

CycLedger: A Scalable and Secure Parallel Protocol for Distributed Ledger via Sharding

Mengqian Zhang*, Jichen Li[†], Zhaohua Chen[‡], Hongyin Chen[§] and Xiaotie Deng[¶]

*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China

^{†‡§¶}School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China

Email: *mengqian@sjtu.edu.cn, {[†]1600012970, [‡]chenzhaohua, [§]1600012903, [¶]xiaotie}@pku.edu.cn

^{*†‡§}These authors contributed equally to this work.

Abstract—Traditional public distributed ledgers have not been able to scale-out well and work efficiently. Sharding is deemed as a promising way to solve this problem. By partitioning all nodes into small committees and letting them work in parallel, we can significantly lower the amount of communication and computation, reduce the overhead on each node’s storage, as well as enhance the throughput of the distributed ledger. Existing sharding-based protocols still suffer from several serious drawbacks. The first thing is that all non-faulty nodes must connect well with each other, which demands a huge number of communication channels in the network. Moreover, previous protocols have faced great loss in efficiency in the case where the honesty of each committee’s leader is in question. At the same time, no explicit incentive is provided for nodes to actively participate in the protocol.

We present CycLedger, a scalable and secure parallel protocol for distributed ledger via sharding. Our protocol selects a leader and a partial set for each committee, who are in charge of maintaining intra-shard consensus and communicating with other committees, to reduce the amortized complexity of communication, computation, and storage on all nodes. We introduce a novel semi-commitment scheme between committees and a recovery procedure to prevent the system from crashing even when leaders of committees are malicious. To add incentive for the network, we use the concept of reputation, which measures each node’s trusty computing power. As nodes with a higher reputation receive more rewards, there is an encouragement for nodes with strong computing ability to work honestly to gain reputation. In this way, we strike out a new path to establish scalability, security, and incentive for the sharding-based distributed ledger.

Index Terms—Distributed Ledger, Sharding, Scalability, Security, Reputation

I. INTRODUCTION

The blockchain revolution has established a milestone with Nakamoto’s Bitcoin [20] during the long search for a dreaming digital currency. It maintains the distributed database in a decentralized manner and has achieved a certain maturity to provide the electronic community with a service that captures the most important features of cash: secure, anonymity, easy to carry, easy to change hand, and exchangeable across a nation’s boundaries. At the same time, Bitcoin realizes the tamperproof of transactions.

Unfortunately, the transaction throughput of Bitcoin is 3 to 4 orders of magnitude away from those centralized payment-

processors like Visa, which stops its popularization. Specifically, Bitcoin can process only 3.3-7 transactions per second in 2016 [7] while Visa handles more than 24,000 transactions per second [25]. The low scalability of Bitcoin is the key factor causing this displeasing result.

Sharding is a famous technique in building the scale-out database by separating the whole state into multiple parts and making them work concurrently. Inspired by this idea, sharding is also used to benefit distributed ledgers in these years. When nodes are partitioned into groups, less computational power is wasted and higher processing capacity is achieved. However, most sharding-based protocols fail to maintain high efficiency when committee leaders betray [17] [27]. They also fail to give an explicit incentive for nodes to join the protocol and behave honestly.

We present CycLedger, a fully decentralized payment-processor that provides scalability over the number of nodes and provable security. Specifically, we expect our protocol to remain robust when leaders in each committee are faulty. This is a problem that remains unsolved until now. Meanwhile, we hope there is enough encouragement for a node to participate honestly in our protocol. Furthermore, scalability is realized via sharding nodes into concurrent committees as previous works have shown [17] [18] [27].

We assume there are no more than 1/3 malicious nodes at any time during the execution of our protocol, which is the best result achieved so far in sharding blockchain schemes [14] [27]. Meanwhile, an adversary can corrupt any nodes as he/she likes, but it requires some time for such corruption attempts to take effect. We select its members in each committee pseudorandomly so that with overwhelming probability, there are more than 1/2 honest validators in a committee. Furthermore, we construct a partial set in each committee. Members in this set supervise the leader’s behavior and act as alternates when the latter is confirmed to be vicious. By setting the size of each partial set to an appropriately large number, we ensure that only with negligible probability there are no honest nodes in a partial set. Finally, we design a recovery procedure to select a new leader when the original one is dishonest. Concretely, when a leader is found to violate the restriction of the protocol, he/she will be evicted and a node in the partial set will take his/her place. By introducing digital signature and semi-commitments between committees,

This research reported in this work was supported by the National Natural Science Foundation of China (No. 61761146005, 61632017).

we can guarantee that a non-void block can be successfully generated in each round with high probability.

To provide validators with enough incentive to enter our network, we introduce the concept of reputation to CycLedger. We hope one's reputation could be a good reflection of his/her computing power so that when we distribute the total revenue according to nodes' reputation, those honest nodes who contribute more to access transactions are paid more. To achieve this goal, we consider one's vote on a certain set of transactions. The more similar his/her vote is with the final decision, the more reputation he/she gains. In this way, as nodes with a higher computational resource can process more transactions correctly within a given time, they will obtain a higher accumulated reputation, consequently, winning more reward than other nodes.

The convenience reputation provides us is that we can simply sort out those nodes with strong computing ability. In CycLedger, those nodes with the best reputation are selected as leaders of each round, hoping they can use their abundant computational resources to bring more transactions into a block. In this way, a promotion on throughput is expected to see in CycLedger.

This paper is organized as follows. In Section II, we review previous works that are relevant to our protocol. In Section III, we state the network and threat models we use, define the problems we aim to solve and give an overview of CycLedger. We elaborate on our main protocol in Section IV. After that, we give the analysis on the security and incentive of our protocol in Section V and Section VII, respectively. Finally, we conclude the paper in Section IX.

