RepShard: Reputation-based Sharding Scheme Achieves Linearly Scaling Efficiency and Security Simultaneously

Gang Wang
University of Connecticut
Email: g.wang.china86@gmail.com

***Abstract*—Sharding technology is becoming a promising candidate to address the scalability issues in blockchain. The key concept behind sharding technology is to partition the network status into multiple distinct smaller committees, each of which handles a disjoint set of transactions to leverage its capability of parallel processing. However, when introducing sharding technology to blockchain, several key challenges need to be resolved, such as security and heterogeneity among the participating nodes. This paper introduces *RepShard*, a reputation-based blockchain sharding scheme that aims to achieve both linearly scaling efficiency and system security simultaneously. RepShard adopts a two-layer hierarchical chain structure, consisting of a reputation chain and independent transaction chains. Each transaction chain is maintained within its shard to record transactions, while the reputation chain is maintained by all shards to update the reputation score of each participating node. We leverage a novel reputation scheme to record each participating node's integrated and valid contribution to the system, in which we consider the heterogeneity of participating nodes (e.g., computational resources). The reputation score used in sharding and leader election processes maintains the balance and security of each shard. RepShard relies on verifiable relay transactions for cross-shard transactions to ensure consistency between distinct shards. By integrating reputation into the sharding protocol, our scheme can offer both scalability and security at the same time.**

I. Introduction

Blockchain functions as a decentralized and immutable ledger that facilitates transparent and auditable management of data and transactions. It shows enormous potential to revolutionize and advance existing financial ecosystems [1] [2]. As blockchain technologies mature, different variants of distributed ledgers emerge, each with its own strengths and weaknesses. Most current blockchain schemes are still far from achieving a high throughput (e.g., at the levels of VISA processing [3]) among a huge amount of participating nodes. A straightforward and effective solution to achieve a "scale-out" blockchain is via *sharding* technology, which has the potential to scale blockchain horizontally. Sharding technology literally partitions the network status into distinct smaller committees, each of which only handles a disjoint set of transactions assigned to it (alternatively called a "shard"). In other words, the transactions are assigned to distinct shards so that different transactions can be handled in parallel among the shards.

Sharding technology advances the distributed ledger technologies by decreasing the processing load on each node, and increasing the overall processing ability of the system. However, sharding is not a one-shot solution to balance the trade-off between scalability and performance. Each shard is an independent and autonomous committee that needs to overcome several key challenges to achieve a consensus. In the literature, there exist several blockchain sharding protocols, e.g., Elastico [4], OmniLedger [5], and RapidChain [6], to address the scalability issues. However, most of these existing sharding protocols fail to consider one important property, *heterogeneity*, among the participating nodes in a practical blockchain system. And, the less-competent nodes may become a bottleneck and hamper the system performance and security (see the motivation in Section II). We, therefore, need a scheme to evaluate the heterogeneity of the participating nodes.

Reputation is a view of one entity on others, commonly established by historical behaviors, which can be regarded as a reliable indicator of the honesty of one entity. We can integrate a reputation scheme into a sharding protocol to mitigate the weaknesses in current sharding protocols. In this paper, we propose a reputation-based sharding scheme, *RepShard*, aiming to achieve both linearly scaling efficiency and system security simultaneously. RepShard adopts a two-layer hierarchical chain structure, including one reputation chain and independent transaction chains. Each transaction chain is maintained within its shard to record transactions, while the reputation chain is maintained by all shards to update the reputation score of each participating node.

RepShard combines a novel reputation scheme with blockchain sharding to enhance its security. Our reputation scheme is established on two factors: the amount of valid and aggregated contributions a participating node has made, and the consistency of these contributions when the system still actively works. This reputation reflects an "integrated contribution" of participating nodes, which is a reliable source to establish the system security. Within the sharding protocol, we design a verifiable relay transaction as the communication channel to deal with cross-shard transactions. A relay transaction carries enough information to verify its validity in destination shard.

This paper presents a reputation-based sharding scheme. Briefly, we make the following contributions:

- We propose a reputation-based sharding scheme, *RepShard*, to achieve linearly scaling efficiency and security at the same time. RepShard adopts a two-layer hierarchical chain structure, recording the transactions and reputation separately.

Postprint online version of paper published in proceeding of 2020 IEEE International Conference on Blockchain (Blockchain'20). DOI https://doi.org/10.1109/Blockchain50366.2020.00037

• RepShard extends the mechanism to calculate the reputation scores, which can be used to explicitly indicate and represent the heterogeneity among the participating nodes when assigning them into shards. Also, the calculation of reputation is based on integrated work, instead of instantaneous computational power.

• RepShard adopts a novel scheme to effectively deal with cross-shard transactions and guarantees the eventual consensus among distinct shards. Our transaction model is based on an *account/balance* model.

The rest of the paper is organized as follows. Section II provides the motivation. Section III shows the models on the system, threat, and transaction. Section IV presents the proposed protocol in detail. Section V provides a security analysis of the proposed protocol. Section VI surveys the related work in this topic, and Section VII concludes this paper.

II. Motivation

Applying a sharding scheme on the blockchain can potentially improve the overall transaction processing capability (w.r.t., *system throughput*) with the help of parallel processing among multiple distinct shards. Practically, the participating nodes in a system pose a huge difference in capacity to contribute a system, such as computational power, bandwidth, and historical behaviors on honesty [7]. However, most existing sharding schemes lack consideration of the heterogeneity among these consensus nodes. This unawareness may significantly affect the overall performance of a system. For example, when applying a random-based sharding [4] [5] or a simple balanced random sharding [6], without considering the heterogeneity among the nodes, those less-competent consensus nodes may become a bottleneck and hamper the system throughput.

Another observation is that most existing blockchain sharding schemes are based on randomness to periodically re-shuffle consensus nodes to ensure the security of a system. And, a *good* randomness needs to meet several important properties, e.g., unbiasability, unpredictability, and public-verifiability, which are not easy to achieve [8]. Without good randomness, the security of a blockchain system can be compromised. Even with good randomness, a blockchain system may also be compromised easily. For example, with a sophisticated attack, such as bribery attack [9], the adversary may have the ability to control a temporary majority (e.g., more than 50%) of the overall computing power, which may, in turn, ruin the entire system.

