Shardora: Towards Scaling Blockchain Sharding via Unleashing Parallelism

Yu Tao*, Lu Zhou*¶, Lei Xie†, Dongming Zhang†, Xinyu Lei‡, Fei Xu†, Zhe Liu†
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
†Zhejiang Lab, China
‡Michigan Technological University, USA
¶Collaborative Innovation Center of Novel Software Technology and Industrialization, China
{yu_tao, lu.zhou, zhe.liu}@nuaa.edu.cn
{xielei, zhangdongming, feixu}@zhejianglab.com
xinyulei@mtu.edu

***Abstract*—Sharding emerges as a promising solution to enhance blockchain scalability. However, it faces two critical limitations during shard reconfiguration: (1) the TPS-Degradation issue, arising from ledger synchronization conflicts during transaction processing, and (2) the Zero-TPS issue, caused by disruptions in transaction processing due to key negotiation. To this end, we propose Shardora, a blockchain sharding system for scaling blockchain by unleashing parallelism. In Shardora, we implement two essential mechanisms: (1) A parallelized dual committee framework with a reputation mechanism to mitigate the TPS-Degradation issue while ensuring system security. (2) A parallelized key pre-negotiation mechanism with a secret-reuse strategy to avoid the Zero-TPS issue while maintaining a continuously high TPS. We prove that Shardora offers theory-guaranteed security. We implement a prototype of Shardora and deploy it on Alibaba Cloud. Experimental results demonstrate that Shardora addresses the limitations by significantly reducing the overhead of both ledger synchronization and key negotiation, which outperforms state-of-the-art sharding schemes by at least 90%. In addition, Shardora shows its superior performance in terms of throughput and latency, achieving a peak throughput of 8300 TPS on a single shard with 600 nodes under LAN conditions. The code of Shardora is publicly available on GitHub.**

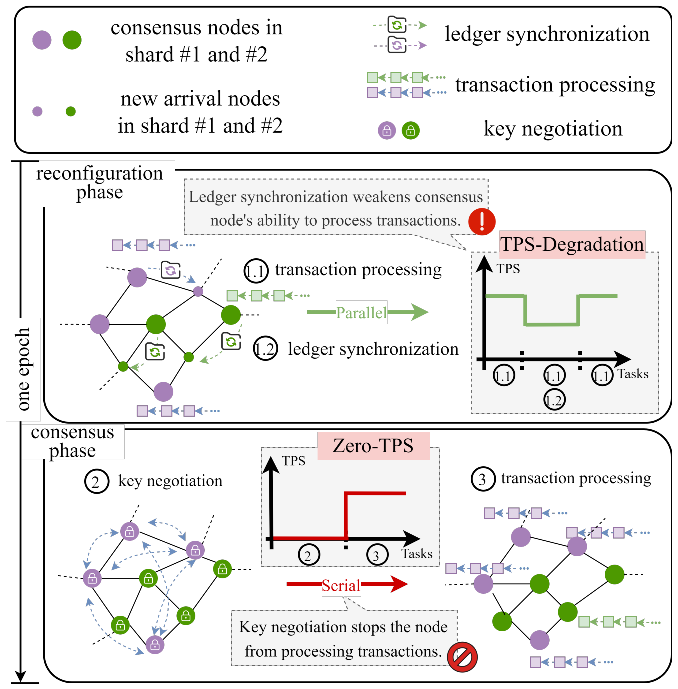
I. Introduction

Blockchain technology has revolutionized various industries by offering a decentralized and secure transaction platform, which supports many security-critical applications such as supply chain [1], [2], finance [3], [4] and healthcare [5], [6], etc. Unfortunately, traditional blockchain systems suffer from low scalability. For example, Bitcoin [7] only processes up to 7 transactions per second (TPS), and the initial Ethereum [8] processes 15 TPS. Such poor scalability impedes the widespread adoption of blockchain in real-world applications. As one of the most promising solutions to address the low scalability issue, blockchain sharding technique has been proposed in prior research. It randomly partitions the network into numerous smaller segments called *shards* [9]–[11], with each shard maintaining its own ledger and processing a subset of transactions in parallel. This approach helps reduce each shard's communication and storage overhead, thereby achieving a high transaction throughput. State-of-the-art systems (e.g., Elastico [12], OmniLedger [13], RapidChain [14], tMPT [15], etc.) typically employ a *two-phase paradigm* to periodically execute shard reconfiguration. These systems operate in defined time intervals known as *epochs*, each comprising two phases: a reconfiguration phase and a consensus phase. During the reconfiguration phase, nodes are strategically shuffled among different shards to ensure system security, making it difficult for potential adversaries to manipulate specific nodes within targeted shards. During a consensus phase, nodes within each shard run a consensus protocol to validate and agree on blocks of valid transactions.

Limitations of Prior Art. Roughly speaking, the previous blockchain sharding solutions can be divided into two types: PBFT-based solutions [12]–[15] and threshold signature-based solutions [16], [17] (see Section VI for details). These solutions all consider the presence of Byzantine adversaries. (1) PBFT-based solutions adopt the Practical Byzantine Fault Tolerance (PBFT) protocol [18] for transaction processing during consensus phases. In the reconfiguration phase, nodes are shuffled dynamically among different shards. After node shuffling, new arrival nodes within a shard are required to synchronize the ledger state of their current shard before they can participate in consensus. (2) Since PBFT exhibits quadratic communication complexity, threshold signature-based solutions have been introduced to achieve linear communication complexity for higher scalability. Nevertheless, threshold signatures necessitate that, during each reconfiguration phase, the consensus nodes within the newly formed shard must renegotiate their keys for signature. Overall, as shown in Figure 1, the prior blockchain sharding systems encounter potential limitations, which are illustrated in detail as follows.

TPS-Degradation Issue: For both PBFT-based and threshold signature-based solutions, consensus nodes (those that remain in their original shards post-reshuffle) are responsible for synchronizing a substantial volume of the ledger state with

Zhou Lu is the corresponding author.
This work was supported by the National Key R&D Program of China (2022YFB2702000), the National Natural Science Foundation of China (62472218, 62071222, 62032025, U21A20467, U20A20176, U22B2029), the Shenzhen Science and Technology Program (JCYJ20210324134810028), and the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX24_0611).



consensus nodes in shard #1 and #2
new arrival nodes in shard #1 and #2
ledger synchronization
transaction processing
key negotiation
reconfiguration phase
one epoch
Ledger synchronization weakens consensus node's ability to process transactions.
TPS-Degradation
1.1 transaction processing
Parallel
1.2 ledger synchronization
TPS
Tasks
consensus phase
Zero-TPS
2 key negotiation
3 transaction processing
Serial
Key negotiation stops the node from processing transactions.

Fig. 1: An overview of limitations in existing sharding systems. *TPS-Degradation issue:* During reconfiguration phase, consensus nodes concurrently perform two tasks: (1.1) processing transactions within shards alongside the other consensus nodes, and (1.2) synchronizing the historical shard ledger with the new arrival nodes. The ledger synchronization weakens the consensus node's ability to process transactions, leading to TPS-Degradation. *Zero-TPS issue:* During consensus phase, to deploy BFT consensus protocols with threshold signatures, all consensus nodes must (2) negotiate keys before (3) conducting consensus. This requirement stops consensus nodes from processing transactions, leading to Zero-TPS.

new arrival nodes. Concurrently, they must process transactions to uphold system liveness. This dual responsibility significantly degrades transaction throughput during the ledger synchronization procedure. For instance, in our experiments on RapidChain, it is observed that TPS is decreased by about 50% during this procedure.

Zero-TPS Issue: For threshold signature-based solutions, consensus nodes must engage in mutual key negotiation each time the shard undergoes reconfiguration. It leads to Zero-TPS issue since transactions cannot be processed during the key negotiation procedure. Zero-TPS issue seriously reduces the availability of the blockchain system since the key negotiation procedure typically lasts for hundreds of seconds.

Challenges & Solutions. To address these issues, we propose a blockchain sharding system with higher TPS named Shardora[1]. The core idea of Shardora involves constructing parallelization mechanisms from two dimension, ensuring continuous and efficient transaction processing throughout the shard reconfiguration. However, to achieve this goal, two technical challenges remain:

The first challenge lies in mitigating TPS-Degradation while maintaining ledger synchronization. The TPS-degradation issue arises naturally as consensus nodes necessitate a resource trade-off between transaction processing and ledger synchronization. To address this challenge, we propose a dual committee structure that decouples the execution logic of ledger synchronization and transaction processing, enabling independent and concurrent task execution. Specifically, each shard contains two committees: a consensus committee for transaction processing, and a waiting committee that stands by and records newly committed transactions. The dual committee structure enables newly arrived nodes to synchronize all historical ledger from waiting committee, without hampering transaction processing by consensus committee.

Unfortunately, dividing the shard into smaller committees raises significant security concerns. It shifts the system's security dependence from the original shard to smaller consensus committees. Compared to shards, smaller consensus committees are more vulnerable to corruption. To ensure the security of dual committee design, we further introduce a reputation-based dual committee shuffling strategy. Nodes' behaviors are analyzed into reputation scores, indicating their capabilities and reliability. Based on their reputation, the consensus and waiting committees periodically conduct partial node eviction and election, respectively. This ensures that the consensus committee comprises the most reliable and capable nodes.

The second challenge lies in maintaining transaction processing capabilities during key negotiations. Within an epoch, key negotiation should precede transaction processing, as nodes require the negotiated keys to proceed. To tackle this challenge, we present a parallelized key pre-negotiation mechanism. It allows key negotiation procedure of the next epoch to run concurrently with the ongoing transaction processing of the current epoch, thus avoiding the Zero-TPS issue.