II. BACKGROUND AND RELATED WORK

A. Sharding-based Blockchains

In traditional blockchain protocols, all nodes in the network have to agree on a certain set of transactions. This scheme leads to a very low throughput as only a rather fixed amount of transactions are included in a block regardless of the number of nodes in the network. An alternative way is to partition nodes into parallel committees and let each committee maintain the status of a certain group of users. Under this method, the amount of transactions is proportional to the number of committees rather than a constant. This technique is known as *sharding* and is considered as a good way to help blockchain scale well in literature.

RSCoin [9] implements a centralized monetary system via sharding, however, it is not decentralized and cannot transplant to distributed ledgers. Elastico [18] is the first sharding-based protocol for public blockchains which can tolerate up to a fraction of 1/4 of malicious parties. Unfortunately, it has a very weak security guarantee as the randomness in each epoch of the protocol can be biased by the *Adversary*. Meanwhile, Elastico's small committees (only about 100 nodes in a committee) cause a high probability to fail under a 1/4 adversary, and cannot be released in a PoW system [12]. Specifically, when there are 16 shards, the failure probability is 97% over only 6 epochs [17]. OmniLedger [17] also allows the adversary

to take control of at most 25% of the validators as well as assuming the adversary to be mildly-adaptive, nevertheless, it depends on the assumption that there is a never-absent trusty client to schedule the leaders' interaction when handling cross-shard transactions. RapidChain [27] enhances the efficiency of sharding-based blockchain protocols on a large scale, but the protocol guarantees high efficiency only when leaders of each committee are honest, an unrealistic assumption in practice. Concretely, in expectation, there is a proportion of 1/3 leaders that are malicious in a round. Under this condition, cross-shard transactions may hardly be included in a block. Furthermore, the protocol does not have an explicit incentive for nodes to participate in. At the same time, all the above postulate a good connection between any pair of truthful nodes, which causes a huge burden in creating connection channels.

We give a comparison of CycLedger with previous sharding blockchain protocols on several aspects in table I.

B. Distributed Randomness and Cryptographic Sortition

It has long been a critical task in blockchain to come up with a random beacon each round (or step, slot, epoch). Algorand [13] makes use of Verifiable Random Functions (VRFs) [19]. The seed of the current round, the round number as well as the leader's secret key are required to produce the next round's seed and a proof of it. In this way, when the leader is honest, the seed is pseudorandom and unpredictable. However, the beacon can be somehow biased when the leader is malicious. Ouroboros [16] takes the usage of the Publicly Verifiable Secret Sharing (PVSS) scheme [4] [23] to generate the random seed. This scheme is now known as a secure way to distributedly generate a random number and is also used in OmniLedger [17]. However, OmniLedger deploys RandHound [24] to generate the Common Reference String (CRS). Therefore, the protocol can only allow a 1/3 proportion of malicious nodes in each committee. At the same time, a trusty client is also required in RandHound. Other protocols [1] [10] use an external cryptographic hash function (CRHF), which takes an unpredictable and tamper-resistant value (*e.g.*, some values contained in previous blocks) as input. The output of the function is seen as the randomness of an epoch.

The technique of cryptographic sortition is widely used in blockchain protocols to select a small set of validators pseudorandomly. In Algorand [13], a node's VRF value on a given input is the ticket to enter the committee. At the same time, VRF also provides proof so that everyone can verify if a node is selected. We mention that many protocols [1] [10] now use this scheme to ensure pseudorandomness, verifiability, and unpredictability without loss of efficiency. Other protocols are utilizing different mechanisms. For example, the Sleepy Model [22] only uses pseudorandom functions while Ouroboros [16] randomly generates several coins and uses the Follow-the-Satoshi (FTS) algorithm [3] to find their owners who are deemed as leaders. However, these methods are either lack of security or not quite efficient.

TABLE I
A COMPARISON OF CYCLEDDER WITH PREVIOUS SHARDING BLOCKCHAIN PROTOCOLS

	Elastico	OmniLedger	RapidChain	CycLedger
Resiliency	$t < n/4$	$t < n/4$	$t < n/3$	$t < n/3$
Complexity	$\Omega(n)$	$O(n)$	$O(n)$	$O(n)$
Storage	$O(n)$	$O(c + \log(m))$	$O(c)$	$O(m^2/n + c)$
Fail Probability within a Round	$\Omega(me^{-c/40})$	$O(me^{-c/40})$	$me^{-c/12} + (1/2)^{27}$	$m(e^{-c/12} + (1/3)^\lambda)$
Decentralization	no always-honest party	an honest client	an honest reference committee	no always-honest party
High Efficiency w.r.t Dishonest Leaders	×	×	×	✓
Incentives	×	×	×	✓
Burden on Connection	heavy	heavy	heavy	light

¹ n : total amount of nodes in the network m : amount of committees c : amount of nodes in a committee, we mention that $n = mc$.

² λ : size of a partial set. Usually λ is set to be no less than 40.

³ When computing complexity, it is assumed that a cross-shard transaction be relative with all committees.

C. Reputation and Blockchain

The open, anonymous and decentralized environment of Peer-to-Peer (P2P) systems brings significant advantages, such as the strong expansibility and load balancing. But in the meantime, it provides great convenience for the self-interested and malicious users to take strategic behaviors, which could lead to the inefficiency or even failure of the system. For example, current P2P file-sharing networks suffer from the rife of inauthentic contents [6]. *Reputation*, regarded as a tool to record the action of users, has been widely used in the collaborative P2P applications to solve this problem. By designing a proper metric and corresponding algorithm, it could help the aforementioned file-sharing system effectively to identify malicious peers [15], avoid undesirable contents [26] [8] and drive cooperation among strategic users [11].