Motivated by the above observations, we propose to apply a reputation-based scheme to the blockchain sharding to jointly achieve both scalability and security. Roughly speaking, reputation (or reputation score) is a stable indicator of the participating nodes' capability and reliability. The reputation can also enhance system security by balancing the overall reputation scores across different shards, making it more difficult for malicious users to take full control of a specific shard. In a centralized network, it is a trivial task to track a reputation, since all peers' reputation can be managed and maintained by one centralized entity [10]. However, in a *decentralized* network, e.g., blockchain, it is a challenging task to track an accurate reputation, especially in the presence of malicious nodes. Also, it is a challenging task to establish trust among different parties that do not have a trustful connection [11]. We, therefore, require a secure and reliable mechanism to establish a reputation in a decentralized manner, even in the presence of malicious nodes. The adopted reputation scheme is not evaluated by a node's "*instantaneous power*", e.g., the node's power in a short time [12]. Instead, the reputation of a consensus node is established on its historical behaviors and *all* previous *valid and integrated contributions*.

III. Models

A. System and Threat Models

In RepShard, we consider a decentralized network with n nodes, in which the nodes may have different capabilities (*e.g.,* computation power). We assume each node i has a public/private key pair (pk_i, sk_i), and the public key pk_i is used to define its identity. All messages sent over the network are authenticated by the sender's private key. In addition, we make a similar assumption as in previous work, e.g., [4] [5] [6]. For example, the network connections among the honest nodes are well-connected and authenticated. And, our consensus protocol is based on a permissioned setting, in which each participating node requires a unique identity to access and contribute to the blockchain. We denote n nodes $V^n = \{v_1, v_2, ..., v_n\}$ and k shards $S^k = \{S_1, S_2, ..., S_k\}$, where $S^k = S_1 \vee S_2 \vee ... \vee S_k$ and $n = |S_1| + |S_2| + ... + |S_k|$. If we use r to represent the reputation score (e.g., the reputation score of node v_i is r_i), the overall reputation score for each shard is *approximately* equal, e.g., $r(S_i) \approx r(S_j)$, where $r(S_i)$ and $r(S_j)$ are the accumulated reputation scores for shard S_i and S_j, respectively.

Our protocol considers a Byzantine adversary who can corrupt at most f nodes among $3f+1$ participating nodes. We consider a replica (node) as honest if it follows the protocol faithfully and behaves honestly. Similar to almost all existing committee-based consensus protocols, our Byzantine adversary is slowly adaptive. This means that the adversary is allowed to choose a set of nodes to corrupt only at the very beginning of the protocol run and/or the interval between two consecutive epochs, but does not have the ability to change its choice within one epoch.

B. Transaction Model

Unlike the Unspent Transaction Output (UTXO) model [1], our scheme adopts the *account/balance* model [13], which is similar to a bank account. For instance, when trying to approve a pending transaction, the bank first must verify whether its account has enough balance to ensure the transaction can proceed correctly. We can extend the field of *balance* to be a more practical case by adding more fields on it, e.g., with the help of programmable logic for specific applications. Different from UTXO transactions, which can only support full state transitions and updates, the account/balance model has the ability to support incremental state updates, e.g., withdraw or deposit the transaction balance incrementally. In addition, the account/balance transaction model can be easily extended to different application domains, e.g., financial services, industrial IoT.

IV. RepShard: Scalable and Secure Sharding Scheme

This section provides a detailed design principle of RepShard. We focus on the following aspects: sharding and leader election, consensus protocol, and reputation scheme.

A. Protocol Overview

RepShard proposes a two-layer hierarchical blockchain structure, which maintains each shard's transaction chain and a reputation chain, separately. Each shard has its own transaction chain to proceed with the transactions assigned to that shard. The reputation chain is maintained by all shards to record the reputation scores of all participants. The reputation score is based on two aspects: one is the *aggregated* amount of valid and effective work (or contribution) that a participating node has contributed to; the other is the frequency of that work. These two aspects are calculated over the whole period when the system is active and functions well. This reputation scheme can thus reflect the heterogeneity and trustworthiness of the participating nodes. Both the sharding process and the leading election process are based on the reputation scores to achieve a balanced and secure shard. The interval, *w.r.t., epoch*, for updating the reputation chain is much longer than that for the transaction chains, e.g., tens or hundreds of transaction blocks yield one reputation block. Also, the updated reputation score of each shard will be recorded in the reputation chain. The transaction chains deal with transactions at high speed to achieve a high system throughput, while the reputation chain (with a low updating frequency) is used to guarantee the system's security and robustness.

Fig. 1 demonstrates an overview of the RepShard scheme. RepShard maintains two types of epochs: $epoch_{tx}$ (or e_{tx} in Fig. 1) for transaction chains (TX chains) and $epoch_{rep}$ (or e_{rep} in Fig. 1) for reputation chain. The membership of a node to a shard will not change during one e_{rep}; however, the shard leader will change between different e_{tx}. When a new epoch e_{rep} begins for the reputation chain, all participating nodes will be sharded into distinct shards, and one node can only be sharded into one shard. At the end of each e_{rep}, each shard has a *synchronization* process to construct a *temporary* reputation block (TRB) within the shard, which can be further used to construct the reputation block (RB) for the reputation chain. To build RepShard, we identify several key components: sharding and leader election, consensus protocols (including intra-chain and inter-chain), and reputation schemes. These key components will be presented in Section IV-B.

Before discussing the detailed protocol, we first exemplify a high-level scenario on how RepShard processes a transaction (or a payment process), which can be easily extended to other scenarios. Without loss of generality, we consider a cross-shard transaction. A transaction instance involves two participating nodes (or we may call them users or clients) from different shards, i.e., shard A and shard B, as shown in Fig. 2. The nodes from different shards perform two interdependent operations, *withdraw* and *deposit*. Typically, the withdraw operation ρ, which only involves the states in shard A, is handled by the participating nodes of shard A. If the target account contains enough balance to perform this withdraw operation, this transaction will only be included in a block (e.g., block $i+1$ in Fig. 2) of shard A. Once this is done, shard A will compose a *relay transaction* carrying the deposit operation ϕ, and this relay transaction will be forwarded to shard B for further processing to complete this payment. After verifying the validity of the relay transaction and the receiving account in shard B, the deposit operation ϕ will *always* execute without needing to check the balance of the receiving account in shard B. Once the deposit operation is executed and appended to a block (e.g., block $j+1$ in Fig. 2) of shard B, the transaction completes. The intra-shard consensus will guarantee the consistency within a shard. We assume the whole system is uniformly split into $k = 2^{\kappa}$ distinct shards, and a shard is identified by both its *sharding scale* κ and *shard index* s ($s \in \{0, 1, ..., 2^{\kappa}-1\}$). According to the sharding scale κ, it is a trivial task to locate the sharding index.

