, no. 4, pp. 1–49, 2024.

[15] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election," in Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, 2020, pp. 12–24.

[16] "Lmd score boosting," Nov 2021. [Online]. Available: https://github.com/ethereum/consensusspecs/pull/2730

[17] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021, pp. 446–465.

[18] D. Boneh, A. Partap, and L. Rotem, "Post-quantum single secret leader election (ssle) from publicly re-randomizable commitments," Cryptology ePrint Archive, 2023.

[19] Catalano, Dario and Fiore, Dario and Giunta, Emanuele, "Efficient and universally composable single secret leader election from pairings," in IACR International Conference on Public-Key Cryptography. Springer, 2023, pp. 471–499.

[20] M. Backes, P. Berrang, L. Hanzlik, and I. Pryvalov, "A framework for constructing single secret leader election from mpc," in European Symposium on Research in Computer Security. Springer, 2022, pp. 672–691.

[21] D. Catalano, D. Fiore, and E. Giunta, "Adaptively secure single secret leader election from ddh," in Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing, 2022, pp. 430–439.

[22] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," arXiv preprint arXiv:2003.03052, 2020.

[23] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," Springer-Verlag, 1991.

[24] I. B. DamgXe, "Commitment schemes and zero-knowledge protocols," Springer Berlin Heidelberg, 1999.

[25] A. D. Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge proof systems," in Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings, 1987.

[26] Y. Desmedt, "Threshold cryptosystems," in International Workshop on the Theory and Application of Cryptographic Techniques. Springer, 1992, pp. 1–14.

[27] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in Annual International Cryptology Conference. Springer, 1999, pp. 148–164.

[28] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in Annual international cryptology conference. Springer, 2018, pp. 757–788.

[29] B. Wesolowski, "Efficient verifiable delay functions," in Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38. Springer, 2019, pp. 379–407.

[30] R. Canetti, A. Jain, and A. Scafuro, "Practical uc security with a global random oracle," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 597–608.

[31] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven, "The wonderful world of global random oracles," in Annual international conference on the theory and applications of cryptographic techniques. Springer, 2018, pp. 280–312.

[32] P. Feldman and S. Micali, "Optimal algorithms for byzantine agreement," in Proceedings of the twentieth annual ACM symposium on Theory of computing, 1988, pp. 148–161.

[33] L. Freitas, A. Tonkikh, A.-A. Bendoukha, S. Tucci-Piergiovanni, R. Sirdey, O. Stan, and P. Kuznetsov, "Homomorphic sortition–single secret leader election for pos blockchains," Cryptology ePrint Archive, 2023.

[34] "Beacon chain," 2025. [Online]. Available: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#get_beacon_proposer_index

[35] "Update proposer_score_boost to 40 percent #2895," May 2022. [Online]. Available: https://github.com/ethereum/consensus-specs/pull/2895/files
[36] OffchainLabs, "Prysm," Apr 2025. [Online]. Available: https://github.com/OffchainLabs/prysm
[37] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in Proceedings 42nd IEEE Symposium on Foundations of Computer Science. IEEE, 2001, pp. 136–145.
[38] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in Theory of Cryptography Conference. Springer, 2013, pp. 477–498.
[39] H. Dou, P. Ni, Y. Gao, and J. Xu, "Mobius: Enabling byzantine-resilient single secret leader election with uniquely verifiable state," Cryptology ePrint Archive, Paper 2025/2191, 2025. [Online]. Available: https://eprint.iacr.org/2025/2191
[40] "Secret single-leader election," 2019. [Online]. Available: https://web.archive.org/web/20191228170149/https://github.com/protocol/research-RFPs/blob/master/RFPs/rfp-6-SSLE.md
[41] A. O'Neill, "Definitional issues in functional encryption," Cryptology ePrint Archive, 2010.
[42] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in Theory of Cryptography: 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings 8. Springer, 2011, pp. 253–273.
[43] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in Proceedings of the 1st ACM Conference on Computer and Communications Security, 1993, pp. 62–73.
[44] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im) possibility of obfuscating programs," in Annual international cryptology conference. Springer, 2001, pp. 1–18.
[45] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38. Springer, 2018, pp. 565–596.
[46] J. Håstad, "The square lattice shuffle," Random Structures and Algorithms, vol. 29, no. 4, pp. 466–474, 2006.
[47] H. Wee, "Attribute-hiding predicate encryption in bilinear groups, revisited," in Theory of Cryptography Conference. Springer, 2017, pp. 206–233.
[48] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23. Springer, 2019, pp. 23–41.
[49] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," Theoretical Computer Science, vol. 777, pp. 155–183, 2019.
[50] F. D'Amato, J. Neu, E. N. Tas, and D. Tse, "Goldfish: No more attacks on proof-of-stake ethereum," Cryptology ePrint Archive, 2022.
[51] S. Azouvi and D. Cappelletti, "Private attacks in longest chain proof-of-stake protocols with single secret leader elections," in Proceedings of the 3rd ACM Conference on Advances in Financial Technologies, 2021, pp. 170–182.
[52] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Everything is a race and nakamoto always wins," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 859–878.
[53] I. Cascudo and B. David, "Albatross: publicly attestable batched randomness based on secret sharing," in International Conference on the Theory and Application of Cryptology and Information Security. Springer, 2020, pp. 311–341.
[54] I. Doidge, R. Ramesh, N. Shrestha, and J. Tobkin, "Moonshot: Optimizing block period and commit latency in chain-based rotating leader bft," in 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2024, pp. 470–482.