Inspired by the success of reputation in many other P2P systems, some researches try to introduce it into the area of blockchain. Nojournian *et al.* [21] proposes a new framework for the PoW-based blockchain, in which each miner has a public reputation value reflecting his historical mining behavior. One's reputation influences his opportunity to participate in future minings, thus, for their long-term interests, miners are encouraged to avoid dishonest mining strategies in this model. Repchain [14] first introduces the reputation mechanism into the sharding-based blockchain system. Reputation is designed to characterize validators' trust and activeness, based on their decisions on the transactions list. Repchain uses the accumulated reputation values for balanced sharding, leader selections, and transaction fee distributions. It is claimed that the schema provides a high incentive for validators to work hard and honestly, as well as improving the system performance. However, building a committee with the help of a reputation reduces its randomness. That is, it indeed trades security for better incentives.

III. MODEL, PROBLEM DEFINITION AND OVERVIEW

A. Notation

CycLedger works in rounds. In each round r , suppose there are n (n is changing as r changes, but to simplify the notation,

we use n instead of n^r) nodes in the network and each of them has a reputation $w_1^r, w_2^r, \dots, w_n^r$ which changes as the protocol executes. We use a Public-Key Infrastructure (PKI) to give each node a public/secret key pair (PK, SK). An honest-majority referee committee C_R^r is selected in round $r - 1$ to manage nodes' identities, produce next round's randomness R^{r+1} and propose round r 's block B^r . At the same time, all other nodes are partitioned into m committees which we denote as $C_1^r, C_2^r, \dots, C_m^r$. Note that we require each committee a trusty majority. Therefore, in our protocol, we would like the size of each committee to be $c = O(\log^2 n)$ in expectation. Readers should mention that $n = mc$. Each committee C_i^r , excluding the referee committee, includes a leader l_i^r , λ potential leaders and $c - \lambda - 1$ members. Potential leaders in C_i^r ($1 \leq i \leq m$) form the partial set of the committee, the latter is denoted as $C_{i,partial}^r = \{c_{i,1}^r, c_{i,2}^r, \dots, c_{i,\lambda}^r\}$. It is ensured that there is at least one non-faulty node in each partial set. In practice, λ can be an appropriate value, like 40. Fig. 1 demonstrates the hierarchical structure of different nodes and committees.

Besides, with overwhelming probability, each round is successfully terminated (*i.e.*, all expected operations are finished) within a fixed time T .

B. Network Model

We assume the good connection within a committee while all leaders and partial set members are linked. Furthermore, we suppose that each leader or partial set member is connected with the whole referee committee C_R . This requires a far less amount of reliable connection channels than other works [14] [17] [18] [27] in which they require a good connection among all honest nodes. Meanwhile, like existing works [17] [27], we assume synchronous communication within committees (*i.e.*, the delay of transporting every message is within some Δ) which is realistic in real-world as a committee only consists of several hundred nodes. Meanwhile, all leaders and partial set members are synchronously linked, however, with a larger time delay of Γ . Concerning other connections, we only need to assume partially-synchronous channels [2] [5] [17].

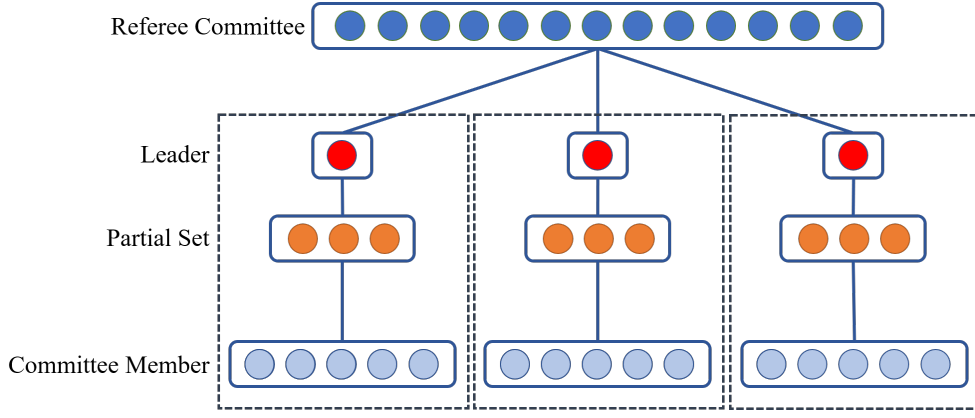


Fig. 1. Hierarchical structure.

C. Threat Model

As we use digital signatures in reaching consensus as RapidChain [27] and RepChain [14] do, we may assume the existence of a probabilistic polynomial-time *Adversary* which takes control of less than $1/3$ part of total nodes. Corrupted nodes may collude and act out arbitrary behaviors like sending wrong messages or simply pretending to be offline. The adversary can change the order of messages sent by non-faulty nodes for the restriction given in our network model, just like in classical BFT protocols [2]. Other nodes, known as *honest* nodes, always follow the protocol and do nothing exceeding the regulation. At the same time, we allow the adversary to be mildly-adaptive, which means that he/she is allowed to corrupt a set of nodes at the start of any round. Nevertheless, such corruption attempts require at least a round's time to take effect. Also, we assume all nodes in the network have access to an external random oracle H which is collision-resistant as well as a Verifiable Random Function (VRF) scheme.

D. Problem Definition

We assume that a large set of transactions are continuously sent to our network by external *users*. Users are almost equally divided into m shards. The status of each shard, including the users' identity and Unspent Transaction Outputs (UTXOs), is maintained by the corresponding committee. All processors have access to an authentication function V to verify whether a transaction is legitimate, *e.g.*, the sum of all inputs of the transaction is no less than the sum of all outputs and there is no double-spending. Our goal is similar to [18] [27] but slightly different. We seek for a protocol Π which, with a given set of transactions as input, outputs a subset, TX , such that the following properties hold:

- **Security.** In each round, the protocol fails at a probability no more than $2^{-\lambda}$, where λ is the security parameter.
- **Validity.** Each transaction in TX passes the verification, *i.e.*, $\forall tx \in TX, V(tx) = True$.
- **Scalability.** $|TX|$ grows quasi-linearly with n .
- **Incentive.** Nodes are awarded to execute the protocol within the given restriction.