B. Description

1) Sharding and Leader Election: We first need to assign the participating nodes into distinct shards, and then we elect a leader within that shard. A good sharding and leader election process must preserve several critical properties to ensure the security of each shard, e.g., randomness and resemblance. We adopt a similar nodes sharding scheme as in [7]. However, our scheme is a hierarchical blockchain structure, and the reputations come from the reputation chain, instead of from each individual state block of [7]. Also, our leader election scheme is based on our reputation scheme with randomness.

Randomness is used to guarantee that the whole process is unpredictable. Both sharding and leader election processes must be random to prevent prediction attacks, e.g., predicting which shard a node belongs to, or which node is the leader. *Resemblance* means the equivalence of reputation score in each shard. Sharding nodes into shards is based on both randomness and reputation, and each shard should be balanced to some extent. The ideal assignment is that each shard has a relatively similar reputation score in sum and the size of each shard, e.g., almost the same as the number of honest and malicious nodes. This feature increases the security of each shard, preventing one particular shard from being compromised. Besides that, due to heterogeneity among the participating nodes, the results of both sharding and leader election processes should be easily verified by each participating node independently, without involving too much communication overhead. Our sharding and leader election scheme can guarantee the above critical properties.

a) Nodes Sharding: The system first runs a node sharding algorithm to assign the participating nodes into shards. Our node sharding algorithm is based on two metrics: the randomness and the cumulative reputation score r^w of a node over all its previous w epochs. Algorithm 1 shows a reputation-based nodes sharding scheme in pseudo-codes. To achieve public-verifiability among all participating nodes, we utilize the hash value of the previous reputation block (except the genesis block), as the random seed of the current epoch, to initialize the random generator G_{rand} (Line 1 in Algorithm 1). Within one reputation epoch, the membership of the shard nodes does not change, except its shard leaders, since the shard leader changes for each transaction epoch. From the last reputation block, we can easily obtain the aggregated reputation score of the previous epoch, which is a list R^{e-1}.

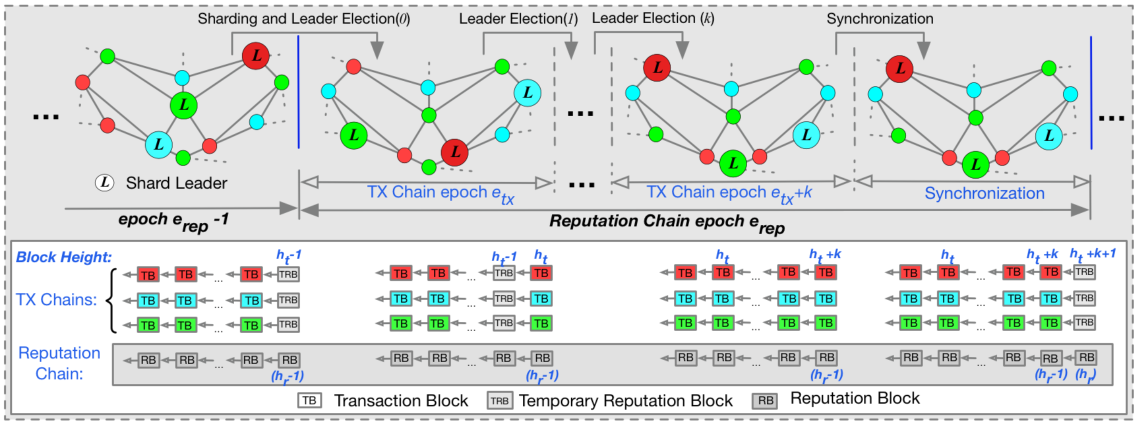

Fig. 1: RepShard overview. RepShard adopts a two-layer hierarchical structure, and each layer has its chain and epoch: e_{tx} for the transaction chain and e_{rep} for the reputation chain. The membership of a node to a shard will not change during one e_{rep}, however, the shard leader will change between different e_{tx}. Different colors represent different shards and the corresponding transaction chains. Node with L means the current shard leader. At the end of each e_{rep}, each shard has a *synchronization* process to construct a *temporary* reputation block (TRB) within the shard, which can be further used to construct the reputation block (RB) for the reputation chain.

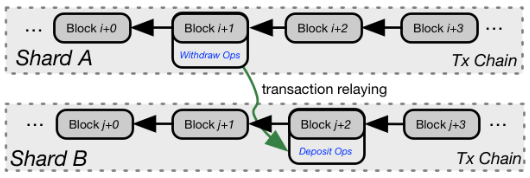

Fig. 2: Message passing mechanism across distinct shards with the help of relay transactions based on *Account/Balance* model.

Algorithm 1: Reputation-based Nodes Sharding

```
Input:
A random set from previous epoch $e-1$
$Seed^e = \{RBH_1^{e-1}, RBH_2^{e-1}, ..., RBH_k^{e-1}\}$, where $RBH$ represent Reputation Block Hash;
The aggregated reputation scores
$R^{e-1} = \{r_1^{e-1}, r_2^{e-1}, ..., r_n^{e-1}\}$ of each node over all the previous $e-1$ epochs;
Output:
$k$ distinct shards $S = \{S_1, S_2, ..., S_k\}$, where
$S_i \wedge S_j = \emptyset$ and $S = S_1 \vee S_2 \vee ... \vee S_k$.
1 Initialize the shards and random number generator:
    $S_i = \emptyset$, where $1 \le i \le k$; $G_{rand}$: = $Seed^e$
2 Sort the aggregated reputation score:
    $R^e = sort(R^{e-1})$
3 for each reputation score $r_i^e$ in $R^e$ do
4     Generate a index sequence $\{t_1, t_2, ..., t_j\}$ and the
        corresponding set $\{S_{t_1}, S_{t_2}, ..., S_{t_j}\}$ of $S$
        satisfying $|S_{t_1}| = |S_{t_2}| = ... = |S_{t_j}| =$
        $min(|S_1|, |S_2|, ..., |S_k|)$, where $j$ is the number
        of satisfied subsets.
5     Obtain a random number (a integer) $x$ from
        $G_{rand}$.
6     Allocate node $i$ with reputation score $r_i^e$ to $S_{t_u}$
        that $u = x$ mod $j$.
7 end
```

We sort all the reputation scores in R^{e-1}, in descending order, as a new list R^e. For each element in R^e list, it is assigned to a shard, which typically is with a minimum size, so that it can maintain the property of *resemblance*. If multiple shards satisfy the minimum size requirement, the system randomly selects a satisfied shard to assign the node (Line 4 - 5 in Algorithm 1). For each iteration (Line 3 - 7 in Algorithm 1), we choose the current unassigned node with the largest reputation score to assign into a shard. This inherently maintains the balance and equivalence of the total reputation score in each shard.