Appendix A
Related Work

SSLE. Single Secret Leader Election (SSLE) is originated from Protocol Labs' foundational research [40], which preliminarily outlined a function encryption-based construction [41], [42] to remedy the deficiencies inherent in existing secret leader election schemes. Building upon this prior work, Boneh et al. [15] further formalize the notion of SSLE.

Current SSLE constructions can be classified into four main approaches: indistinguishability obfuscation (iO) based schemes [43], [44], threshold mechanisms [45], shuffling techniques [46], and secure multi-party computation (MPC) [20]. Among these constructions, iO-based construction [15] demonstrates superior theoretical efficiency, achieving optimal performance in latency, communication cost, and on-chain overhead. However, practical limitations of iO have restricted the widespread adoption of these schemes. For achieving optimal on-chain efficiency, Backes et al. [20] propose an MPC-based framework. This construction employs a binary-search-like method, which randomly eliminates half of the parties in each iteration and eventually gets exactly one leader. Unfortunately, this approach results in a latency of $O(\log n)$, thereby undermining its practical applicability. For threshold-based construction, Boneh et al. [15] propose a scheme utilizing threshold fully homomorphic encryption (TFHE). While more practical than the constructions of iO and MPC, their scheme still introduces complexity due to homomorphic operations and requires a setup phase to redistribute keys after each election. To improve efficiency, Catalano et al. [19] propose a novel SSLE scheme based on a specialized function encryption scheme, Public Key Encryption with Keyword Search (PEKS) [47]. This approach achieves TFHE-like functionality without complex circuit operations. Moreover, by moving threshold decryption to off-chain processes, the work significantly reduces on-chain overhead, making the threshold-based SSLE protocol more efficient.

The shuffle-based approach is deemed the most practical among these constructions. In [15], parties are recorded in a list of commitments, which are re-randomized in each round, and a random beacon selects the leader. Unlike other constructions, shuffle-based SSLE protocols rely only on a series of fundamental primitives, including shuffle, commitment and random beacon, which makes them simpler to implement. Moreover, this method requires no reconfiguration when the unregistered node requests to participate in the election, further improving efficiency and practicality. Building on [15], Catalano et al. [21] propose an enhanced shuffle-based method that reuses the same group element during registration, reducing communication costs by half. They further introduce an optimized scheme where the list is divided into two parts: one shuffled and the other only re-randomized (without shuffling). This design preserves forward security even against adaptive adversaries.