However, with partial node shuffling, a majority of consensus nodes in the current epoch may be re-elected to the consensus committee for the next epoch. These nodes are tasked with dual responsibilities: managing transaction processing and participating in key negotiation for the next epoch. As key negotiation demands considerable communication and computational resources, it poses a risk of inducing the TPS-Degradation issue in prior works. To improve the efficiency of transaction processing during key negotiation, we further introduce a secret-reusable decentralized key negotiation mechanism. It enables these re-elected nodes to reuse previously negotiated secrets in the current negotiation, thereby diminishing resource consumption for consensus nodes and making the TPS-degradation issue significantly mitigated.

Contribution. Our main contributions are as follows.

- **Cost-effective Reconfiguration.** By decoupling and parallelizing the dual committee within the shard, we reduce ledger synchronization overhead of consensus nodes, thereby mitigating the TPS-Degradation Issue. It is observed 50-1000× reduction in ledger synchronization overhead compared to prior works [14], [15].
- **High Security.** Shardora employs a reputation-based node shuffling mechanism to prevent shard corruption, offering lower and negligible system failure probability than random-based node shuffling.

[1]Our source code is available at https://github.com/tenondvpn/shardora.

- **Availability-aware Reconfiguration.** Utilizing parallelized key pre-negotiation, Shardora is the first sharded blockchain to achieve linear consensus complexity while ensuring continuous transaction processing throughout reconfiguration, thereby avoiding Zero-TPS issue.
- **Lightweight Key Negotiation.** We introduce a lightweight key negotiation mechanism that allows consecutive elected consensus nodes to reuse their previously negotiated secret for the current negotiation, cutting communication and computation overhead on consensus nodes for key negotiation by at least 90% compared to [16], [17].

II. Problem Formulation

A. System Model

Figure 2 shows the system model. Shardora system contains 2 different types of shards, which are introduced below.

Shard. Shardora comprises multiple transaction shards and a root shard.

1) *Transaction Shard:* These shards handle transaction processing through consensus protocols.
2) *Root Shard:* It tracks and updates node reputation scores within each transaction shard. Based on these scores, it executes partial node shuffling to ensure system security.

For each shard, it is composed of 3 types of blockchain nodes, which form 2 types of committees, respectively.

Node. Each node (i.e., a device with computing power) can freely join in the system. These nodes can be categorized into consensus nodes, waiting nodes, and candidate nodes.

1) *Consensus Node:* They are responsible for processing transactions via consensus protocols to win block rewards and reputation scores. Consensus nodes are elected within respective shards to form a *consensus committee*.
2) *Waiting Node:* In addition to consensus nodes, the remaining nodes within a shard are classified as waiting nodes. These nodes do not engage in transaction processing but continuously maintain the most recent shard ledger state and await potential election to become consensus nodes. A group of waiting nodes forms a *waiting committee*. Note that, newly arrived nodes within shards are automatically designated as waiting nodes and synchronize the current shard ledger from existing waiting nodes.
3) *Candidate Node:* Candidate nodes hold a transitional status, indicating their potential elevation to consensus nodes in the next epoch, contingent on successful key negotiation. Success elevates them to consensus nodes. Conversely, failure relegates them back to waiting nodes.

B. Transaction Model

A transaction model refers to the way transactions are structured, validated, and recorded on the blockchain. Our scheme adopts the account/balance model for transaction [19]. The system will process a transaction only after confirming that there is sufficient balance in the trading account to ensure that the transaction proceeds properly. In our scheme, we categorize transactions into the following types:

1) *Time Transaction:* It serves to achieve clock synchronization across different shards. Periodically, the root shard initiates a time transaction, which is then finalized through consensus to generate a *time block*. This time block is then broadcast to all transaction shards, initiating a new epoch and periodic shard reconfiguration.
2) *Transfer Transaction:* It is used to update the state of user's account. A transfer transaction can be finished via two consecutive operations: withdraw and deposit. Taking the transfer from account A to B as an example, the process requires a withdrawal from A and a subsequent deposit into B, ensuring transaction atomicity.
3) *Creation Transaction:* Nodes can join the system by initiating a creation transaction, thus designating themselves within either root or transaction shards. Additionally, a transfer transaction needs to be launched to deposit funds into the public account as the node's stake.
4) *Election Transaction:* In each epoch, the root shard initiates an election transaction for each transaction shard to elect its next consensus committee. After achieving consensus on the election transaction, the election results are included in an *election block*. It will be broadcast to the relevant shards to inform them to shuffle nodes.

C. Network Model

We consider a peer-to-peer network comprising nodes with heterogeneous computing and bandwidth resources. Honest nodes are well interconnected, and messages are synchronized among them via a gossip protocol [20]. Similar to most public blockchain [12]–[14], [21], we assume that messages sent by an honest node will reach all other honest nodes within a known fixed delay Δ. For minimized latency, we only consider such Δ in slow operations like identity creation and shard assignment. For other protocols, we adopt partially synchronous model [18] with optimistic, exponentially increasing timeouts.

D. Threat Model and Security Assumption

We consider the presence of a Byzantine adversary that can corrupt up to F nodes among N participating nodes in the whole system. Similar to prior sharded blockchain schemes [12]–[14], [22], [23], we assume that the adversary can control at most $1/4$ of the nodes in the system at any given time, i.e., $F \leq N/4$. In Shardora, honest nodes obey all protocols while Byzantine adversaries may cause arbitrary failures to deviate from the protocol. We further assume our Byzantine adversary is slowly-adaptive [12]–[14] which means it can choose nodes to corrupt only at the start of a protocol or an epoch, but cannot alter its choice within one epoch. Driven by attack goals, adversaries in Shardora will perform different malicious behaviors: (1) *Simple Attack:* They continuously perform malicious behaviors, including sending invalid or inconsistent or no message, to prevent consensus from being reached. (2) *Camouflage Attack:* Due to the employment of reputation-based node shuffling, there exists a potential camouflage attack where adversaries act honestly to establish the same reputation score distribution as the honest one before

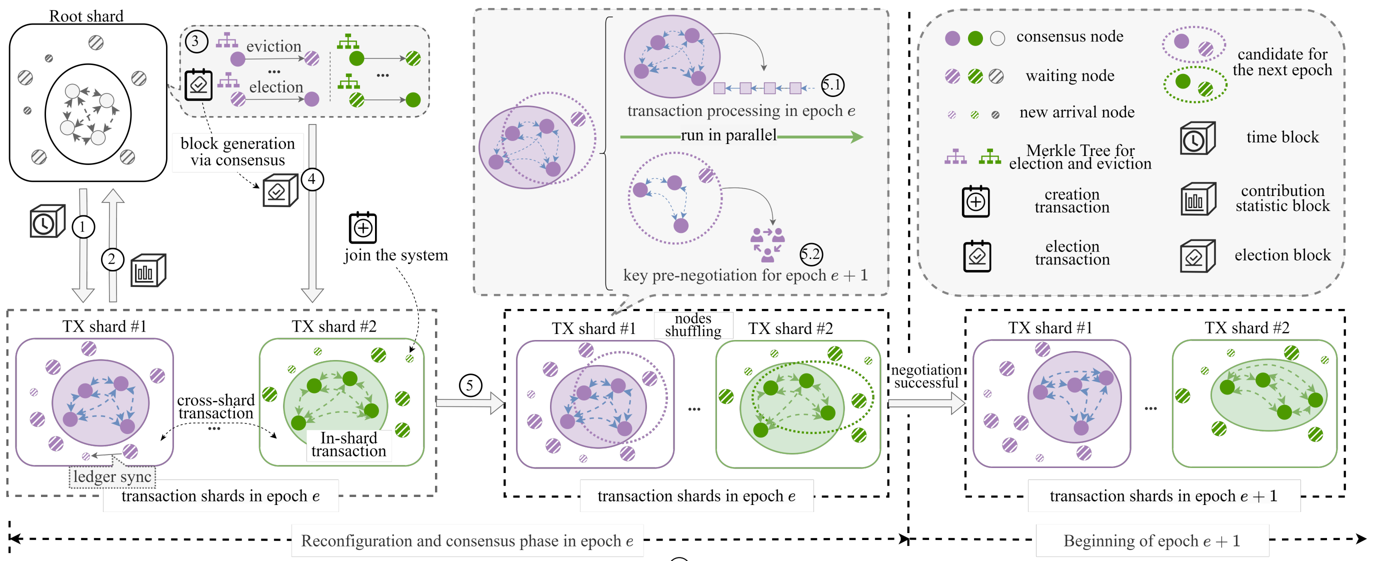

Root shard
eviction
election
block generation via consensus
join the system
transaction processing in epoch e
run in parallel
key pre-negotiation for epoch $e+1$
consensus node
waiting node
new arrival node
Merkle Tree for election and eviction
creation transaction
election transaction
candidate for the next epoch
time block
contribution statistic block
election block
TX shard #1
TX shard #2
cross-shard transaction
In-shard transaction
ledger sync
transaction shards in epoch e
nodes shuffling
negotiation successful
transaction shards in epoch $e+1$
Reconfiguration and consensus phase in epoch e
Beginning of epoch $e+1$

Fig. 2: System model. There are five major sub-phases within each epoch: ① Root shard broadcasts time block to trigger the beginning of a new epoch e. ② Transaction (TX) shards statistic and upload nodes' contribution in previous epoch $e-1$. ③ Root shard updates nodes' reputation score and performs reputation-based node election/eviction through the FTS algorithm. ④ Root shard agree on the election/eviction results to generate election blocks, which are then sent to the respective shards. ⑤ Upon receiving election blocks, transaction shards carry out node shuffling. During this phase, two tasks run in parallel: (5.1) consensus nodes continue to process transactions. (5.2) the elected candidates for the next epoch conduct key pre-negotiation. Should the key negotiation prove successful, these candidates will ascend to the role of consensus nodes in the subsequent epoch. Additionally, any node can initiate a creation transaction to join the system and become a new arrival node which can sync ledger information of the current shard from waiting nodes instead of consensus nodes.

launching the attack. (3) *Key Negotiation Attack:* On the one hand, adversaries and malicious candidates may compromise the key negotiation process by launching attacks, resulting in negotiation failures. On the other hand, adversaries may exploit the key negotiation reuse feature to compromise the system security.