b) Leader Election: After the shards are formed and the participating nodes are assigned to the corresponding shards, each shard starts its leader election process. Similar to the nodes sharding process, our leader election scheme is also based on the same metrics: the randomness and the cumulative reputation score r^w of a node over all its previous w epochs. Algorithm 2 shows a leader election process in pseudo-codes. The reputation means a node with a high reputation score has more chance to be elected as the shard leader. We follow the proof-of-stake (PoS) protocol to form a potential committee of leaders, and the shard leader is randomly selected from the potential leader committee. In our case, only the nodes having enough high reputation scores, e.g., higher than both the median score and the mean score of all shard members, can have the chance to be a shard leader. If a node does not satisfy the requirements, it will be assigned a value $+\infty$. Each qualified node is required to divide its reputation score by a randomly generated number based on the same random number generator G_{rand} and node id (e.g., the public key of the node) (Line 7 - 8 in Algorithm 2), this will ensure the randomness of a potential leader. We then select the node with a minimum value as a shard leader. After each shard selects its leader, the shard runs the consensus process to deal with transactions.

2) Consensus: This section presents the consensus protocols we adopted. Specially, we discuss intra-shard and inter-shard consensus protocols separately. The intra-shard consensus is responsible for the transactions within one shard, while the inter-shard consensus handles the transactions across multiple shards.

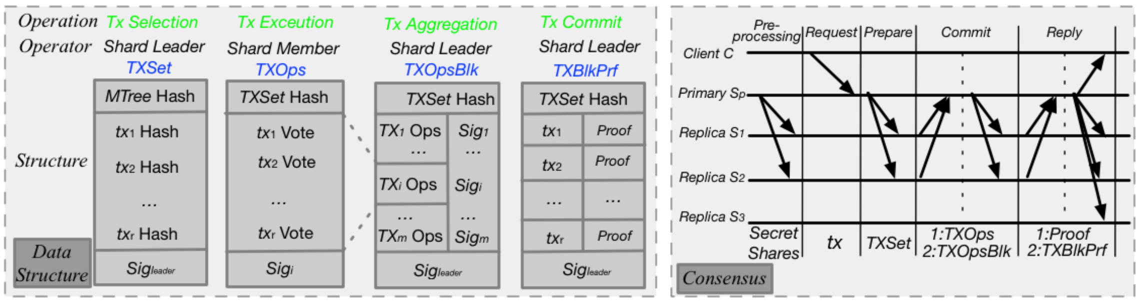

Fig. 3: Intra-shard consensus overview. The left part is the data structures of transactions in different consensus phases; the right part is a scalable BFT consensus protocol powered by a threshold signature.

Algorithm 2: Reputation-based Leader Selection

```
Input: $k$ distinct shards $S = \{S_1, S_2, ..., S_k\}$ from Alg. 1;
Output:
$k$ distinct shard leaders $L = \{l_1, l_2, ..., l_k\}$, where $l_i \in S_i$.
1  Initialize the leaders for each shard: $l_i = \emptyset$ where $1 \leq i \leq k$
2  for each shard $S_i \in S$ do
3      $r^{med}$ = median of the reputation score of $S_i$
4      $r^{mean}$ = mean of the reputation score of $S_i$
5      $\lambda = \frac{1}{2}(r^{med} + r^{mean})$
6      for each validator $v_j \in S_i$ do
7          if ($r_j^{e-1} \geq r^{med}$ && $r_j^{e-1} \geq r^{mean}$) then
8              Obtain a random float $0 < x < 1$ from $G_{rand}$
9              $w_{i,j} = \frac{1}{2}(1 + \frac{x - r_j^{e-1}}{\lambda + |x - r_j^{e-1}|})$
10         else
11             $w_{i,j} = +\infty$
12         end
13     end
14     $l_i = v_j$ where $w_{i,j} = min(w_{i,1}, w_{i,2}, ..., w_{i,|S_i|})$
15 end
```

a) Intra-shard Consensus: Considering the scalability within a shard, the intra-shard consensus is based on a scalable BFT protocol. This scalable BFT protocol uses a threshold signature to reduce the communication complexity among the participating nodes. Simply, a threshold signature scheme can guarantee that, for a given threshold parameter k ($1 \leq k \leq n$), any k signers from a total of n signers can collaboratively generate (or *recover* during decryption) a valid signature for any given message. However, there is no way to do so when the number of signers is less than the threshold k [14]. Our BFT protocol utilizes a well-established Boneh-Lynn-Shacham (BLS) signature [15] as a threshold signature scheme.

To apply a BFT protocol on account/balance transactions, we need some data structures on transaction-block for the transaction chain in each shard. The left side of Fig. 3 shows the related data structures, including *TXSet, TXOps, TXOpsBlk* and *TXBlkPrf*. *TXSet*, Transaction Set, contains a set of transactions for consensus procedure, which consists of the hashes of transactions in a specific order (e.g., ascending order) and the signature from shard leader. *TXOps*, Transaction Operations, provides the operations and the decisions (e.g., the vote on the validity of a transaction) from the shard members, as well as their signatures. Taking a *withdraw* operation as an example, if the corresponding account has enough balance to cover the withdraw operation, then this withdraw operation is valid, and the node votes "*yes*". Note that the entities on *TXSet* and *TXOps* should be in the same order. Each shard member shows a decision on each transaction, e.g., valid/invalid or yes/no options, within its shard. *TXOpsBlk*, Transaction Operations Block, contains all *TXOps* from all shard members. *TXBlkPrf*, Transaction Block Proof, contains the commit-proof information for this round consensus, so that each honest node can decide by itself if this block is valid or not. When the epoch starts, the shard leader first needs to collect and broadcast the pending transactions, submitted by clients, within its shard. And all the communicated messages must be signed by its *original* sender.

For TXSet, each shard member checks all transactions in this TXSet and makes a decision on the validity of each transaction. Our design considers the heterogeneity among the participating nodes, e.g., some participating nodes may not have enough resources to make a decision at that moment. Instead of only providing two options, e.g., *Yes* or *No*[1], we provide a third option *Unknown*. The option *Unknown* is given by a node to avoid punishment on its reputation score in the following cases: (1) when it does not have the ability to handle so many transactions due to hardware limitations; (2) when the transaction itself has not been received. After verifying the transactions, the shard members are required to send their decisions, with the signatures, back to the shard leader for further processing.

The consensus procedure of our scalable BFT protocol is elaborated below and illustrated on the right side of Fig. 3.

• *Pre-processing*: In this phase, the primary S_p generates a group of random secret shares using a BLS threshold signature scheme, and then publishes the generated cryptographic hash of each secret share. Then, S_p sends one share to each S_i. At the end of this phase, each replica possesses one distinct private key for signing (alternatively called 'signing key'), which can be further used to generate a signature share for any message.