We mention that CycLedger differs from previous protocols [17] [18] [27] as there is an explicit incentive design in it.

E. Protocol Overview

Roughly, in each round r , our protocol consists of the following phases.

- **Committee Configuration.** As the referee committee C_R^r , leaders and partial sets of round r are already determined in the previous round, in this phase, all other verified nodes are grouped and committees are formed.
- **Semi-Commitment Exchanging.** Each leader constructs a semi-commitment of all members in his/her committee and sends it to C_R^r , the partial set of his/her committee and all other leaders. Loosely speaking, the semi-commitment helps to detect a malicious leader when he/she tries to cheat in the phase of inter-committee consensus.
- **Intra-committee Consensus.** In this phase, non-faulty nodes within a committee reach an agreement on those transactions whose inputs and outputs only relate with the shard they are responsible for.
- **Inter-committee Consensus.** In short, all truthful nodes agree on the validity of cross-shard transactions, whose inputs and outputs are related to multiple shards. To achieve the goal, several steps are required, which will be discussed later in detail.
- **Reputation Updating.** Reputation is an important indicator for each node in CycLedger. The higher a node's reputation, the more rewards he/she will get. After the above consensus phases, the reputation of all members in a committee are updated according to their votes.
- **Referee Committee, Leaders and Partial Sets Selection.** We use a Proof-of-Work (PoW) process to figure out those nodes who will participate in the next round. After that, the referee committee and partial sets in round $r+1$ are uniformly selected while leaders in the next round are selected concerning the updated reputation. At the same time, C_R^r produces the randomness R^{r+1} for the next round.

- **Block Generation and Propagation.** After receiving all transactions, C_R^r verifies and packs the legitimate ones as well as next round's randomness, next round's participants, nodes' updated reputation, and all members in C_R^{r+1} , all leaders and partial set members into the block B^r , and releases it to all nodes. All nodes in a committee reach a consensus on the UTXOs they are in charge of after seeing B^r , and the leader will send them to C_R^{r+1} .

We roughly show how the protocol works when a transaction is submitted in Fig. 2.

IV. MAIN PROTOCOL

A. Committee Configuration

For simplicity, it is by default that all messages are sent authentically via the digital signature scheme throughout the protocol. We implicitly omit the signature verification in the description. Meanwhile, to avoid unnecessary tautology, a committee's leader and partial set members are referred to as *key members* of the committee. Other validators in the committee are also known as *common members*.

To begin with, we propose a cryptographic sortition mechanism using the Verifiable Random Function (VRF) scheme. Except for the pre-selected referee committee participants and key members, an undetermined node can find out the committee he/she belongs to via this method. Algorithm 1 takes node's public/secret key pair (PK, SK), round number r and round's randomness R^r as input, and returns a committee ID id and a proof π which certifies that the node belongs to C_i^r in the round.

Algorithm 1 Cryptographic Sortition

```

1: procedure CRYPTO_SORT( $PK, SK, r, R^r$ )
2:    $\langle hash, \pi \rangle \leftarrow VRF_{SK}(COMMON\_MEMBER||r||R^r)$ ;
3:    $id \leftarrow hash \bmod m$ ;
4: return ( $id, hash, \pi$ ).

```

For a non-key member i :

- 1) He/she determines his/her committee via Algorithm 1.
- 2) He/she sends his/her public key PK_i , address, VRF value $hash$ and VRF proof π to the key members of the committee whose addresses are already shown in block B^{r-1} .
- 3) When node i receives a list from a key member, he/she delivers his/her public key, address, the hash value together with the proof to all unconnected committee members on the list.
- 4) When i receives a PK_j from a validator j , first he/she checks if node j is in the same committee with the given proof and forms the link if right.

For a key member in committee C_k :

- 1) He/she maintains a $\langle PK, address \rangle$ list. Initially, the list contains the $\langle PK, address \rangle$ pairs of all key members in the committee.
- 2) When receiving a public key from node j , first he/she checks whether node j belongs to the committee via the

provided proof. If $j \in C_k^r$, he/she responds the current list back, and adds $\langle PK_j, address_j \rangle$ into the list.

Committees are formed in parallel as nodes independently execute the program shown in Algorithm 2. As a default, in all pseudocodes, the function *BROADCAST* implicitly suggests that the message is multicast to all known members in the committee. We note that in expectation, each committee is consist of $c = O(\log^2 n)$ nodes. To avoid complex notations, we sometimes use C_k, l_k and C_R instead of C_k^r, l_k^r and C_R^r in algorithms when there is no ambiguity. Meanwhile, typically, we use l to represent a leader, pm to represent a partial set member when the specific committee is unimportant and rm to represent a referee member.