III. Our Proposed Shardora

A. Overview of Shardora

Shardora can be divided into the following phases: reputation-based dual committee node shuffling, key pre-negotiation, consensus agreement and reputation scoring.

Reputation-based Dual Committee Node Shuffling. This phase is executed to ensure shards' dynamism and consensus committees' robustness, thereby enhancing the system's resistance against slow-adaptive attacks and boosting overall performance. The root shard initiates this phase at the beginning of each epoch. It achieves consensus on the reputation-based node shuffling results of each shard for the next epoch, conducting the dual committee node shuffling one epoch ahead. It includes (1) intra-shard node shuffling and (2) cross-shard node shuffling. Through intra-shard node shuffling, nodes with higher reputations are retained in the consensus committee, ensuring their reliability and capability. Cross-shard node shuffling aims to further resist slow-adaptive attacks and keep shard reputation resemble.

Key Pre-negotiation. This phase aims to prevent key negotiation from stopping transaction processing. It allows each shard to procure the consensus committee candidates in advance and initiate key negotiations for threshold signatures. Once a node in a shard learns that it has been selected as a candidate consensus node for the upcoming epoch, it will negotiate with other candidates via DKG (Distributed Key Generation) [24]–[27] to generate the required secret keys for BLS threshold signatures. Subsequently, the node can assume the role of a consensus node only upon the conclusion of these negotiations.

Consensus Agreement. During each epoch, the current consensus committee executes the consensus protocol. Meanwhile, the candidate consensus committee for the upcoming epoch is involved in a key pre-negotiation process. Such two processes run in parallel. According to the negotiated keys from the previous epoch, the current consensus committee utilizes BFT protocols with BLS threshold signature [28]–[30] as the intra-shard consensus protocol to batch valid transactions into blocks for consensus. For cross-shard transactions, our system implements the Atomic protocol from Monoxide [31].

Reputation Scoring. Shardora utilizes reputation scores to evaluate the reliability and capability of nodes. Factors such as stake, historical contributions, and recent contributions are considered in the computation of reputation scores. By leveraging a reputation-based mechanism for node shuffling and leader selection, Shardora brings the following benefits: (1) Ensuring that nodes in the consensus committee, especially leaders, are powerful and robust. (2) Improving the system throughput and security through reputation-based competition among nodes. (3) Promoting a balanced distribution of resources across shards according to reputation, making it challenging for attackers to take control of a shard.

B. Detailed Description

As a public blockchain platform, Shardora enables any node to join freely and become either a root or transaction shard node by initiating a creation transaction. This transaction requires the node to deposit some funds into the public account as its stake St_i. After the transaction is committed, the root shard will allocate the node to shards in the following epoch based on its preferences. Nodes preferring the root shard will be allocated there. Alternatively, if the transaction shard is desired, assignment occurs randomly, based on each shard's reputation distribution. Subsequently, the node becomes a waiting node in its designated shard. The shard assignment follows a balancing principle: the root shard assigns the node to the transaction shard with the lowest reputation. It ensures a balanced distribution of reputation scores across all shards.

At the beginning of each epoch, the root shard will shuffle the nodes based on their reputation scores. Specifically, it rotates specific nodes among corresponding committees intra/cross-shard, and selects a leader for each new consensus committee. To ensure the security of shards, node shuffling must possess two key properties: *Randomness* and *Resemblance*. *Randomness* means that all node shuffling and leader election results are unpredictable. *Resemblance* dictates that each shard maintains similar reputation scores to prevent any particular shard from being targeted by attacks.

1) Reputation-based Dual Committee Node Shuffling: In each epoch, the root shard will proactively perform reputation-based node shuffling for the subsequent epoch. This shuffling process is illustrated in Algorithm 1, which involves 6 subphases: epoch randomness generation, consensus committee eviction, waiting committee election, leader election, shard resemblance and ledger synchronization.

Epoch Randomness Generation. As demonstrated in line 1-3 in Algorithm 1, Shardora allows the root shard to perform epoch randomness generation in advance before the epoch starts. For example, the random value r for epoch e is generated at the end of epoch $e-1$. At the beginning of each epoch, the root broadcasts r to all shard nodes, initiating shard reconfiguration. Similar to [14], shardora utilizes the Follow-the-Satoshi (FTS) [32] algorithm for random selection and eviction, which relies on an unbiased random number. Therefore, in each epoch, the Root shard will conduct a verifiable secret sharing (VSS) of Feldman [33] to generate such random r.

Consensus Committee Eviction. After receiving the clock block from the root shard, nodes in each transaction shard will agree on their contribution in the previous epoch as $Cont_{e-1}$ and then sync this information to the root shard to update their reputation scores (line 5-7). The root shard will first rank the received contribution within each shard. Then, bottom k_1 nodes will be directly evicted (line 8-9). In addition to directly evicting k_1 nodes, k_2 nodes also need to be randomly evicted based on their reputation scores. Notably, to ensure security, Shardora requires $k_1 + k_2 \geq \frac{1}{3}n_c + 1$, a detailed proof is provided in Section IV-B2. As line 10-16 show, for the nodes remaining in the current consensus committee, the root shard generates a new list $Scores_{cn}^{\#m}$ according to the updated reputation scores and such that $maxValue$ represents the highest reputation score. Then the root shard subtracts

Algorithm 1 Reputation-based Dual Committee Node Shuffling

```
1: Procedure EpochStart($epoch\ e$)
2:     $r \leftarrow$ GENERATERANDOMNESS($e-1$)
3:     Broadcast $r$ to all nodes
4: In-Shard Node Shuffling:
5: for each tx shard $\#m$ do
6:     Consensus Committee Eviction:
7:     $Cont_{e-1} \leftarrow$ SYNCCONT(shard $\#m$, root)        ▷ Sync respective contributions to the root shard
8:     $L_c^{\#m} \leftarrow$ SORTDESCENDING($Cont_{e-1}$) ▷ Done by the root
9:     $Evicted_1^{\#m} \leftarrow$ GETNODE($L_c^{\#m}[(n-k_1+1):n]$)
10:    $L_r^{\#m} \leftarrow L_{\#m,c}[1:n-k_1]$
11:    $Scores_{cn}^{\#m} \leftarrow$ UPDATEREPUTATIONSCORES($L_r^{\#m}$)
12:    $maxValue \leftarrow$ MAXVALUE($Scores_{cn}^{\#m}$)
13:    for $i=0$ to length($Scores_{cn}^{\#m}$)-1 do
14:        $Scores_{cn}^{\#m'}[i] \leftarrow maxValue - Scores_{cn}^{\#m}[i]$
15:    $MT_{eli} \leftarrow$ GENERATEMERKLETREE($Scores_{cn}^{\#m'}$)
16:    $Evicted_2^{\#m} \leftarrow$ EVICTWITHFTS($MT_{eli}, k_2, r$)
17:    for $node$ in $Evicted_1^{\#m}$ do
18:        $Evicted_1 \leftarrow Evicted_1.append(node)$
19:    for $node$ in $Evicted_2^{\#m}$ do
20:        INSHARDSHUFFLE($node$, $waiting$, $e+1$)
21:    Waiting Committee Election:
22:    $MT_{ele} \leftarrow$ GENERATEMERKLETREE($Scores_{wn}^{\#m}$)
23:    $Elected^{\#m} \leftarrow$ ELECTEDWITHFTS($MT_{ele}, (k_1+k_2), r$)
24:    for $node$ in $Elected^{\#m}$ do
25:        INSHARDSHUFFLE($node$, $candidate$, $e+1$)
26:    KEYNEGOTIATION($candidate$, $e+1$)
27:    if success then
28:        PROMOTETOCONSENSUSNODE($candidate$)
29:    else
30:        DEMOTETOWAITINGNODE($candidate$)
31: Cross-Shard Node Shuffling:
32: $Evicted_c \leftarrow$ SORTDESCENDINGBYSCORE($Evicted_1$)
33: for all $node$ in $Evicted_c$ do
34:     $toShard \leftarrow$ SHARDWITHLOWESTREPUTATION
35:     CROSSSHARDSHUFFLE($node$, $waiting$, $e+1$, $toShard$)
36:     REQUESTLEDGERSYNC($node$, $toShard$)
```

each element in $Scores_{cn}^{\#m}$ from $maxValue$ to obtain a new list $Scores_{cn}^{\#m'}$, and uses each element in $Scores_{cn}^{\#m'}$ as a stakeholder of each node to construct a Merkle tree for random eviction. Based on this Merkle tree, the FTS algorithm runs on the root shard to evict k_2 nodes. Such FTS algorithm relies on the epoch random number as a seed.