• *Request*: In this phase, the clients[2] send operation requests to S_p. The S_p handles these requests and constructs a transaction set *TXSet* from its gathered transactions[3].

[1]About the validity of a transaction, *Yes* implies the transaction is valid, *No* implies the transaction is invalid.

[2]The client can be a transaction from the originate shard if involving the cross-shard transaction, e.g., a *relay* transaction containing the *deposit* operations.

[3]Typically, each replica has a transaction pool to buffer these received transactions.

• *Prepare*: In this phase, S_p sends the *TXSet* to all replicas. Upon receiving this *TXSet*, each replica prepares its vote on the validity of each transaction in the form of *TXOps*, and signs this constructed *TXOps* using its private signing key. Each replica then prepares its commitment proof on this *TXOps*.

• *Commit*: In this phase, each S_i signals its commitment by sending its *sign-share* message, together with *TXOps* and its commit proof (this proof is signed by S_i's own private key). S_p then gathers all signature shares, and logs all received commit proofs together to create a succinct *full-commit-proof* for the *TXBlkPrf*. Besides, the primary S_p will also collect all returned secret shares from each replica to re-assemble the secret, and this process can represent an aggregated commitment from all replicas. S_p multicasts this reassembled secret, the *full-commit-proof* and the *TXBlkPrf* to all S_is, which is sufficient for replicas to verify.

• *Reply*: Similar to the *Commit* phase, all the *reply* messages from all active replicas will be aggregated to S_p to verify the commitment on full-commit-proof and *TXBlkPrf*. In the second phase of the reply, after each replica S_i verifies the secret, it starts to reveal its secret share to S_p. The primary then rebuilds the reply secret and forwards the secret with all communication logs to the clients for their verification.

Each shard member keeps checking the received *TXBlkPrf* and transaction-block from the leader. If any malicious behaviors (e.g., tampered data) performed by a leader are detected, the honest shard member will broadcast a signed *Warning* signal to other shard members. When at least half of shard members send out the *Warnings*, the honest members can begin a rollback process. The rollback process follows a general view-change process, e.g., in [16]. In addition to the general operations (e.g., electing a new leader), our rollback process will clean the malicious leader's cumulative reputation score to an initial state, e.g., 0.

b) Cross-shard Transactions: The intra-shard consensus potentially reduces both computation and communication overheads within the shard by isolating the transaction validation and confirmation within a small range. Typically, a cross-shard transaction is more complicated, because all associated shards must reach a consensus and keep consistency among the involved shards. This section presents a novel way to handle cross-shard transactions based on the adopted transaction model. Similar to the settings adopted in [13] [17] [18], we consider a cross-shard transaction scenario: a withdraw operation ρ from payer a and a deposit operation ϕ to payee b, and a and b are from different shards, e.g., Shard A and Shard B, respectively. To complete this cross-shard transaction, we perform a two-phase process: Withdraw Operation at Shard A (e.g., transaction validation and forwarding at Shard A), Deposit Operation at Shard B (e.g., relay transaction processing at Shard B).

Suppose a cross-shard transaction tx is $< \rho, a, \phi, b >$, which is an unconfirmed transaction.

Phase 1: Withdraw Operation at Shard A

The shard leader of Shard A picks up this unconfirmed transaction tx (together with other unconfirmed transactions) to construct a new block (e.g., *TXSet*). The shard leader checks whether the withdraw operation in tx is valid or not. By comparing the balance of a with the amount to transfer in tx, if the amount to transfer is no more than the balance, then tx is valid; otherwise, the balance is not sufficient to perform the withdraw operation, and tx will be labeled as invalid. The corresponding decision (either valid or invalid) will be included in the proposed block for verification from its shard members, which involves a round of intra-shard consensus. For this block, the shard leader typically does not have the right to include a *Unknown* option, but it has the right to propose a blank block in case of no transaction at that epoch. Otherwise, a *tentative* chaining-block Θ^a and a tentative transaction block Φ^a are constructed by the leader. Φ^a contains a list of valid transactions, from a perspective of the leader, including the transaction from a to b, in the form of *TXSet*[4].

Within Shard A, the shard leader runs an intra-shard consensus protocol. The transaction block Φ^a consists of both intra-shard and inter-shard transactions. If a transaction is the intra-shard transaction (e.g., both payer and payee are in the same shard) or an inbound relay transaction (e.g., only payee in this shard for deposit operation), then this transaction is executed and completed at this stage. The withdraw operations ρ in all cross-shard transactions of Φ^a are then executed within shard A, and the deposit operation ϕ will be executed at another shard. Each cross-shard transaction derives an outbound relay transaction $\psi :=< \phi, b, \gamma >$ (where γ is a verification data, e.g., *TXBlkPrf*), which will be sent to the destination shard B.

Phase 2: Deposit Operation at Shard B

Suppose the relay transaction $\psi :=< \phi, b, \gamma >$ has been forwarded to Shard B from Shard A, as an inbound transaction, as shown in Fig. 2.

The shard leader of Shard B picks up the inbound relay transaction ψ to construct a new block. The leader first checks the validity of ψ, e.g., verify against its originating block Θ^a of shard A. This is used to verify the validity of this relay transaction. If it is not invalid, then skip this inbound relay transaction. Otherwise, the leader constructs a chaining-block Θ^b as well as a transaction block Φ^b. Θ^b includes this inbound relay transaction ψ. Similar to the scenario in Shard A, this block is tentative. The shard leader broadcasts this block Φ^b to its shard members. An intra-shard consensus is executed and concluded. As a rule of thumb, the deposit operation ϕ is always executed if the destination shard verifies its validity. And once the deposit operation ϕ is executed, then the lifecycle of the transaction $< \rho, a, \phi, b >$ is completed.

To ensure that the relay transaction is indeed executed, we add an acknowledgment mechanism to its originate shard to guarantee the relay transactions are executed. Upon execution of the relay transaction, the destination shard will send its originate shard a confirmation (at the *reply* stage of intra-shard consensus), including the created chaining-block and the relay transaction. In case that a relay transaction is accidentally discarded, the discarded transaction can be re-launched, according to the confirmed initiative transactions on the chain, by any participating node of the originate shard.

[4]We call it "tentative" because it might contain some transactions (either intra-shard or inter-shard) which are not valid in the view of other shard members. It must pass the intra-shard consensus to agree on the proposed transaction block.