Algorithm 2 Committee Configuration

```

1: procedure COMM_CONFIG( $r, B^{r-1}$ )
2: For any key member  $km$  in committee  $C_k$ :
3:    $S \leftarrow \bigcup_{i \in \{l_k\} \cup C_{k,partial}} \{ \langle PK_i, address_i \rangle \}$ ;
4:    $\Psi \leftarrow \emptyset$ ;
5:    $Q \leftarrow COMMON\_MEMBER||r||R^r$ ;
6:   upon DELIVER( $i$  | CONFIG,  $\langle PK_i, address_i \rangle$ 
,  $hash_i, \pi_i$ ) do
7:     if VRF_VERIFY $_{PK_i}(Q, hash_i, \pi_i) = TRUE$ 
then
8:        $S \leftarrow S \cup \{ \langle PK_i, address_i \rangle \}$ ;
9:        $\Psi \leftarrow \Psi \cup \{ \langle hash_i, \pi_i \rangle \}$ ;
10:      SEND( $i$  | MEM_LIST,  $S$ );
11: For any non-key member  $i$ :
12:    $S \leftarrow \emptyset$ ;
13:    $Q \leftarrow COMMON\_MEMBER||r||R^r$ ;
14:    $(id, hash, \pi) \leftarrow CRYPTO\_SORT(PK_i, SK_i, r, R^r)$ ;
15:    $(l_{id}, C_{id,partial}) \leftarrow B^{r-1}$ ;
16:   BROADCAST(CONFIG,  $\langle PK_i, address_i \rangle$ 
,  $hash, \pi$ );
17:   upon DELIVER( $km$  | MEM_LIST,  $S'$ ) do
18:      $S \leftarrow S \cup S'$ ;
19:     BROADCAST(MEMBER,  $\langle PK_i, address_i \rangle$ 
,  $hash, \pi$ );
20:   upon DELIVER( $j$  | MEMBER,  $\langle PK_j, address_j \rangle$ 
,  $hash_j, \pi_j$ ) do
21:     if VRF_VERIFY $_{PK_j}(Q, hash_j, \pi_j) = TRUE$ 
then
22:        $S \leftarrow S \cup \{ \langle PK_j, address_j \rangle \}$ ;

```

B. Semi-Commitment Exchanging

This phase starts at a given time after the first phase starts. The recommended delay is 8Δ .

We rely on Algorithm 3 which works efficiently to reach an agreement inside a committee. For simplicity, we omit the

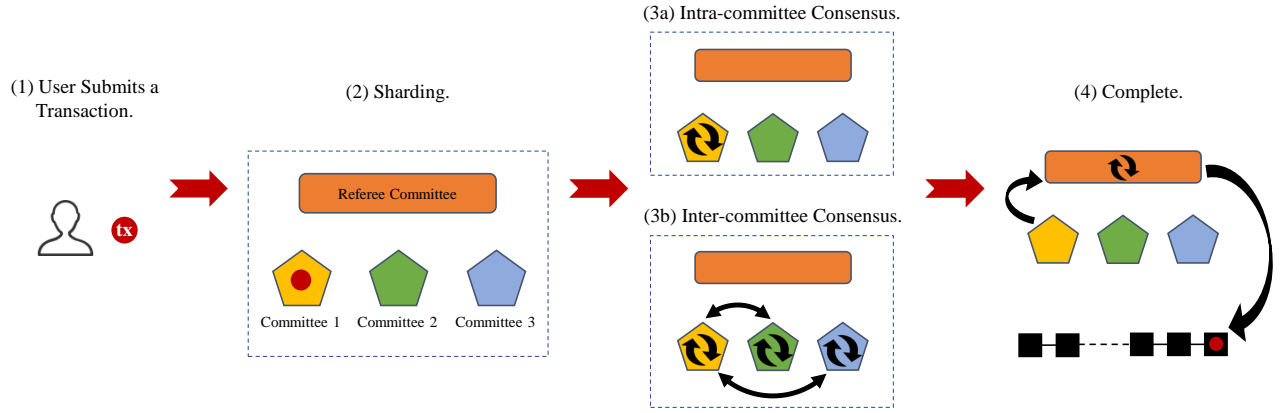


Fig. 2. CycLedger architecture overview: (1) When someone submits a transaction (tx, for short), (2) the transaction will be sent to the corresponding shard, which is in the charge of committee 1. (3a) If the inputs and outputs of the transaction all belong to committee 1, they reach an intra-committee consensus on it. (3b) Otherwise, they reach an inter-committee consensus with the help of other committees. (4) After that, the verified transaction will be sent to the referee committee, who verifies and packs it into the block.

message verification procedure in the algorithm. A demonstration of its process is in Fig. 3. Specifically, the algorithm contains three synchronous steps. In the first step, the leader multicasts the round number r , the original information M , the digest $H(M)$, and the sequence number of the message sn to each group member with the tag PROPOSE. Here, the sequence number sn is unique and monotonically increasing over time. And the digest helps to mitigate the burden on the channel in later steps. When a committee member i receives M and $H(M)$, he/she verifies the correctness of the digest, checks the round number and the uniqueness of the sequence number, and broadcasts $\langle r, sn, H(M), i \rangle$ with the tag ECHO as well as transmits the original PROPOSE message to all members in the committee. If i receives the identical ECHO and transmitted messages from more than half validators in the committee as well as the corresponding PROPOSE message from the leader, he/she gossips $\langle r, sn, H(M), i \rangle$ tagged by CONFIRM with all the authenticated ECHOes he/she received back to the leader. If any non-faulty node notices that the leader is malicious (*e.g.*, proposed different messages to different nodes), he/she informs all members of the committee immediately to stop the consensus process. A trustful partial set member then arouses a recovery procedure to evict the current leader and elect a new one. The recovery procedure will be explained in detail later.

Back to the phase, we only require the computational-binding property of a commitment scheme here. That is where the name "semi-commitment" comes from. The semi-commitment exchanging phase runs as follows.

- 1) To start, each committees leader l_k^r should unite the member list $S = \{PK_{k,1}^r, PK_{k,2}^r, \dots\}$ from all key members in the committee, and compute the committees semi-commitment via the external hash function H :

$$SEMI_COM_k^r = H(S).$$

Then he/she multicasts it together with the member list

to everyone in C_R^r . To prevent any means of cheating, he/she also delivers the latter with the belongingness certificate to the partial set $C_{k,partial}^r$.