Shardora employs two stages of node eviction in the consensus committee (corresponding to k_1 and k_2 nodes in each stage). For k_1 nodes in list $Evicted_1^{\#m}$ that are directly evicted, they will be reassigned to the waiting committee in other shards (line 17-18 & 31-35). Such cross-shard eviction is implemented to achieve a balanced distribution of reputation scores across shards. In contrast, the k_2 nodes in list $Evicted_2^{\#m}$ will be rotated to become waiting nodes within the same shard (line 19-20). This intra-shard eviction ensures the robustness and efficiency of the consensus committee.

Waiting Committee Election. Similar to eviction, the root shard utilizes the reputation scores of waiting nodes $Scores_{wn}^{\#m}$ within each shard as stakeholders to construct a Merkle tree for election. For each shard, the root shard then executes the FTS algorithm to elect k_1+k_2 waiting nodes as candidate consensus nodes for the subsequent epoch (line 22-25). Only after successfully completing the key negotiation can candidates to become consensus nodes. Failure to do so results in their demotion waiting nodes (line 26-30).

Leader Election. In the elected consensus committee, the root shard performs the FTS algorithm again to elect the leader. Through reputation-based node shuffling, Shardora ensures that the elected consensus committee and leader are robust.

Shard Resemblance. During node shuffling, k_1 nodes with the lowest recent epoch contribution will be directly evicted to other shards. Assume that there are m transaction shards in the system, then a total of $m \cdot k_1$ nodes will be evicted directly. For balanced reputation scores among all shards, the reputation scores of all these nodes will be arranged in descending order to generate a list, denoted as $Evicted_c$. Each element in $Evicted_c$ is then allocated in descending order to the shard with the lowest overall reputation score (line 31-35).

Ledger Synchronization. During node shuffling, the evicted consensus nodes will be reassigned to other shards as new arrival waiting nodes. These nodes are required to synchronize the historical ledger state of the new shard. Through line 36 in Algorithm 1, Shardora allows these nodes to request ledger synchronization from other old waiting nodes within the shard. The consensus nodes will uniformly ignore such requests to prevent them from hindering transaction processing.

2) Key Pre-negotiation: Shardora employs BLS threshold signatures-based BFT consensus protocol for intra-shard agreement. Unfortunately, threshold signatures inherently require signers to engage in key negotiation before signing, as depicted in Figure 1. Only with the negotiated keys can signature aggregation and verification be achieved. Consequently, each time the consensus committee changes (node shuffling), key negotiation must be repeated within the new committee, causing Zero-TPS issue as transactions cannot be processed during the negotiation.

To prevent key negotiation from interrupting transaction processing, Shardora employs a pre-negotiation mechanism. In the mechanism, the root shard pre-publishes the next epoch's candidate consensus committees for each shard, so that candidates can conduct key negotiation during the current epoch. In other words, during the current epoch, transaction processing runs in parallel with key negotiation for the following epoch. However, given Shardora's partial node shuffling implementation, most consensus nodes in the current epoch may still be elected as candidates for the consensus committee in the next epoch, as shown in Figure 2. During this epoch, these re-elected nodes must handle transaction processing while also engaging in key negotiation for the next epoch. Since key negotiation consumes significant bandwidth and computational resources, it reduces current transaction processing efficiency. To mitigate this impact, Shardora proposes a lightweight key negotiation mechanism that enables re-elected nodes to reuse some intermediate values from the preceding round of key negotiations. Specifically, our key negotiation mechanism consists of the following sub-phases.

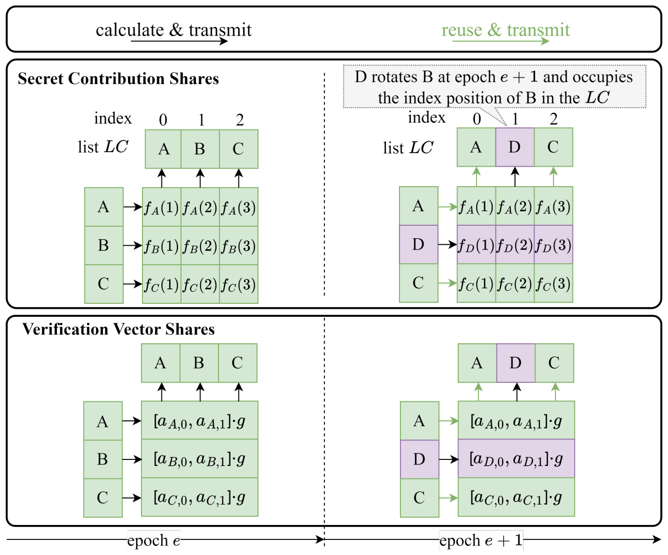

calculate & transmit
reuse & transmit
Secret Contribution Shares
D rotates B at epoch $e+1$ and occupies the index position of B in the LC
index 0 1 2
list LC
A B C
A D C
$f_A(1)$ $f_A(2)$ $f_A(3)$
$f_B(1)$ $f_B(2)$ $f_B(3)$
$f_C(1)$ $f_C(2)$ $f_C(3)$
$f_D(1)$ $f_D(2)$ $f_D(3)$
Verification Vector Shares
$[a_{A,0}, a_{A,1}]\cdot g$
$[a_{B,0}, a_{B,1}]\cdot g$
$[a_{C,0}, a_{C,1}]\cdot g$
$[a_{D,0}, a_{D,1}]\cdot g$
epoch e
epoch $e+1$

Fig. 3: Lightweight key negotiation mechanism. In epoch $e+1$, D rotates B and occupies the same index position as B in LC. The re-elected nodes {A, C} maintain their original index position in LC to reuse secret contribution shares and verification vector shares calculated in epoch e.

Parameter Settings. Assume that g is a base point on the elliptic curve BN128. Let $\mathbb{Z}_p$ be a finite fields, where p is a prime order on the elliptic curve BN128. Let G_1, G_2, and G_{T} be three multiplicative groups of prime order q, a bilinear map is denoted as $\mathbf{e} : G_1 \times G_2 \rightarrow G_T$. $h : (0,1)^* \rightarrow G_1$ expresses a hash function. Furthermore, let n_c denote the total number of nodes in the consensus committee. The threshold t represents the minimum number of participating nodes required to reach consensus, such that $t \geq \frac{2}{3}n_c+1$ for BFT. In addition, since the genesis block, there will be a list LC to record the candidate consensus nodes, where $|LC| = n_c$. Each candidate node in the list will take its index in LC as an identity number. To achieve a lightweight key negotiation process, re-elected consensus nodes are allowed to reuse the intermediate values employed in the prior epoch's key negotiation, as shown in Figure 3. To achieve this goal, LC will not be completely reordered after each election of a new consensus committee. Re-elected consensus nodes maintain their original index positions in LC, while newly added nodes will fill the vacant positions left by evicted nodes.

Polynomial Generation. Upon receiving the election block from the root shard, each node will know whether it is elected to join the next epoch's candidate consensus committee. For each newly elected node P_i, it will first obtain its identity number $id_i = index(P_i)$ in LC. The function $index(P_i)$ returns such a number associated with the node P_i. The index values are uniquely assigned to a newly elected node in LC for the vacant position in an arbitrary manner. Then it locally generates a random polynomial $f_i(x)$ of degree $t-1$ as

$$f_i(x) = \sum_{j=0}^{t-1} a_{i,j} x^j,$$

where $a_{i,j} \in \mathbb{Z}_p$, $t = \frac{2}{3}n_c + 1$. For consensus nodes who are re-elected to serve on the consensus committee in the upcoming epoch, they will retain their previous positions in LC. It enables them to reuse the polynomial $f(x)$ generated in the previous epoch.

Secret Sharing. Each candidate P_i then broadcasts its public verification vector V_i. V_i is generated by mapping $f_i(x)$'s coefficients to points on the elliptic curve BN128 as

$$V_i = [a_{i,0}, a_{i,1}, a_{i,2}, \dots, a_{i,t-1}] \cdot g.$$

Newly elected nodes are required to generate a verification vector V and broadcast it to all other $n_c - 1$ candidates. In contrast, re-elected nodes can reuse the V generated in the previous epoch and only distribute it to $k_1 + k_2$ new nodes. An example presented in Figure 3 demonstrates that, during epoch $e + 1$, the re-elected nodes A and C can reuse the verification vector calculated in epoch e. Additionally, candidates are further required to compute and distribute secret key contributions. Each candidate P_i will generate a secret contribution for other candidates P_j based on their polynomial. The secret contribution is computed as $s_i^j = f_i(id_j + 1)$. The newly elected node needs to calculate and distribute secret contributions for all n_c candidates (including itself), while the re-elected node only needs to distribute secret contributions for new nodes, and the contributions for all nodes can be reused. As shown in Figure 3, the re-elected nodes A and C can reuse all the secret shares computed in epoch e. This is because node B has been rotated to D, and D occupies the same index position in LC of epoch $e+1$ as B in LC of epoch e. For nodes A and C in epoch $e+1$, node D is regarded as a node that with the identity number 1. Hence, they transmit respective secret contributions $f_A(2)$ and $f_C(2)$ to D. These contributions are mathematically equivalent to the ones previously transmitted to node B in the preceding epoch.

Verification. With V_i, P_j can verify the received s_i^j through

$$s_i^j \cdot g = \sum_{k=0}^{t-1} (id_j + 1)^k \cdot V_i[k]. \tag{1}$$

If equation (1) holds for $t = \frac{2}{3}n_c + 1$, it ensures that secret contribution s_i^j is correct and consistent with $f_i(x)$.