3) Reputation Scheme: Previous sub-sections describe detailed principles to construct a transaction chain. This section focuses on the construction of the reputation chain. Our reputation scheme is based on the concept of proof-of-reputation (PoR), which can be used to enhance the security of RepShard. With PoR, in addition to create enough transaction blocks, the participating node must demonstrate that it has behaved honestly and participated in creating these transaction blocks on a regular basis. A node's reputation is established based on all its previous behaviors, which is an accumulated result. In contrast, several previous schemes, such as proof-of-work (PoW) or proof-of-membership (PoM), an attacker with a high *instantiate* computational capability can join the system and launches an attack. Different from traditional PoRs, RepShard is based on *all* previous *valid and integrated work* to establish the participationship in a system. We first present the reputation scheme, then illustrate how to construct a reputation chain based on our reputation scheme.

a) Reputation System: When a fixed number of transaction blocks are chained (e.g., one hundred blocks), a temporary reputation block will be constructed among all shard members to reflect the reputation of its shard members. Each shard member individually calculates a temporary reputation block according to available information, e.g., *TxBlkPrf*s and the transaction blocks. Given the transaction chain, our scheme can guarantee the reputation score of any node can be calculated at any time. Based on the public agreed blocks on the reputation chain, every consensus node has its own copy of the reputation scores about all other participating nodes. The calculation of reputation score R of a node can be obtained from Algorithm 3.

With publicly available information on each shard, it is a trivial task to convert the information on a transaction block to a matrix. We denote some notations used to calculate the reputation.

• $H_{MN}^{(i)}$: a $M \times N$ matrix for transaction block i, where M is the number of transactions in block i and N is the number of shard members to construct this block. The $h_{mn}^{i} \in H_{MN}^{(i)}$ (where $1 \leq m \leq M$ and $1 \leq n \leq N$) is the element at the m-th row and n-th column, and it satisfies the following equation:

$$h_{mn}^{i} = \begin{cases} 1 & \text{if node } n \text{ contributes on transaction } m \\ -1 & \text{if node } n \textit{ misbehaves} \text{ on transaction } m \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

• α: a column vector in the matrix, e.g., $\alpha(i, s)$ denotes the s-th column vector of block i (or matrix $H_{MN}^{(i)}$).

• $\mathbb{1}$: a row vector containing all value "1", e.g., $\mathbb{1}_m = [1, 1, ..., 1]_{1 \times m}$.

• c: a function on vector α to calculate the honesty that a node behaves on a block, which satisfies the following equation:

$$c(\alpha) = \begin{cases} 1 & \text{if } \alpha \text{ does not contain "-1"} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

• w: a weight function on $H_{MN}^{(i)}$ (e.g., the leader node will get more rewards), which satisfies the following equation:

$$w(i, s) = \begin{cases} w_0 & \text{if the node in } s\text{-th column of } H_{MN}^{(i)} \\ & \text{is the leader node of block } i \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

where w_0 ($w_0 > 1$) is a system parameter to represent the work contributed by shard leader.

• m_i: the number of transactions included in block i (or the number of rows of $H_{MN}^{(i)}$).

• B: the number of transaction blocks generated in current reputation epoch e_{rep}.

• R_{e-1}: the reputation score of the previous epoch.

• (a, λ): the system parameters.

Algorithm 3: Reputation Score Calculation

```
Input:
Transaction chain length $B$;
Matrix-related $\{H_{MN}^{(i)}\}_{i=1}^{B}$, $\{m_i\}_{i=1}^{B}$;
Previous reputation $R_{e-1}$
Parameters $a, \lambda, Ext$
Output:
The reputation $R \in [0, 1]$
1  $x_0 = \sum_{i=0}^{B} m_i - \sum_{i=0\&\&1 \leq m \leq m_i}^{B} |\mathbb{1}_{h_{ms}^{i}=0}|$
2  $x = \sum_{i=1}^{B} \sum_{s=1}^{N} \mathbb{1} \cdot \alpha(i, s) \cdot c(\alpha(i, s)) \cdot w(i, s)$
3  $f(x) = \frac{1}{2}(1 + \frac{x-a}{\lambda + |x-a|})$
4  if $x < x_0$ then
5      $H = 0$
6  else
7      $H = 1$
8  end
9  $R_e = min(1, H \cdot (Ext + f(x)))$
10 $R = \frac{R_e + R_{e-1}}{2}$
```

Considering the heterogeneity, we add some initial external views on participating nodes, e.g., $Ext \in [0, 1]$. When a very reliable and powerful node (e.g., a node with tamper-resistance hardware) joins RepShard, it may have an initial reputation that is higher than that of a random individual joiner. This provides a fair share, namely reputation, for a new joiner. For different behaviors, e.g., honest or dishonest, we pose distinct scaling criteria to either reward or punish. In general, the punishment of dishonest behaviors is more severe than the reward for honest behaviors, e.g., dishonest behaviors may cause the node to lose all its accumulated reputation score. This is very important and critical for our reputation system. The case to commit an illegitimate transaction is more severe, or even disastrous to the whole system, than the case of aborting a legitimate transaction. Typically, an aborted legitimate transaction can be resumed, however, once an illegitimate transaction is confirmed, it may be a disaster for the system and result in system inconsistency. The option of "*Unknown*" typically means that the participating nodes do not have enough computing/communication resources to make a decision; thus, the system neither rewards nor punishes the corresponding participating nodes on their cumulative reputation scores. $H \in \{0, 1\}$ is to show the *honesty* of the node, which is set to "1" for the successive contribution, is set to "0" if any malicious behaviors found. A node is said to be a misbehave if: (i) it presents conflicting signed messages to other consensus group members; or (ii) it commits transaction blocks with conflicting transactions when this node is as the leader [12].

Our reputation scheme specifies three features: careful start, quick reward, and prevention of over-control. The careful start is achieved via an initial slow growth; the quick reward is for mature participants via a fast growth rate to reward the calculative honest nodes; the prevention of over-control is achieved via a slow increase when it nears to the top. The sigmoid function $f(x)$ can be used to define this kind of progression, which is a bounded, differentiable, and real function. This function can achieve the above targets. For example, increasing the reputation scores slowly at the beginning, regardless of how powerful a node might be, this can potentially prevent the attack from an instantiate powerful adversary. The node is required to be continuously in the system for a long time, and should always behave honestly. Once a node earns enough reputation to a certain level (e.g., up to a threshold), this node is literally proved to be a trusted enough one, and after that, its reputation score will increase much faster. However, if any malicious behavior is detected, this node will lose its reputation. Also, according to the sigmoid function, the reputation score cannot grow forever. In Algorithm 3, the reputation function can be parameterized by adjusting the parameters (a, λ), which makes it flexible to different settings, e.g., changing the speed of increasing the reputation scores. This parameterized function makes the reputation scheme more robust.