Other Secret Leader Election. Prior to the introduction of the SSLE protocol, random functions were commonly employed in distributed protocols to ensure fairness and unpredictability in the leader election process. Some studies employ *pseudo-random random function*s (PRFs) [11] or hash functions [9], [48] for leader election. Specifically, all parties compute the function using a shared seed, which ensures fairness and

unpredictability. Since the seed is public, participants can independently verify the result and determine the identity and number of elected leaders. However, this public verifiability inherently implies that the unpredictability is weaker than what is provided by verifiable random functions (VRFs).

Another prevalent approach involves the use of VRFs [10], [49], [50]. In this scenario, each node computes a random value and a corresponding proof using its private key. Leaders are selected based on whether their VRF output is the smallest or falls within a predefined threshold. Crucially, the leader's identity remains unknown until they disclose their proof. While VRFs provide strong fairness and unpredictability, their probabilistic nature means they are unable to ensure a fixed, predetermined number of leaders.

Attacks of Multiple Leaders. Most leader-based distributed protocols rely on the fundamental assumption that each round has a unique honest leader to maintain security. Violations of this assumption (i.e., the emergence of multiple leaders in a single round) can compromise critical protocol properties such as liveness.

Recent research on secret leader election mechanisms highlights this vulnerability. Azouvi et al. [51] demonstrate how multiple leaders in longest-chain Proof-of-Stake (PoS) protocols facilitate consensus attacks. Their analysis reveals that, compared to SSLE, these approaches reduce the cost of private attacks [52]—the most severe threat to longest-chain protocols—by 25% for equivalent security parameters (e.g., confirmation latency). This security degradation forces nodes to increase settlement times, directly impairing protocol efficiency. The multi-leader scenario also exacerbates grinding attack vulnerabilities [11], a persistent challenge in PoS systems. Azouvi's findings indicate a 10% reduction in fault tolerance thresholds under these conditions, introducing a fairness challenge to the system.

Appendix B
Formal Definitions of Building Blocks

In this section, we present the formal definitions of the fundamental building blocks analogous to those in [21], specifically the functionalities of non-interactive zero-knowledge proof and random beacon.

Non-Interactive Zero-knowledge Proof. We consider an ideal UC-NIZK for a single prover and a single proof, and provide a functionality $\mathcal{F}_{NIZK}^{\mathcal{R}}$ in Fig. 11.

Random Beacon. We also demonstrate the functionality of random beacon, which is first introduced in [53] and denoted as $\mathcal{F}_{ct}^{n}$ in Fig. 12.

Appendix C
Continuous Attack Strategy under Multi-Leader Protocol

This section delineates the process of leader election within Ethereum's Gasper [22] consensus protocol and demonstrates how adversarial exploitation of multi-leader scenarios makes the optimization mechanism inefficient and subverts the security properties of it.

Setup.

- Sample a common reference string from distribution $\mathcal{D}$, $crs \leftarrow^{\$} \mathcal{D}$.
- Upon receiving $(\mathsf{setup}, \mathcal{P})$, output $(\mathsf{setup}, crs) \hookrightarrow \mathcal{P}$.

Prove. Upon receiving $(\mathsf{prove}, sid, x, \omega) \hookleftarrow \mathcal{P}$ for the first time:

- If $(x, w) \in \mathcal{R}$, sample proof π, store $(\mathsf{prove}, sid, x, \omega, \pi)$, and broadcast $(\mathsf{proof}, sid, x, \pi)$.
- Otherwise, abort.

Verify. Upon receiving $(\mathsf{verify}, sid, x, \pi) \hookleftarrow \mathcal{V}$:

- If (sid, x, ω, π) has been stored, then return $(\mathsf{verified}, sid, x, \pi)$.
- Otherwise, abort.

Fig. 11: Ideal functionality $\mathcal{F}_{NIZK}^{\mathcal{R}}$ that models NIZK.

Upon receiving (toss, sid) from $n - f$ nodes, sample $x \leftarrow^{\$} [n]$ and broadcast $(\mathsf{tossed}, sid, x)$; otherwise, return $(\mathsf{tossed}, sid, \perp)$.

Fig. 12: Ideal functionality $\mathcal{F}_{ct}^{n}$ that models random beacon.

A. Leader Election in Gasper

The Ethereum consensus layer adopts a shuffle-based framework to elect a single leader (a.k.a., proposer) across executions. Specifically, building upon the framework, the protocol executes the coin-tossing and shuffle operations around a series of random factors. Informally, the detailed process is as follows.