Public Key Calculation. Upon receiving $\frac{2}{3}n_c + 1$ correct secret contributions, candidates generate a secret contribution list L_s corresponding to LC locally, which is then disseminated to all peers. A candidate's secret contribution is considered valid only if it corresponds with the contributions of over two-thirds of the candidates occupying the same position. Every candidate incorporates the first element of the verification vector from each valid candidate to construct the system public key, represented as $PK = \sum_{j=0}^{n_c-1} V_j[0]$, where V_j is derived from a valid P_j. If P_j is deemed invalid, then $V_j[0] = null$. Candidates excluded from this process will degenerate into waiting nodes.

Secret Key Reconstruction. Meanwhile, the secret sharing S_i of each valid candidate P_i is computed as $S_i = \sum_{j=0}^{n_c-1} s_j^i$. It is computed by aggregating all the secret contributions received from all valid candidates P_j. To further reconstruct the collective secret key SK, at least $t = \frac{2}{3}n_c + 1$ valid participants need to collaborate and do Lagrange interpolation. Then SK is computed as $SK = \sum_{i=1}^{t} S_i \cdot \lambda_{i+1}$, where $\lambda_i = \prod_{j=1}^{t, j \neq id_i} \frac{j}{j - id_i}$ are the Lagrange coefficients. Upon completing the key negotiation process, all valid candidates transition into consensus nodes, forming the consensus committee for the subsequent epoch. They leverage their secret shares S_i for signing. The elected leader is tasked with aggregating all valid signatures received. Provided that over two-thirds of the nodes have correctly contributed their signatures, it becomes feasible to aggregate these into a signature signed by the collective secret key SK. All nodes are capable of verifying this aggregated signature through the system public key PK.

3) Consensus Agreement: This section presents the consensus protocols employed in Shardora. We discuss intra-shard and cross-shard consensus protocols separately. The intra-shard consensus is responsible for transactions within one shard, while the cross-shard consensus handles transactions across multiple shards.

Intra-shard Consensus. It is achieved in each shard through a specific protocol during any given epoch e. Shardora employs BFT consensus protocol as its intra-shard consensus protocol, supplemented by well-established BLS threshold signature schemes. Threshold signatures necessitate that signers negotiate relevant keys before signing. Consequently, Shardora completes the necessary key negotiation for epoch e during the preceding epoch $e - 1$, as outlined in the previous section. With the keys obtained, consensus nodes can perform signing and verification to implement the BFT consensus protocols, ultimately reaching an agreement on a valid block. The specific process is as follows:

The leader of the consensus committee proposes a block containing a batch of valid transactions. Following the proposal, the leader broadcasts a pre-vote message to all other consensus nodes. This message includes the proposed block and its metadata. Upon receiving the pre-vote message, other consensus nodes undertake the validation of the proposal. If successful validation, these nodes utilize their respective secret shares S_i to sign the pre-vote message m_{pv} as $\sigma_{pv,i} \leftarrow h(m_{pv}) \cdot S_i$. Then they relay the signature back to the leader as an endorsement of the proposed block. A block is deemed ready for pre-commit once a supermajority (typically defined as two-thirds) of the consensus committee members have sent their valid pre-vote messages, confirming consensus on the proposed block. Through Lagrange interpolation, all valid signatures on the pre-vote message can be aggregated into an entire signature σ_{pv} as $\sigma_{pv} \leftarrow \sum_{i=1}^{t} \sigma_{pv,i} \cdot \lambda_i$. This σ_{pv} will then be incorporated into the pre-commit message as metadata. Only the $\mathbf{e}(g, \sigma_{pv}) = \mathbf{e}(PK, h(m_{pv}))$ holds can indicate that over two-thirds of the consensus committee members have confirmed the pre-vote message. Once the verification is successful, they will sign the pre-commit message again using their own secret share S_i, similar to the pre-vote phase. After all three phases of sign and verification (pre-vote, pre-commit, and commit), the leader can finalize the block and commit it into the shard's ledger. After the block confirmation, the leader randomly rotates to ensure that each consensus node has the opportunity to propose a block.

Cross-shard Consensus. Cross-shard transactions are inherently more complex to process than in-shard transactions, necessitating consensus across all involved shards to ensure consistency. Similar to [15], [31], Shardora decomposes a cross-shard transaction into two intra-shard transactions. For example, we depict a cross-shard transaction TX_{cs}, transferring an amount φ from account a in shard #1 to account b in shard #2, as follows:

$$TX_{cs} = \langle \{\#1, a\}, \{\#2, b\}, \varphi \rangle .$$

TX_{cs} can be represented as two in-shard transactions TX_{is} and conducted in respective shards in two-phase:

$$TX_{is}^{\#1} = \langle \rho, a, \varphi \rangle , TX_{is}^{\#2} = \langle \phi, b, \varphi, \gamma \rangle .$$

Phase 1: $TX_{is}^{\#1}$ is processed in shard #1 to perform withdraw operation ρ from account a. Following the confirmation of $TX_{is}^{\#1}$, a proof of acceptance, denoted as γ, is generated as

$$\gamma = \left\langle \{\sigma_c, PK\}_e , TX_{is}^{\#1}, H_{header}, H_{TXs} \right\rangle ,$$

where σ_c represents the aggregated signature on the commit message that finalizes the block containing $TX_{is}^{\#1}$. Therefore, $\{\sigma_c, PK\}_e$ are utilized to validate σ_c from any epoch e. In addition, H_{header} is the hash of the block including transaction $TX_{is}^{\#1}$, and H_{TXs} denotes Merkle proof of account a.

Phase 2: $TX_{is}^{\#2}$ is processed in shard #2 to perform deposit operation ϕ to account b. With γ, consensus nodes in shard #2 can confirm that $TX_{is}^{\#1}$ has been committed to the ledger of shard #1, thereby ensuring the atomicity of TX_{cs}.

4) Reputation Scoring: Shardora employs reputation scores to evaluate the reliability and capability of nodes. Reliability indicates the nodes' adherence to honest behavior, while capability measures the number and complexity of creating valid transaction blocks when they serve as leaders. In our reputation model, reliability is evaluated based on the nodes' stake, similar to the Proof of Stake (PoS) mechanism. Capability is assessed by the nodes' accumulative contributions to block creation, like literature [16], [17]. For each epoch, the root shard updates the reputation scores across all shards. The reputation score Rep_i^e for node i in epoch e is calculated as

$$\text{Rep}_i^e = \frac{St_i^e + \sum_{k=t}^{e-1} C_i^k}{2}, \tag{2}$$

where St_i^e denotes the accumulative stake of node i. C_i^k denotes the contribution made by node i in epoch k. Let t be the epoch at which node i joins the system. The reputation score Rep_i^e for node i aggregates the scores accumulated up to the most recent epoch $e-1$ since its initial participation. The calculation of St_i^e and C_i^k are introduced in detail below.

Stake St_i^e. Shardora requires that each node i joining the system must transfer a certain tokens to the public account as its initial stake St_i^t. At any subsequent epoch k, users can pledge additional stakes δSt_i^k, which will be accumulated into Rep_i^{k+1} at the start of epoch $k+1$, thus increasing the probability of election. Furthermore, block rewards R_i^k earned during mining in each epoch k are automatically deposited to the public account as part of the node's stake. Then the accumulated stake up to epoch e is calculated as $St_i^e = St_i^t + \sum_{k=t+1}^{e-1} \left(\delta St_i^k + R_i^k\right)$. Importantly, the stake may only be transferred to the node's account upon its departure with no detected malicious behaviors. This design aims to increase the economic costs of attacks, where any malicious behaviors result in the lose of all accumulated stakes. Conversely, for honest nodes, this design increases their staked amounts and potentially enhance their gains over time. In addition, to prevent the stake domination caused by the rich, Shardora smooths St_i^e and limits it to [0,100]. For normalized reputation score in equation (2), C_i^k will also be mapped to this range.

Block Creation Contribution C_i^k. The block creation contributions represent the valid transaction contributions confirmed by each node when serving as a leader in each epoch. Shardora specifies that within an epoch, once a leader successfully submits a valid block, it will automatically switch to the next leader, ensuring that every consensus node has the opportunity to serve as a leader. Followers, upon verifying the validity of a block, locally statistic contribution of the block proposed by the leader. When one of the followers transitions to the next leader, it will include a transaction in the proposed block to agree on the contribution of the previous leader. The contribution C_i^k of leader i in epoch k is calculated as

$$C_i^k = \sum_{x=1}^{n_i^k} \left(\frac{\sum_{j=1}^{l_x} \xi_{i,j}^x G_j^x}{\sum_{j=1}^{l_x} G_j^x} \right),$$

where n_i^k is the number of blocks that node i proposed in epoch k when it serves as a leader. l_x denotes the number of valid transactions in the x-th proposed block. G_j^x specifies the gas cost for evaluating the value of transaction j in the x-th proposed block. An reward-penalty coefficient $\xi_{i,j}^x$ is employed to either reward or penalize different behaviors of leader i on transaction j in the x-th proposed block, assigning a reputation score of {1, 0, -1} to represent correct, unknown and incorrect behaviors, respectively.

IV. Security Analysis

This section will discuss the security properties of Shardora in terms of system security and key negotiation security.

A. System Security Analysis

The security of Shardora relies on two probabilistic selection processes: first, select n nodes from a set of N nodes to form a shard, and then select n_c nodes from these n nodes to establish a consensus committee. Let X_1 and X_2 represent the number of malicious nodes in the selected shard and consensus committee, respectively. With the requirement of BFT, only when $X_2 \geq \lceil n_c/3 \rceil$, the consensus committee becomes vulnerable to result in system failure.