- 2) After every participant in C_R^r receives semi-commitments from all committees, all trusty nodes in C_R^r apply Algorithm 3 to check i) all members in any list are registered; ii) all semi-commitments are valid. (To execute the algorithm, each node in C_R^r is regarded as the leader.) They transmit the set of valid semi-commitments to all key members and expel the cheating leaders afterward.
- 3) When a partial set member $c_{k,i}^r$ gets the semi-commitment $SEMI_COM_k^r$ from C_R^r , he/she verifies whether his/her committees semi-commitment corresponds with the member list S he/she receives from the leader. The list S should be no smaller than the set he/she locally maintains. Meanwhile, the certificate should be valid. Once a truthful partial set member notices any mismatch, he/she may invoke the recovery procedure to evict the current leader.

We show the phase in Algorithm 4, however, the verifying process executed by partial set members is omitted.

C. Intra-committee Consensus

The phase is a straight application of Algorithm 3.

- 1) In the beginning, after receiving some constant amount of transactions from external users, each leader l_k^r creates a $TXList$ whose inputs are all within the shard they are in charge of.
- 2) The leader l_k^r broadcasts his/her $TXList$ to everyone in the committee.
- 3) After receiving $TXList$, every member votes listed transactions with *Yes*, *No* or *Unknown*. When an honest node fails to judge a transaction within the given time, he/she should vote *Unknown*. Afterwhile, he/she forwards the voting list back to the leader.

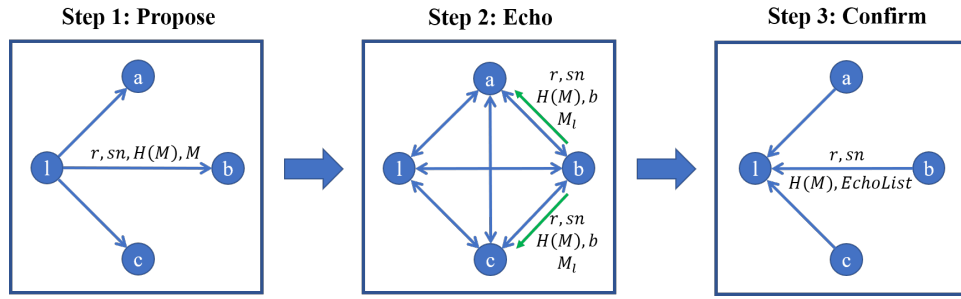


Fig. 3. A demonstration of inside-committee consensus.

Algorithm 3 Inside-committee Consensus

```

1: procedure INSIDE_CONSENSUS( $r, sn, M$ )
2: For leader  $l$ :
3:    $SigList \leftarrow \emptyset$ ;
4:    $BROADCAST(SIG_l < PROPOSE, r, sn, H(M) >$ 
   ,  $M$ );
5:   upon  $DELIVER(i \mid SIG_i <$ 
   CONFIRM,  $r, sn, H(M), i >, EchoList_i)$  do
6:      $M_{c,i} \leftarrow SIG_i < CONFIRM, r, sn, H(M), i >$ ;
7:      $v \leftarrow v + 1$ ;
8:      $SigList \leftarrow SigList \cup \{M_{c,i}\}$ ;
9:     if  $v > C/2$  then  $C :=$  the committee size.
10:  return  $SigList$ ;
11: For any member  $i$  (including leader  $l$ ):
12:    $EchoList \leftarrow \emptyset$ ;
13:    $v \leftarrow 0$ ;
14:   upon  $DELIVER(l \mid SIG_l <$ 
   PROPOSE,  $r, sn, H(M) >, M)$  do
15:      $M_{p,l} \leftarrow SIG_l < PROPOSE, r, sn, H(M) >$ ;
16:      $BROADCAST(SIG_i < ECHO, r, sn, H(M), i >$ 
   ,  $M_{p,l}$ );
17:   upon  $DELIVER(j \mid SIG_j < ECHO, r, sn, H(M), i >$ 
   ,  $M_{p,l})$  do
18:      $M_{e,j} \leftarrow SIG_j < ECHO, r, sn, H(M), j >$ ;
19:      $v \leftarrow v + 1$ ;
20:      $EchoList \leftarrow EchoList \cup \{M_{e,j}\}$ ;
21:     if  $v > C/2$  then
22:        $M_{c,i} \leftarrow SIG_i <$ 
   CONFIRM,  $r, sn, H(M), i >$ ;
23:        $SEND(l \mid M_{c,i}, EchoList)$ ;

```

Algorithm 4 Semi-commitment Exchange

```

1: procedure COM_EXCHANGE( $r, S, \Psi$ )
2: For leader  $l_k$ :
3:    $SEMI\_COM \leftarrow H(S)$ ;
4:    $ComList \leftarrow \vec{0}$ ;
5:    $ConfList \leftarrow \mathbf{0}$ ;
6:   for  $rm \in C_R$  do
7:      $SEND(rm \mid SIG_{l_k} <$ 
   SEMI_COM, SEMI_COM,  $r, S, k >)$ ;
8:   for  $pm \in C_{k,partial}$  do
9:      $SEND(pm \mid SIG_{l_k} <$ 
   SEMI_COM, SEMI_COM,  $r, S, \Psi >)$ ;
10:  upon  $DELIVER(rm \mid SIG_{rm} <$ 
   SEMI_COM, SEMI_COM $_j, r, j, rm >)$  do
11:     $ConfList[j][SEMI\_COM_j] \leftarrow$ 
    $ConfList[j][SEMI\_COM_j] + 1$ ;
12:    if  $ConfList[j][SEMI\_COM_j] > |C_R|/2$  then
13:       $ComList[j] \leftarrow SEMI\_COM_j$ ;
14:  For any referee member  $rm$ :
15:  upon  $DELIVER(SIG_{l_k} <$ 
   SEMI_COM, SEMI_COM $_k, r, S, k >)$  do
16:     $SigList \leftarrow Inside\_Consensus(r, < k, l_k >, <$ 
   SEMI_COM, SEMI_COM $_k, k >)$ 
17:    for each leader  $l$  do
18:       $SEND(l \mid SIG_{rm} <$ 
   SEMI_COM, SEMI_COM $_k, r, k, rm >, SigList)$ ;