In general, our reputation scheme ensures that a node's reputation is computed based on its *integrated contribution*. The integrated contribution is measured by *all* its valid work, during the time of being active and honest, to the whole system. The instantaneous power at a given time has little effect on the overall reputation score. The adopted reputation scheme can guarantee the robustness of the whole system. For example, after the system successfully runs a sufficient time, there is no way for a node (e.g., even equipped with powerful resources) to quickly establish its reputation. To obtain a high reputation, the node must work both honestly and regularly all the time, with no other way to achieve a high reputation.

b) Reputation Chain: RepShard is a two-layer hierarchical structure, and the reputation chain is based on the integrated work on the transaction chains. And the integrated work contributes to the reputation score. A shard, as a committee, applies a collective signing, e.g., CoSig [19], to create a temporary reputation block, which can be chained to the transaction chain as a specific block. This temporary reputation block carries the reputation of each shard member, the confirmed transaction blocklists, the previous reputation score from the reputation chain, as well as a collective signature from the participating nodes. The temporary reputation blocks are chained to the transaction chain, which can be viewed as a state block, while the reputation blocks in the reputation chain are a collection of temporary reputation blocks from each shard, and contain all the reputation scores of all nodes. The collective signature on a temporary reputation block includes the identity information on who signed this block, and correspondingly, this signature can be publicly verified by other shards without involving private information.

As each temporary reputation block is agreed upon and confirmed within the shard, we can literally consider the leader of a temporary reputation block to be honest. We construct a second layer of committees, consisting of the leaders from each shard called the reputation committee. Each temporary block is gossiped among the reputation committees, which can be considered a transaction of the reputation committee. The reputation committee will run one round intra-shard consensus to obtain the final reputation block and chain it to the reputation chain. The final reputation block contains only the reputation scores from each temporary reputation block. Unlike transaction blocks in the transaction chain, the reputation block does not need to perform a complicated account/balance check; they simply aggregate the temporary reputation blocks together and verify the validity of these temporary reputation blocks.

Once a reputation block has been created on the reputation chain, the reputation scores of the nodes are updated. RepShard then runs the sharding and leader election processes (see Section IV-B1) for the next reputation epoch.

V. Security and Performance Analysis

A. Consensus Security

This section demonstrates the safety and liveness of the intra-shard consensus, and the atomicity analysis of cross-shard transactions.

a) Safety: Following the intra-shard consensus, we claim that a malicious leader does not have the ability to generate a wrongful transaction block using a correct *TXOpsBlk*. The *TXOpsBlk* contains all the signatures of its shard members, and a malicious leader cannot modify this set based on the threshold signature. And the threshold signature requires a threshold number of participating nodes agreeing on the performed operations. If more than half shard members have sent the "Warning" signals in their decisions, the rolling scheme will act to intercept and stop the functionality of the malicious leader. Accordingly, the system will clean up the malicious leader's cumulative reputation, or even kick this malicious node out of the system. For example, if the reputation is cleared, to be a shard leader again, the node must continue to work hard and contribute enough integrated work before gaining the right to be a leader. Also, we claim that it is impossible to create a wrongful transaction block under an honest shard leader. The transaction block created by an honest leader will typically be accepted by all other honest nodes, since we assume the majority of nodes are honest.

b) Liveness: In our protocol, an honest leader always sends out a valid *TXSet* to its shard members. If each shard has at least half of its shard members behaving honestly[5], the transaction decision set $TXOpsBlk$ can be obtained. And the decision on a valid block made by an honest leader will be accepted by all honest shard members. However, if the shard leader behaves maliciously, e.g., by preventing the progress of consensus, the rolling scheme can detect, e.g., via monitoring the number of "Warning" messages, and take over the cleanup processes, and a new shard leader will be elected to continue the protocol.

[5]This proof is a trivial task, based on recursive formula and hypergeometric distribution [20].

c) Cross-shard Transaction: Our transaction model is based on account/balance transaction model, involving both *withdraw* and *deposit* operations. For cross-shard transactions, we allow the withdraw operation executes first in one shard, and then the corresponding deposit operation is settled later in other shards. As long as the withdraw operation was successfully verified and confirmed, the deposit operation will finally be completed, either within the same shard or at other shards. The originate shard records a proof of accept or reject for cross-shard transactions on a relay transaction. Once the relay transaction is forwarded to its destination shard and successfully be duplicated among the shard members, there is no way to let this transaction expire unless the transaction is included in a valid transaction block. In case a relay transaction is dropped, according to the established block information on the originate shard, the relay transaction still can be re-built by any consensus nodes of the originate shard. In the case of a malicious leader, the rolling scheme will take over to undo unconfirmed transactions and elect a new leader to redo all unconfirmed cross-shard transactions.

B. Reputation Security Analysis

When applying our reputation scheme, an attacker who wants to break the system needs to gain some reputation and hence contribute to the blockchain, rather than relying only on its computational ability, e.g., computing power in PoW. The reputation of a participating node in RepShard is based on the continuity and regularity of the valid work that contributes to the system. To explore the worst-case scenario, we assume no participating nodes are having established an agreed-upon trust, e.g., Ext is set to be 0 for all nodes. Many reputation-based schemes have a similar assumption, e.g., the work [12] [21]. For simplicity, we assume that each participating node (even a malicious one) faithfully follows the protocol and behaves honestly for a certain period of time, e.g., mimicking the *camouflage attacks*.

From our reputation scheme in Section IV-B3, we limit the upper bound that one node can contribute to the whole system to prevent a small group of nodes controlling and managing the whole system. The reputation scheme is built upon the "integrated power" instead of "instantaneous power". And each shard is almost balanced, in which each shard has almost the same number of participating nodes (refer Section IV-B1 for details). We can prove that the growth rate of our reputation scheme for any participating node is bounded.

In RepShard, except those nodes that provide the *Unknown* option on their decisions, at any time of the system being active, the growth rate of the reputation scores for any node is bounded by $\frac{1}{2\lambda}$[6].

This bounded reputation growth rate can prevent any node from becoming a "supernode". The reputation in RepShard is a collective reputation of the shard members. In Section IV-B1 sharding and leader election, only the nodes with much higher reputation scores have the right to be a member of the potential shard leader committee. This means, the decision power of a participating node partially depends on its cumulative reputation score in the shard.

[6]λ is a system parameter in Line 3 of Algorithm 3. The proof of this theorem is a trivial task.