Claim 1: *Compared to random-based selection, reputation-based selection offers higher security. What's more, the probability that the number of malicious nodes among the n_c final selected nodes exceeds $n_c/3$ is negligible.*

Proof: In the first round of selection, let the probability of choosing a malicious node be p_{1b} and the probability of choosing an honest node be p_{1h}, where $p_{1h} = 1 - p_{1b}$. The number of malicious nodes chosen, X_1, approximately follows a binomial distribution, i.e., $X_1 \sim Binomial(n, p_{1b})$. Then, in the second round of selection, n_c nodes are further selected from the n nodes. Let the probability of choosing a malicious node in the second round be p_{2b}, the probability of choosing an honest node be p_{2h}, where $p_{2h} = 1 - p_{2b}$. Conditioned on $X_1 = k$, the number of malicious nodes selected in the second round, X_2, approximately follows a binomial distribution: $X_2|X_1 = k \sim Binomial(n_c, p_{2b})$. System failure occurs when $X_2 \geq \lceil n_c/3 \rceil$, such probability can be expressed as:

$$P(X_2 \geq \frac{n_c}{3}) = \sum_{k=0}^{n} P(X_1 = k) \times P(X_2 \geq \frac{n_c}{3}|X_1 = k),$$

where,

$$P\left(X_1 = k\right) = \binom{n}{k} {p_{1b}}^k (1 - p_{1b})^{n-k},$$

$$P(X_2 \geq \frac{n_c}{3}|X_1 = k) = \sum_{l=\lceil \frac{n_c}{3} \rceil}^{n_c} \binom{n_c}{l} {p_{2b}}^l (1 - p_{2b})^{n_c - l}.$$

For random-based selection, the probability that a malicious node is selected in both rounds is equal, denoted as $p_{1b}^{\text{rand}} = p_{2b}^{\text{rand}} = \frac{F}{N}$. For reputation-based selection, $p_{1b}^{\text{rep}} = \frac{F\text{rep}_b}{F\text{rep}_b + (N-F)\text{rep}_h}$, where $\text{rep}_b, \text{rep}_h$ denote the reputation of Byzantine nodes and honest nodes, respectively. Regardless of whether it is a *simple attack* or *camouflage attack*, it holds that $\text{rep}_b \leq \text{rep}_h$. Thus, it can be proved that $p_{1b}^{\text{rep}} \leq \frac{F}{N} = p_{1b}^{\text{rand}}$. Similarly, $p_{2b}^{\text{rep}} = \frac{E[X_1^{\text{rep}}]\text{rep}_b}{E[X_1^{\text{rep}}]\text{rep}_b + (n - E[X_1^{\text{rep}}])\text{rep}_h} \leq \frac{E[X_1^{\text{rep}}]}{n} \leq \frac{F}{N} = p_{2b}^{\text{rand}}$, where $E[X_1^{\text{rep}}]$ represents the expected number of malicious nodes chosen in the first round, computed as $E[X_1] = n \times \frac{Fp_{1b}^{\text{rep}}}{Fp_{1b}^{\text{rep}} + (N-F)p_{1h}^{\text{rep}}}$. Therefore, it can be proved that $P(X_2^{\text{rep}} \geq \frac{n_c}{3}) \leq P(X_2^{\text{rand}} \geq \frac{n_c}{3})$. Following the assumption in most public blockchain system, we consider a system fault tolerance of $1/4$ (set N=2000, F=500). Additionally, given $n = 600$, $n_c = 300$, $p_{1b}^{\text{rep}} = 0.15$ (0.25 for random), the Shardora system's failure probability is calculated to be 6.87×10^{-16}. It is negligible and significantly lower than the failure probability for random-based selection under the same conditions, which is 4.84×10^{-4}. Even in the worst-case scenario, where all adversaries collude to launch a camouflage attack, the system's failure probability only reduces to a random-based level of 10^{-4}, maintaining a high security level. Moreover, it is highly challenging for nodes to collude in a distributed environment. □

B. Key Negotiation Security Analysis

1) Key Negotiation Security: When malicious nodes are elected as candidates for the upcoming consensus committee, they are required to jointly negotiate the keys for threshold signatures with other candidates through DKG. During this process, malicious candidates may refuse to cooperate with other participants or deliberately provide incorrect information to obstruct the key generation process.

Claim 2: *Even if all malicious candidates refuse to cooperate or provide incorrect information, it will not obstruct the key generation process.*

Proof: The DKG specifies that as long as more than the threshold number t of nodes provide the correct secret values, the complete key can be recovered. Assumed that there are f_c malicious nodes in the consensus committee with n_c nodes. To keep the liveness of key negotiation, the threshold value should satisfy $t \leq n_c - f_c$. With the requirement of BFT, Shardora can guarantee that each consensus committee satisfies $n_c \geq 3f_c + 1$ (defined in Section II-D). Therefore, only when $t \leq 2f_c + 1$ is satisfied can $t \leq n_c - f_c$ hold. In addition, to keep safety, at least one honest node is required to participate in key negotiation, hence t should satisfy $t \geq f_c + 1$. Consequently, when $f_c + 1 \leq t \leq 2f_c + 1$, we can ensure the success of the key negotiation process. □

2) Key Negotiation Value Reuse Security: DKG ensures the randomness of each node's polynomial to maintain security, preventing replay attacks and collusion by malicious nodes from recovering the aggregate polynomial. In contrast, during consecutive and independent DKG processes across epochs, Shardora allows unshuffled nodes to reuse most of their key negotiation value for mitigating the TPS-Degradation issue.

Claim 3: *Key negotiation value reuse by unshuffled nodes does not compromise the security of Shardora, as the aggregate polynomial has changed.*

Proof: The security of DKG primarily relies on the randomness of the aggregate polynomial formed by honest nodes. This randomness ensures the aggregate polynomial is unpredictable, guaranteeing system security. Shardora maintains security since it allows key negotiation value reuse while ensuring the aggregate polynomial changes randomly during independent DKG processes across epochs. In each new epoch, Shardora partially rotates the consensus committee ($k_1 + k_2$ nodes). As analyzed in Section IV-A, the probability of having $f \geq \frac{1}{3}n_c$ malicious nodes in a consensus committee of size n_c is negligible. Since Shardora requires $k_1 + k_2 \geq \frac{1}{3}n_c + 1$, at least $f + 1$ nodes are rotated in each epoch. According to BFT security criterion, this guarantees the inclusion of at least one new honest node in the consensus committee candidate after each rotation. Although unshuffled nodes reuse their key negotiation values, the inclusion of new honest nodes ensures that the aggregate polynomial generated in the new epoch changes, therby maintaining system security.

□

V. Performance Evaluation

In this section, we evaluate the performance of our shardora by local experimental comparison and cloud testbed deployment. The experimental setup is outlined as follows:

Local experiment setup. We evaluate the performance of Shardora against state-of-the-art schemes across several metrics to highlight its distinctive features. The evaluation of state-of-the-art schemes is implemented based on our codebase.

Local experiments are conducted on a laptop with Intel(R) Core(TM) i7-1165G7 CPU @2.8GHz (4 cores), 16GB RAM. The experimental setup involves 2 transaction shards, each comprising 5 nodes. Each block contains 1000 transactions, with an average transaction size of around 200 bytes.

Cloud testbed deployment setup. We deployed the Shardora prototype on 100 virtual servers rented from Alibaba Cloud to evaluate the system's performance in real-world networks. Each server is equipped with CentOS Linux 7 Core, an Intel® Xeon® Platinum 8163 CPU @ 2.5GHz (8 cores), and 16GB RAM, with the network bandwidth configured to 3Gbps. Notably, this experimental setup allowed for initiating up to 600 nodes with Shardora deployed.

A. Performance Comparison

1) Ledger Synchronization Overhead: We evaluate the data size synchronized by consensus nodes to newly joined nodes under different shard reconfiguration schemes. Under the local experiment setup, we further set the reconfiguration interval to 100 blocks. The results are shown in Figure 4(a). It can be observed that synchronized data size in RapidChain [14] linearly increases with the proposed number of blocks since newly arrived nodes need to synchronize the entire ledger. In contrast, tMPT [15] maintains a relatively stable data size by synchronizing only the states of active accounts. Similarly to RapidChain, our Shardora also requires the newly arrived nodes to download the entire ledger. However, Figure 4(a) shows that Shardora incurs negligible data synchronization overhead (approximately 200KB) for consensus nodes during the ledger synchronization. Such overhead is reduced by $50\times$ than [15] and $1000\times$ than [14]. This is because the dual committee design, Shardora enables consensus nodes to synchronize only the latest block, while new arrival nodes can retrieve the historical ledger from waiting nodes.

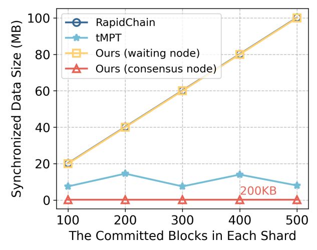
2) Key Negotiation Overhead: To evaluate the communication and computation overhead of nodes during key negotiation, we set consensus committee sizes as 100, 256, 512, 600, 800 and 1024, with a node shuffling ratio of 10%. The experimental results, illustrated in Figure 4(b) and 4(c), highlight two key findings: (1) Shardora exhibits significantly enhanced performance compared to RepShard [16] and RepChain [17], reducing communication and computation overhead by at least 90%. This efficiency gain stems from Shardora's ability to allow nodes to reuse data from prior epochs of key negotiation, thereby eliminating the need for redundant computations and data transmissions required by RepShard and RepChain when the consensus committee changes. (2) Furthermore, the computation and communication overhead is lower on re-elected consensus nodes than on the newly elected waiting nodes. In Shardora, newly elected waiting nodes must compute and broadcast their verification vectors and secret shares to all network nodes, while re-elected consensus nodes only transmit secret shares to the newly elected nodes and verify their correctness. With a consensus committee size of 1024, a re-elected consensus node consumes approximately about 8MB of bandwidth and 17ms in key negotiation time, which causes a minor impact on its transaction processing efficiency.