Randomness generation. Within the Ethereum consensus protocol, the random value that is utilized to elect the leaders is generated through an epoch-based aggregation. Specifically, in each slot, a publicly verifiable random beacon, called `RANDAO`, is produced. Then, the proposer for that slot signs this value and includes it in the proposal as part of the block (i.e., `randao_reveal`). When an epoch ends, all `randao_reveal` values from the canonical chain blocks are aggregated via `XOR` operations to derive the final epoch random factor.

Integration between randomness and shuffle. Furthermore, the Ethereum consensus protocol employs the above epoch-based random factors to construct a leader election mechanism by shuffling and coin-tossing. Upon getting the random value at the end of epoch e, the obtained randomness serves as the entropy source for the committee and leader (a.k.a., proposer) distribution in epoch $e + 2$.

Specifically, we take the epoch e as an example to illustrate the election process (cf. Fig. 13). When transitioning into epoch e, the parties will shuffle the participant list by the random value r_{e-2} generated at the end of epoch $e - 2$. After obtaining the refreshed list, the parties can deterministically select proposers by tossing a coin with r_{e-2} and a certain slot index θ (in epoch e) as inputs at the start of epoch e. Consequently, the output of the process identifies a designated

leader that is responsible for the block proposal in slot θ.

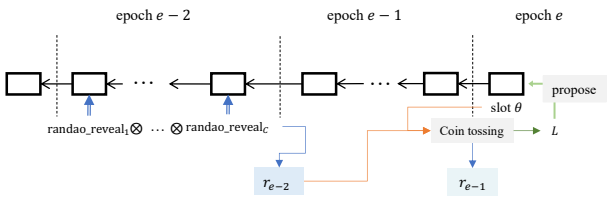


Fig. 13: An example of the election process in Ethereum.

B. Detailed Attack Strategy

Following the attack strategy presented in Section III, we further apply the cross-epoch withholding strategy [17] to the aforementioned strategy and propagate the chain split into subsequent epochs, thereby compromising the liveness property of the protocol. Similarly, we assume that n validators divided into committees across C slots per epoch, where $W := n/C$ validators are eligible to vote in a slot with $2f+1$ being honest and f controlled by the adversary among them. For clarity, we suppose there are two competing chains as depicted in Fig. 2(a), which are denoted as ch and ch'. Informally, in this attack scenario, the adversary in slot θ executes two complementary adversarial strategies by selective message withholding to manipulate honest parties' consensus decisions: intra-epoch deception that manipulates honest parties in the subsequent slot $\theta+1$, and cross-epoch disruption that disturbs the honest parties in slot $s+C$ (where each epoch consists of C slots). Assuming the first epoch and slot following the network reaching GST are respectively epoch 0 and slot 0, the attack can proceed as follows (cf. Fig. 14 and 15):