```

4) After the leader obtains all voting lists, he/she picks up the set of transactions marked with a majority of *Yes*, which is named as $TXdecSET$. (Note that the collecting process should be within a certain time, e.g. 6Δ to avoid malicious nodes from indefinitely delaying. Those nodes who fail to reply in the period are deemed as voting

Unknown on all transactions. In the description of Algorithm 5, we tacitly approve that all members reply within the time limit.) Then he/she bales everyone's vote in a set named $VList$. The leader runs Algorithm 3 to reach consensus on both $TXdecSET$ and $VList$.

5) Finally, the leader sends $TXdecSET$ with at least half members' certification to C_R^r .

Readers can refer to Algorithm 5 for the execution of the phase.

Algorithm 5 Intra-committee Consensus

```
1: procedure INTRA_CONSENSUS( $r, TXList$ )
2: For leader  $l$ :
3:    $TXdecSET \leftarrow \emptyset$ ;
4:    $VList \leftarrow \vec{0}$ ;
5:    $TXSigNum \leftarrow \vec{0}$ ;
6:    $ReplyNum \leftarrow 0$ ;
7:   BROADCAST( $TX\_LIST, r, SIG_l < TXList >$ );
8:   upon  $DELIVER(i | VOTE, r, SIG_i < VList_i >)$  do
9:      $ReplyNum \leftarrow ReplyNum + 1$ ;
10:     $VList[i] \leftarrow VList_i$ ;
11:    for  $tx \in TXList$  do
12:      if  $VList_i[tx] = Yes$  then
13:         $TXSigNum[tx] \leftarrow TXSigNum[tx] + 1$ ;
14:        if  $TXSigNum[tx] > C/2$  then  $\triangleright C :=$ 
the committee size.
15:
16:         $TXdecSET \leftarrow TXdecSET \cup \{tx\}$ ;
17:
18:    upon  $ReplyNum = C$  do
19:       $SigList \leftarrow Inside\_Consensus(r, < l, INTRA >, <$ 
 $TXdecSET, VList >)$ ;
20:      for each  $rm \in C_R$  do
21:         $SEND(rm | SIG_l <$ 
 $INTRA, r, TXdecSET, VList >)$ ;
22:
23: For any member  $i$  (including leader  $l$ ):
24:   upon  $DELIVER(l | TX\_LIST, r, SIG_l < TXList >)$ 
do
25:      $VList_i \leftarrow Vote(TXList)$ ;
26:      $SEND(l | VOTE, r, SIG_i < VList_i >)$ ;
```

D. Inter-committee Consensus

We use the semi-commitment scheme introduced before to ensure the security of cross-shard communication.

Consider those transactions whose inputs and outputs scatter across two shards, which are maintained respectively by C_i^r and C_j^r . First of all, l_i^r should reach a consensus on such transactions within C_i^r by Algorithm 3. We call this list $TXList_{i,j}$. Then, the leader sends the consensus on $TXList_{i,j}$ as well as the member list to l_j^r and $C_{j,partial}^r$. Note that in this process, a faulty leader cannot fabricate a consensus result concerning the semi-commitment.

Resembling the last phase, C_j^r will reach an agreement on $TXList_{i,j}$. l_j^r then sends the consensus result back to l_i^r .

E. Reputation Updating

Once reaching the agreement on a voting of a $TXList$ (either an inner-committee list or a cross-committee one), the leader grades every group member. As previously described, $VList$ contains c marked votes, *i.e.*, all members' opinion on the validity of listed transactions. On the vote, someone marks *Yes* for those transactions he/she agrees, *No* for disagreed and

Unknown for the left. Facing the votes of all members, the leader scores each member according to the proximity between his/her opinion and the final decision.

Let $+1$, -1 and 0 represent *Yes*, *No* and *Unknown* respectively. Suppose the amount of transactions to be determined is D . Then there is a D -dimensional vector space and each transaction represents an axe of the space. From this aspect, a vote is seen as a vector in this space. Let $\vec{v}_i = \{v_{i,k} | k = 1, 2, \dots, D\}$ denote the vote of member i , where $v_{i,k}$ is the member i 's opinion on the k^{th} transaction. We use the cosine of the angle between two vectors to measure the proximity of two corresponding opinions. In other words, we define the *score* of one node as the cosine similarity between its voting vector and the resulting vector, which is determined by the majority algorithm and denoted by $\vec{u} = \{u_k | k = 1, 2, \dots, D\}$. Let s_i denote the member i 's score. We have

$$s_i = \cos(\vec{v}_i, \vec{u}) = \frac{\sum_{k=1}^D v_{i,k} u_k}{\sqrt{\sum_{k=1}^D v_{i,k}^2} \sqrt{\sum_{k=1}^D u_k^2}} \in [-1, 1]. \quad (1)$$

Owing to the random generation of each committee and the design of Algorithm 3, the majority result reflects truthful members' views. Here, the main idea is, the closer one's opinion stands with the consensus, the higher the grade he/she will get. Specifically, if a member has the same answers with the consentaneous results, he/she would be rewarded the highest score, *i.e.*, $+1$. On the contrary, if a member replies with completely opposing opinions, he/she will face a loss of -1 in reputation.

After calculating all scores, the leader assembles them into a $ScoreList = \{s_1, s_2, \dots, s_c\}$. Then he/she broadcasts $ScoreList$ and the original $VList$ to all members, waiting for the consensus by Algorithm 3. In this process, each non-faulty member should sign on the $ScoreList$. If successful, the leader sends the agreement to C_R^r , together with relevant certification. Then C_R^r updates their reputation by simply adding the listed score.