3) Throughput During Shard Reconfiguration: To demonstrate the capability of Shardora for seamless and cost-effective shard reconfiguration, we configure the system epoch time to 5 minutes. It means that the system automatically performs shard reconfiguration every 5 minutes. Then we measure the system throughput over two epoch periods, and the comparison results among different schemes are presented in Figure 4(d). During shard reconfiguration, our Shardora maintains a stable throughput. The difference in highest and lowest throughput remains within 400 TPS. While throughput in other schemes is affected by the shard reconfiguration. Elastico [12] experiences zero-TPS during the reconfiguration phase due to the complete rotation. Consensus nodes in RapidChain [14] need to allocate significant resources to solve the offline PoW puzzle. Both RapidChain and tMPT [15] require consensus nodes to perform ledger synchronization.

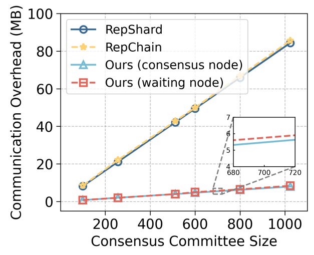
B. Cloud Testbed Evaluation of Shardora

1) System Throughput Stress Test: To conduct a stress test on the system's throughput, we vary the transaction arrival rate to determine the peak throughput that the system can sustain within 600 nodes under different number of shards S and consensus committee size n_c. The results are shown in Figure 5(a). At a consensus committee size of $n_c = 600$, the throughput of a single shard can peak at 8326 transactions per second (TPS). The system's overall throughput surpass 50,000 TPS when $n_c = 100$ and $S = 6$.

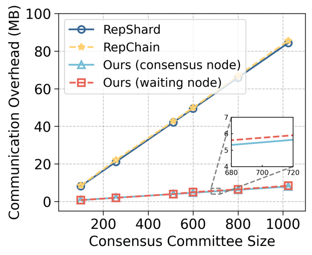
2) Throughput and TX Confirmation Latency: In this section, we measure the system throughput and transaction confirmation latency across various numbers of shards, consensus committee sizes, and transaction arrival rates. Figure 5(b) illustrates the impact of the consensus committee size on the throughput and transaction confirmation latency. Transaction confirmation latency rises with the increase of consensus committee size due to the need for more nodes to validate and agree on each transaction. Meanwhile, throughput decreases since the increased communication overhead brought about by a larger consensus committee, thereby affecting the transaction processing speed. Figure 5(c) demonstrates the positive effects of increasing the number of shards on throughput and transaction confirmation latency. As the number of shards increases, the number of transactions that each shard needs to process decreases, potentially accelerating the consensus speed within individual shards, and reducing transaction confirmation latency. Since sharding allows for parallel transaction processing in the blockchain network, the overall throughput increases. This finding is consistent with the results shown in Figure 5(a). But the total throughput in Figure 5(c) remains below 2000 TPS, not exceeding the transaction arrival rate. Figure 5(d) depicts the impact of transaction arrival rates on throughput and transaction confirmation latency. Given that the transaction arrival rates have not reached the system's transaction processing limit, throughput increases as transaction arrival rates rise. However, as more transactions need to be



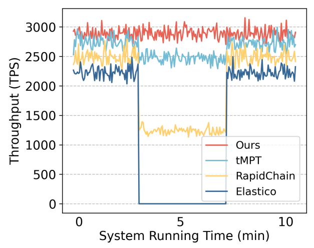
(a) Bandwidth consumption on consensus nodes for ledger sync.



(b) Communication overhead for key negotiation.

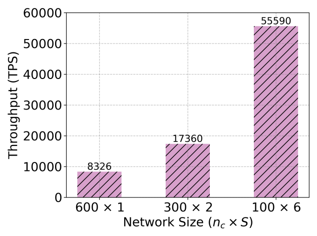


(c) Computation overhead for key negotiation.

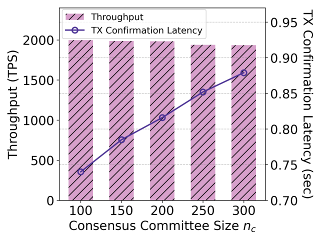


(d) Throughput at a transaction arrival rate of 3000 TXs/sec.

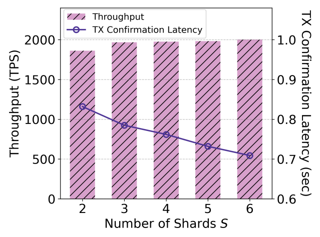
Fig. 4: Comparison of bandwidth consumption, communication overhead, computation overhead, and throughput during shard reconfiguration.



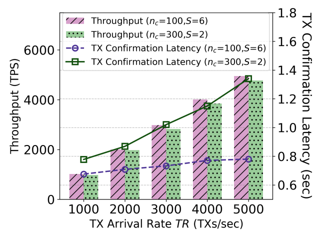
(a) Throughput stress test.



(b) TR=2000 TXs/sec, S=2.

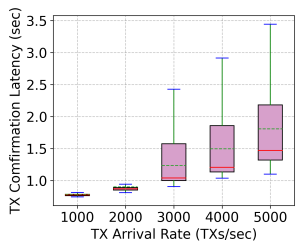


(c) TR=2000 TXs/sec, n_c=100.

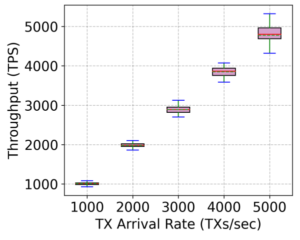


(d) n_c=100, S=6 and n_c=300, S=2.

Fig. 5: Our performance evaluation in a LAN. We deploy Shardora prototype on 100 servers (initiate 600 nodes) rented from Alibaba Cloud to evaluate our throughput and transaction confirmation latency under different conditions: number of shards, consensus committee size, and transaction arrival rates.

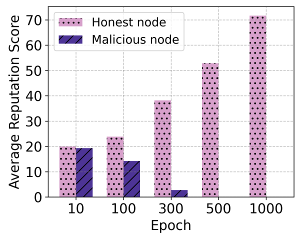


(a) TX confirmation latency.

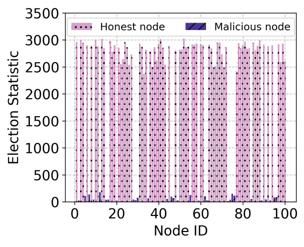


(b) Throughput.

Fig. 6: Transaction confirmation latency and throughput distribution at n_c=300 and n=2 network scale when TR is varied from 1000 to 5000 TXs/sec.



(a) Reputation score comparison over different epochs.



(b) Consensus committee election statistics over 5000 epochs.

Fig. 7: Reputation score and election statistic.

processed, the transaction confirmation latency also increases. Moreover, Figure 5(d) indicates that at a constant network scale of 600 nodes, a smaller consensus committee size and a larger number of shards typically lead to higher throughput and shorter transaction confirmation latency, particularly as transaction arrival rates ascend.

We further measure the system throughput and transaction confirmation latency over two epoch periods, and the experimental results are presented in Figure 6, where the upper and lower bounds represent the maximum and minimum values, respectively. The green dashed line represents the mean and the red solid line represents the median. As shown in Figure 6(a), when the transaction arrival rate is 2000, the difference between the maximum and minimum values of transaction confirmation delay does not exceed 0.3 seconds, and the mean and median values are almost equal. This indicates that the transaction confirmation latency of Shardora remains relatively stable during shard reconfiguration. Although the volatility of transaction confirmation latency increases as the transaction arrival rate increases. It is caused by queuing in the transaction pool rather than reconfiguration. Figure 6(b) further demonstrates the seamless reconfiguration process of Shardora, as it maintains a stable TPS regardless of the transaction arrival rate. Furthermore, the small TPS spread and roughly equal mean and median values in Figure 6(b) further support the stability of the reconfiguration process.

3) Reputation Statistic: Figure 7(a) compares the average reputation scores of static malicious nodes and honest nodes across different epochs. As epochs progress, the average reputation of honest nodes gradually rises, whereas it declines for malicious nodes, approaching zero by the 500th epoch. The reputation gap between malicious and honest nodes is also reflected in the selection frequency for consensus nodes. Figure 7(b) reveals that, after 5000 epochs, honest nodes are

TABLE I: Comparison on existing blockchain sharding schemes. The ledger synchronization overhead on consensus nodes ranges from zero, denoted as ○, to full synchronization, denoted as ●. A checkmark ✓ indicates the fulfillment of the property, while a cross × indicates the lack of fulfillment. The abbreviation "N/A" denotes that the status of the property remains unknown.