For any shard consensus group, RepShard ensures the safety of consensus protocol under the following two conditions, (i) no more than half of participating nodes are compromised; and (ii) the total of reputation score of compromised nodes $\mathbb{R}_{compromised}$ fulfils the following condition:

$$\mathbb{R}_{compromised} < \frac{\sum_{i=1}^{|X|} R_i}{3},$$

where $|X|$ is the size of a shard, and the shard member's reputation score R_i is obtained from Algorithm 3. This means, the RepShard system keeps safe unless both conditions (i) and (ii) are broken, which rate of success is negligible.

C. Performance Analysis

Assuming that the tx is assigned to a shard with the size c. When consensus protocol starts, the user sends tx to the originate shard to proceed with the withdraw operation. Specifically, the user sends tx to one of the participating nodes of that shard (e.g., the shard leader), then this participating node will multicast this tx within its shard, which incurs a $O(c)$ communication cost. The shard leader creates a transaction set $TXSet$ containing tx, then it multicasts $TXSet$ to all its shard members for the consensus processes, which induces a communication complexity of $O(c)$. After the decision making process, the leader will collect all transaction operation results $TxOps$ from shard members, and this involves another $O(c)$ communication overhead. The shard leaders are required to multicast the transaction operation blocks $TxOpsBlk$ and transaction block proof *TXBlkPrf* to all its shard members, each of which takes $O(c)$ time for intra-shard consensus. The overall communication overhead during this consensus procedure would be $O(c)$.

If tx is a cross-shard transaction, it involves a relay transaction. This requires that the operations in $TXOpsBlk$ of the originate shard should be transmitted to the destination shard (deposit shard) in the form of a relay transaction. In our scheme, the relay transaction is *directly* forwarded to the destination shard. Given the fact that the number of deposit shards in a practical system is constant (e.g., typically only one deposit shard in our *account/balance* model), the total communication between different shards for the tx is constant $O(1)$. Thus, the communication complexity of the consensus protocol for any kind of transaction, either intra-shard or inter-shard, would be $O(c)$.

In RepShard, compared to a general participating node in shard, the shard leader typically needs to contribute more computational power and bandwidth to create and deal with the transaction set, $TXSet$, transaction operations $TXOps$, and transaction blocks $TXOpsBlk$. Actually, RepShard explicitly takes the capacity variations among the nodes into consideration, e.g., the heterogeneity of nodes. In particular, the reputation scheme encourages honest nodes to participate in the consensus processes faithfully; by filtering out the "lazy" or malicious nodes, it can also improve overall system performance. Those nodes that work faithfully earn more reputation scores by contributing more computational power and bandwidth, which in turn might be transferred into monetary reward.

VI. Related Work

Sharding-based protocols provide a potential solution to scale blockchain. There exist many these protocols in literature, e.g., Elastico [4], OmniLedger [5], RapidChain [6] and Monoxide [13], to name a few. In OmniLedger, the adversary has an ability to corrupt at most $1/4$ fraction of all participating nodes. Similar to Elastico, Ominiledger has several challenges that leave unsolved, e.g., lower fault tolerance. Similar to OminLedger, RapidChain has some weaknesses. For example, RapidChain depends mainly on the reference committee during the sharding process and processing cross-sharding transactions. Monoxide proposes the concept of asynchronous consensus zones for a blockchain system, and each zone acts as an independent shard to perform the consensus task. However, the overall system is still based on PoW protocol, which is a power-consuming protocol. Our SoK work [22] provides a detailed comparison of popular sharding schemes.

Most existing sharding-based protocols fail to consider the capability differences. They typically only consider the honesty, either honest or malicious; other capabilities are assumed to be all the same. However, these assumptions are not exactly true in a practical system. Also, most of these protocols are based on a random scheme, which is unfair for less-competent participants. We, therefore, need a secure scheme to establish participation among nodes. Reputation is a good candidate to do so. CertChain [23] presents a consensus protocol based on the dependability among nodes. It develops a reward scheme considering the misbehavior of the nodes. The design of CertChain is targeted solely for the applications of certificate managements, and without considering the scalability issues when adopting to large-scale applications. RepuCoin [12] aims to integrate a reputation-based scheme into its chain design, whose reputation depends on the PoW a node contributes without considering the heterogeneity of nodes. Also, this scheme does not consider the issues of scalability.

VII. Conclusion

This paper proposes a reputation-based blockchain sharding scheme, *RepShard*, which achieves scalability and system-level security at the same time. RepShard leverages a hierarchical chain structure to record the transactions and reputation separately. The transactions are handled by each shard in parallel to achieve scalability, and a separate reputation chain updates each node's reputation via the proposed reputation scheme. The reputation scheme is based on the valid work that each node contributes to the system. As future work, we plan to implement the proposed RepShard scheme and thoroughly evaluate the performance in terms of throughput and transaction latency on a real-world application.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, pp. 2–5, 2017.

[3] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—a systematic review," *PloS one*, vol. 11, no. 10, p. e0163477, 2016.

[4] 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.

[5] 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.

[6] 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.

[7] 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," *arXiv preprint arXiv:1901.05741*, 2019.

[8] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 444–460.

[9] P. McCorry, A. Hicks, and S. Meiklejohn, "Smart contracts for bribing miners," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 3–18.

[10] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," in *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. ACM, 2003, pp. 144–152.

[11] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.

[12] J. Yu, D. Kozhaya, J. Decouchant, and P. Verissimo, "Repucoin: Your reputation is your power," *IEEE Transactions on Computers*, 2019.

[13] 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)*, 2019, pp. 95–112.

[14] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 568–580.

[15] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

[16] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Smchain: A scalable blockchain protocol for secure metering systems in distributed industrial plants," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, 2019, pp. 249–254.

[17] S. Ma, Y. Deng, D. He, J. Zhang, and X. Xie, "An efficient nizk scheme for privacy-preserving transactions over account-model blockchain." *IACR Cryptology ePrint Archive*, vol. 2017, p. 1239, 2017.

[18] H. Chen and Y. Wang, "Sschain: A full sharding protocol for public blockchain without data migration overhead," *Pervasive and Mobile Computing*, vol. 59, p. 101055, 2019.

[19] 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.

[20] A. Hafid, A. S. Hafid, and M. Samih, "A methodology for a probabilistic security analysis of sharding-based blockchain protocols," in *International Congress on Blockchain and Applications*. Springer, 2019, pp. 101–109.

[21] A. Biryukov, D. Feher, and D. Khovratovich, "Guru: Universal reputation module for distributed consensus protocols," University of Luxembourg, Tech. Rep., 2017.

[22] 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.

[23] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "Certchain: Public and efficient certificate audit based on blockchain for tls connections," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2060–2068.