- **Attack Preparation.** Before epoch 0, the adversary prepares to initiate the state uniqueness attack aimed at undermining the boosting mechanism. Specifically, the adversary leverages the network propagation delays to maintain a weight equilibrium between ch and ch', which leads to that left chain ch with a random value r_{-2} accumulates a vote weight of $w(ch) = f$ whereas the right chain ch' disseminates a distinct random value $r_{-2'}$ with a weight of $w(ch') = f+1$. In this scenario, honest nodes that are misguided to support different chains will adopt distinct random values for leader selection. For instance, honest nodes $\mathcal{P}$ supporting ch hold a random value r_{-2} on ch, while honest nodes $\mathcal{P}'$ maintaining ch' retain a distinct random value $r_{-2'}$ on ch'.
- **Attack Activation (Epoch 0).** Suppose that two leaders L_θ and $L_{\theta'}$ at slot θ are both valid but determined by different random values. In this case, the adversary $\mathcal{A}_0$ can activate the state uniqueness attack and $\mathcal{A}_\theta$ can further recover the balance attack as follows:
  - Slot 0: Before the proposing process, $\mathcal{A}_{-1}$ sends 2 votes for ch to L_0, while withholding them from $L_{0'}$. As a result, L_0 and $L_{0'}$ will respectively extend ch and ch' with blocks B_0 and $B_{0'}$, resulting in the condition of $w(ch) = w(ch')+1$, where $w(ch) = f+2+40\%W$ and $w(ch') = f+1+40\%W$. Before the voting process, $\mathcal{A}_{-1}$ releases 2 votes of ch' to $\mathcal{P}_{0'}$, where $|\mathcal{P}_{0'}| = f$. Then, during the voting process, $\mathcal{P}_0$ ($|\mathcal{P}_0| = f+1$) will vote for B_0, whereas $\mathcal{P}_{0'}$ will vote for $B_{0'}$, as they recognize ch and ch' as their respective main chain. At the end of slot 0, it follows that $w(ch) = w(ch') = 2f+3$.
  - Slot 1: Before the proposing process, the adversary $\mathcal{A}_0$ sends 1 vote for B_0 to L_1 and 1 vote for $B_{0'}$ to $L_{1'}$, which induces L_1 to extend ch with B_1, while simultaneously prompting $L_{1'}$ to extend ch' with $B_{0'}$. Furthermore, both chains satisfy that $w(ch) = w(ch') = 2f+4+40\%W$. Prior to the voting process, $\mathcal{A}_0$ releases 1 vote for B_0 to $\mathcal{P}_1$, and 1 vote for $B_{0'}$ to $\mathcal{P}_{1'}$, where $|\mathcal{P}_1| = f$ and $|\mathcal{P}_{1'}| = f+1$, to have honest nodes cast their votes for distinct choices. At the end of slot 1, it holds that $w(ch') = w(ch)+1$, thereby precisely satisfying the initial condition set forth in slot 0.
  - Slot θ ($2 < \theta < C$): Repeating the executions of slot 0 and slot 1 in sequence, it always holds that $w(ch) = w(ch')$ or $w(ch)+1 = w(ch')$ at the end of each slot, with nearly $33.3\%W$ for each chain.
- **Semi-Steady State (Epoch 1).** During epoch 1, the adversary $\mathcal{A}$ in epoch 0 releases more withheld votes, which is from epoch 0 and the adversary-controlled parties has not released these vote in epoch 0, to pursue two objectives: (1) first, the strategy keeps splitting the honest parties into two groups, one of which sees ch as leading and voting for it, and all the adversary needs to do is release withheld votes so as to reaffirm the honest parties in their illusion that whatever chain they previously voted on in epoch 0 happens to be still leading; (2) secondly, the enforces the leaders that are respectively elected by r_{-1} and $r_{-1'}$ sees ch and ch' as leading and extending it respectively. As a result, at the end of epoch 1, there are still two chains with about $33.3\%n$ votes and thus neither get finalized.
- **Steady State (Epoch $e > 1$).** During epoch e, the attack reaches a steady state, where the adversary is required to repeat its actions in each epoch. At this time, the adversary selectively releases withheld votes from epoch $e-1$ to achieve the same objectives in epoch 1. At the end of epoch e, there are still two chains with about $33.3\%n$, which continues indefinitely. Thus, neither chain can reach the finalization condition, leading to a breakdown of the liveness property.

The above process can continue indefinitely, leading to neither of the completing chains reaching the finalization condition. As a result, no blocks can be finalized and the liveness property of the protocol is broken down.

Appendix D
Challenges of Efficiency Optimization

While prior works [1], [54] have demonstrated protocol acceleration through mature parallelization techniques via stacking different instances, addressing the inherent four-round