Scheme	Shard Reconfiguration	Ledger Synchronization	Reconfiguration Interval	Intra-Shard Consensus	Consensus Complexity	Reconfiguration System Availability	Heterogeneity	Incentives
Elastico [12]	Full	●	one-day long	PBFT	$O(n^2)$	×	×	×
RepShard [16]	Full	●	N/A	BFT+BLS	$O(n)$	×	✓	×
RepChain [17]	Full	●	N/A	CSBFT	$O(n^2)$	×	✓	✓
Omniledger [13]	Partial	◕	one-day long	PBFT	$O(n^2)$	✓	×	×
RapidChain [14]	Partial	●	one-day long	PBFT	$O(n^2)$	✓	×	×
SSRR [34]	Partial	●	N/A	BFT	$O(n^2)$	✓	×	×
tMPT [15]	Partial	◔	100 blocks	PBFT	$O(n^2)$	✓	×	×
SkyChain [35]	Partial	●	(0,1000s)	BFT	$O(n^2)$	✓	×	×
Cuckchain [36]	Partial	●	N/A	PBFT	$O(n^2)$	✓	✓	✓
ECFR [37]	Partial	●	N/A	PBFT	$O(n^2)$	✓	✓	✓
Our Shardora	**Partial**	◷	**10 min (Default)**	**BFT+BLS**	$O(n)$	✓	✓	✓

chosen more frequently as consensus nodes than malicious nodes. These findings demonstrate that Shardora becomes increasingly robust as system runtime extends.

VI. Related Work

In this section, we will first discuss the shard reconfiguration mechanism in existing blockchain sharding schemes, then introduce their respective in-shard consensus protocols to reveal the motivation of our work. Comparisons of the existing schemes for sharding reconfiguration is listed in TABLE I.

Blockchain sharding schemes with reconfiguration. As the first sharding protocol with presence of Byzantine adversaries, Elastico [12] enables each shard to process transactions in parallel, thereby improving scalability. But it requires all previously participating nodes to recompute a PoW puzzle during the reconfiguration phase. This requirement inevitably results in a temporary loss of system availability. To ensure continuous availability, other solutions such as Omniledger [13], RapidChain [14], tMPT [15], SkyChain [35], etc., adopt partial node shuffling during the reconfiguration phase. Omniledger shuffles only a small portion ($\leq 30\%$) of the nodes in each shard, allowing most nodes to remain in their original shards and continue processing transactions. In RapidChain, the bounded Cuckoo rule is used to strategically shuffle a small number of nodes. However for existing schemes that support partial node shuffling, when a node is shuffled to a new shard, it needs to download and synchronize the complete ledger of that shard from the remaining nodes (which work on transaction processing in this period). This synchronization consumes significant bandwidth and computational resources on these nodes, hampering the system throughput. Ominiledger and tMPT mitigate synchronization overhead by compressing ledger information, while SkyChain utilizes reinforcement learning to dynamically adjust the reconfiguration interval, minimizing the impact on throughput. As public blockchain sharding systems, [12]–[15], [34], [35] ignore the heterogeneity among participating nodes in terms of computing resources, network bandwidth, and historical behaviors. To mitigate potential bottlenecks caused by less capable nodes and improve system throughput, nodes are reputation scored in [16], [17], [36], [37]. These scores help in the strategic shuffling of nodes to maintain a balanced distribution of competent nodes across all shards. Furthermore, [17], [34], [36] introduce an incentive mechanism to reward consensus nodes involved in transaction processing.

Intra-shard consensus protocols of blockchain sharding. Elastico [12] was the first to adopt the PBFT protocol for handling transactions within each shard. Subsequent works further leveraged PBFT for intra-shard consensus [13]–[15], [34], [36]. PBFT is well-suited for dynamic sharding systems, as nodes joining the network can establish their identity using a public-private key pair and seamlessly sign and verify transactions with their own key, regardless of future shard reassignment. However, PBFT exhibits a communication complexity of $O(n^2)$ per block within a shard of n nodes, which escalates to $O(n^3)$ during the view-change protocol [18]. This high communication complexity limits optimal system throughput. To address this, RepShard [16] and RepChain [17] incorporated BLS threshold signatures into the BFT protocol, reducing the communication complexity to $O(n)$. Nevertheless, the use of threshold signatures requires the consensus nodes in newly formed shards to renegotiate keys during each shard reconfiguration. This results in the system being temporarily unavailable, as transactions cannot be processed during the negotiation process.

VII. Conclusion

In this paper, we propose Shardora, a blockchain sharding system designed to achieve high system throughput and security. By implementing a reputation-based dual committee node shuffling strategy, we reduce the TPS-Degradation issue during reconfiguration phases. Furthermore, Shardora incorporates a lightweight decentralized key pre-negotiation mechanism to avoid Zero-TPS issue during key negotiation periods. We systematically prove the security of Shardora through rigorous tests and analyses. We develop a prototype of Shardora and make our source code available on GitHub. To evaluate performance, we deploy our Shardora prototype on Alibaba Cloud. Experimental results have demonstrated that the proposed Shardora outperforms other baselines in terms of ledger synchronization overhead, key negotiation overhead, and throughput stability during reconfiguration.

References

[1] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, “Blockchain technology and its relationships to sustainable supply chain management,” *International journal of production research*, vol. 57, no. 7, pp. 2117–2135, 2019.

[2] P. Dutta, T.-M. Choi, S. Somani, and R. Butala, “Blockchain technology in supply chain operations: Applications, challenges and research opportunities,” *Transportation research part e: Logistics and transportation review*, vol. 142, p. 102067, 2020.

[3] P. Treleaven, R. G. Brown, and D. Yang, “Blockchain technology in finance,” *Computer*, vol. 50, no. 9, pp. 14–17, 2017.

[4] B. Scott, “How can cryptocurrency and blockchain technology play a role in building social and solidarity finance?” UNRISD Working Paper, Tech. Rep., 2016.

[5] H. M. Hussien, S. M. Yasin, N. I. Udzir, M. I. H. Ninggal, and S. Salman, “Blockchain technology in the healthcare industry: Trends and opportunities,” *Journal of Industrial Information Integration*, vol. 22, p. 100217, 2021.

[6] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, “Blockchain in healthcare applications: Research challenges and opportunities,” *Journal of network and computer applications*, vol. 135, pp. 62–75, 2019.

[7] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[8] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[9] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, “A survey on the scalability of blockchain systems,” *IEEE network*, vol. 33, no. 5, pp. 166–173, 2019.

[10] G. Wang, Z. J. Shi, M. Nixon, and S. Han, “Sok: Sharding on blockchain,” in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 2019, pp. 41–61.

[11] A. I. Sanka and R. C. Cheung, “A systematic review of blockchain scalability: Issues, solutions, analysis and future research,” *Journal of Network and Computer Applications*, vol. 195, p. 103232, 2021.

[12] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 17–30.

[13] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “Omniledger: A secure, scale-out, decentralized ledger via sharding,” in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 583–598.

[14] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: Scaling blockchain via full sharding,” in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. ACM, 2018, pp. 931–948.

[15] H. Huang, Y. Zhao, and Z. Zheng, “tmpt: Reconfiguration across blockchain shards via trimmed merkle patricia trie,” in *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. IEEE, 2023, pp. 1–10.

[16] G. Wang, “Repshard: reputation-based sharding scheme achieves linearly scaling efficiency and security simultaneously,” in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 237–246.

[17] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, “Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2020.

[18] M. Castro and B. Liskov, “Practical byzantine fault tolerance and proactive recovery,” *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[19] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, “Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding,” in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1968–1977.

[20] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, “Randomized rumor spreading,” in *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000, pp. 565–574.

[21] 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.

[22] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, “{Bitcoin-NG}: A scalable blockchain protocol,” in *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, 2016, pp. 45–59.

[23] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing bitcoin security and performance with strong consistency via collective signing,” in *25th usenix security symposium (usenix security 16)*, 2016, pp. 279–296.

[24] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, “Aggregatable distributed key generation,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 147–176.

[25] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, “Practical asynchronous distributed key generation,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2518–2534.

[26] S. Das, Z. Xiang, L. Kokoris-Kogias, and L. Ren, “Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5359–5376.

[27] H. Zhang, S. Duan, C. Liu, B. Zhao, X. Meng, S. Liu, Y. Yu, F. Zhang, and L. Zhu, “Practical asynchronous distributed key generation: Improved efficiency, weaker assumption, and standard model,” in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2023, pp. 568–581.

[28] R. Bacho and J. Loss, “On the adaptive security of the threshold bls signature scheme,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 193–207.

[29] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. G. Gueta, and S. Devadas, “Towards scalable threshold cryptosystems,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 877–893.

[30] M. Bellare, E. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu, “Better than advertised security for non-interactive threshold signatures,” in *Annual International Cryptology Conference*. Springer, 2022, pp. 517–550.

[31] J. Wang and H. Wang, “Monoxide: Scale out blockchains with asynchronous consensus zones,” in *16th USENIX symposium on networked systems design and implementation (NSDI 19)*. USENIX, 2019, pp. 95–112.

[32] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, “Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract] y,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014.

[33] P. Feldman, “A practical scheme for non-interactive verifiable secret sharing,” in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 1987, pp. 427–438.

[34] Y. Liu, J. Liu, Y. Hei, W. Tan, and Q. Wu, “A secure shard reconfiguration protocol for sharding blockchains without a randomness,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2020, pp. 1012–1019.

[35] J. Zhang, Z. Hong, X. Qiu, Y. Zhan, S. Guo, and W. Chen, “Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system,” in *Proceedings of the 49th International Conference on Parallel Processing*. ACM, 2020, pp. 1–11.

[36] J. Tian, C. Jing, and J. Tian, “Cuckchain: A cuckoo rule based secure, high incentive and low latency blockchain system via sharding,” in *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2023, pp. 1228–1234.

[37] M. Zhang, J. Li, Z. Chen, H. Chen, and X. Deng, “An efficient and robust committee structure for sharding blockchain,” *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2562–2574, 2023.