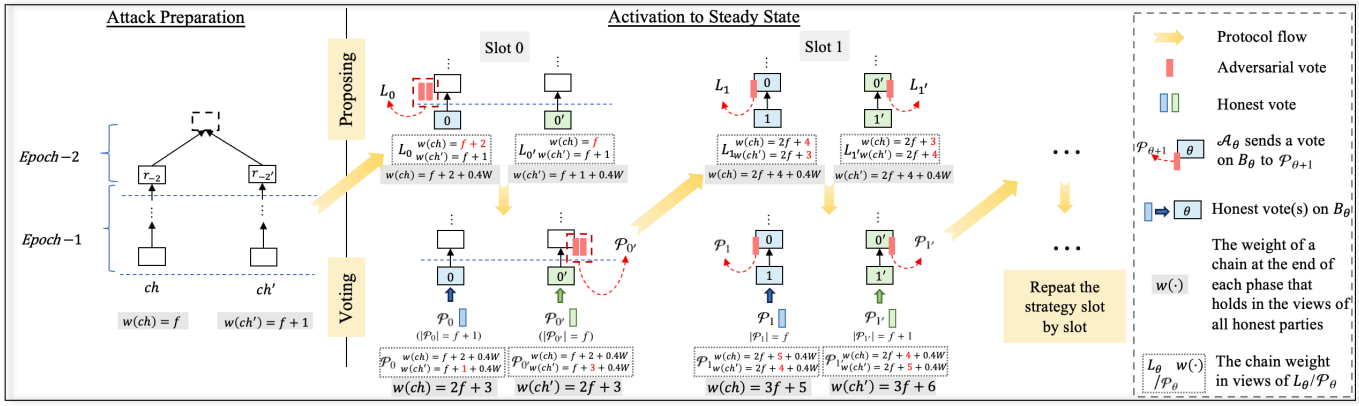


Fig. 14: The attack workflow caused by multiple leaders, wherein ch and ch' represent two competing chains that both are prevented from finalization. Additionally, the blocks in ch and ch' are proposed by leader L_θ and $L_{\theta'}$ and supported by two equal-sized node groups $\mathcal{P}_\theta$ and $\mathcal{P}_{\theta'}$, where θ represents the corresponding slot number.

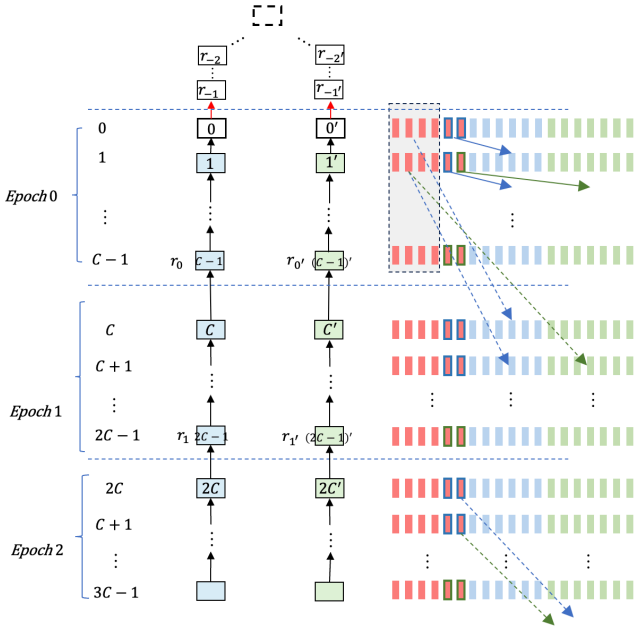


Fig. 15: The overview of the cross-epoch attack with multiple leaders.

latency in our context presents particular technical hurdles. These obstacles emerge primarily because[7]:

- **Sequential-state dependencies.** The "*Elect-Randomize*" design pattern of the protocol imposes a fundamental constraint that requires strictly sequential randomization of state, particularly for the node list **h**. The constraint emerges from security considerations: the adversary could exploit reused states by rejecting randomization attempts, and electing by a reused state can potentially leak the privacy about honest nodes. Thus, the current parallelization technique by stacking instance is not suitable for our protocol.

[7]We also provide a detailed explanation of the challenges in the full version [39].

- **Non-composable confidence parameter.** Furthermore, the updates of states and commitments also cannot be simply combined together to achieve paralleled. This constraint stems from that, the interleaved protocol flow leads to that message delivery processes from different nodes all rely on the confidence parameter. Consequently, an adversary can prevent state update from honest nodes through its own faulty behavior, irreversibly compromising the privacy of the honest.

Therefore, these interdependent constraints demand a novel construction that carefully handles parallel execution opportunities with strict security requirements.