F. Referee Committee, Leaders and Partial Sets Selection

Participants in C_R^r distributedly generate next round's seed R^{r+1} via a random beacon generator. Here, the SCRAPE [4] scheme is preferred as it guarantees the pseudorandomness and unbiasedness of the seed even when the adversary takes control of almost half nodes. The nodes who want to participate in the next round need to solve a PoW puzzle in advance. The difficulty of the puzzle is appropriate and equal to everyone. Upon solving, a node is supposed to deliver the solution to the referee committee C_R^r and the latter will record his/her identity. As a result, by the end of the round r , C_R^r is aware of all next round's participants P^{r+1} . C_R^r chooses m nodes with the highest reputation as new leaders in round $r + 1$. Afterwhile, based on randomness R^r , C_R^r determines the next referee committee C_R^{r+1} and partial sets in the next round $\{C_{1,partial}^{r+1}, \dots, C_{m,partial}^{r+1}\}$. For example, a member in C_R^r can see if the following inequality holds for a node i :

$$H(r + 1 || R^r || PK_i || role) \leq d^r(role)$$

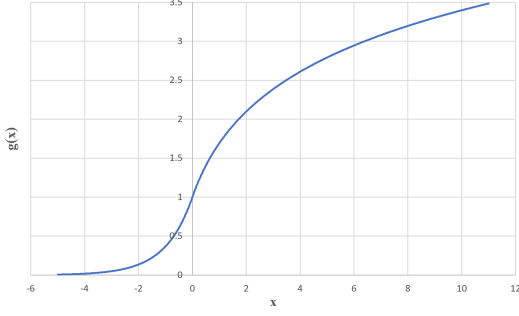


Fig. 4. The monotone function $g(x)$ mapping reputation to a positive number.

where $d(\cdot)$ is a difficulty function and $role$ is an optional string including `REFEREE_COMMITTEE_MEMBER` or `PARTIAL_SET_MEMBER`. A new $d^r(role)$ for either $role$ can be proposed every several rounds as the number of nodes in the network changes. If a node is selected as a partial set member, a member in C_R^r can calculate $H(r + 1 || R^r || PK_i || \text{PARTIAL_SET_MEMBER}) \bmod m$ to determine the committee he/she belongs to.

G. Block Generation and Propagation

Each time C_R^r members receive a $TXdecSET$ from a leader, they check the validity of its signature. By the end of the round, the referee committee comes to an agreement using Algorithm 3 on the set of valid $TXdecSETs$ and pack them up, together with all participants of next round S^{r+1} , their reputations W^{r+1} , the elected referee committee C_R^{r+1} , leaders and partial sets as a *block* B^r . By releasing it to the whole network, all nodes could obtain the contained information. After viewing the proposed block B^r , each committee member traverses all transactions packed in it. In this process, members delete the used ones from their local *UTXO Lists* and append the newly generated outputs that they are responsible for. Meanwhile, limited by the package size or other reasons, there may be some unpacked valid transactions within each committee, which form a *Remaining TX List*. The leader of each committee runs Algorithm 3 to reach a consensus on the final *UTXOs List* and *Remaining TX List*, and sends them to C_R^r . Next, C_R^r binds these lists with the *Committee ID* and forwards them to the corresponding new partial sets. Up to this point, all committees have finished their tasks.

Afterward, a node obtains part of the transaction fee based on his/her reputation. The sum of all nodes' revenue equals the fee of all transactions admitted in this round. Considering that one's reputation may be negative, we first map the reputation to a positive number using a monotone function $g(\cdot)$. Then rewards are distributed proportionally to the mapped value. Specifically, the function $g(\cdot)$ is designed as follows. The reputation of C_R^r will be updated by C_R^{r+1} in the next round.

$$g(x) = \begin{cases} e^x, & x \leq 0; \\ 1 + \ln(x + 1), & x > 0. \end{cases} \quad (2)$$

The monotone increasing function and proportional distribution ensure that whoever works more gets more. According to (2), $g(0) = 1$. Thus, nodes whose reputation is zero (e.g., nodes who always vote *Unknown*) could still get little rewards. By contrast, the negative reputation is mapped to near zero, which means the corresponding nodes can hardly obtain revenue. Under this reward mechanism, it is better to do nothing rather than do something bad, thus encouraging the malicious nodes to do right.

V. SECURITY ANALYSIS

In this section, we provide a comprehensive discussion on the security of our protocol, showing that CycLedger is highly secure with overwhelming probability.

A. Security on Randomness

In CycLedger, we apply the SCRAPE scheme [4] within C_R to distributedly generate a random string. SCRAPE guarantees that as long as the majority of nodes in C_R are honest, the output random string is pseudorandom and unpredictable. At the same time, no leader is required in the execution of SCRAPE. This feature suits the construction of the referee committee well as C_R is the only committee without a leader. As one can see in the following analysis, each committee, including C_R , has more than half of non-faulty nodes with high probability, hence, we assert that the randomness produced by SCRAPE with C_R is reliable.

B. Security on Committee Configuration

We say a committee is secure when more than half of nodes are non-faulty. Recall that committees are formed uniformly except leaders. Let X denote the number of malicious nodes in a committee, and c be the expected committee size. We consider the tail bound of hypergeometric distribution which gives the following result:

$$\Pr[X \geq \frac{c}{2}] = \sum_{x=\frac{c}{2}}^c \frac{\binom{t}{x} \binom{n-t}{c-x}}{\binom{n}{c}} \leq e^{-D(\frac{1}{2} || f)c}, \quad (3)$$

where $D(\cdot || \cdot)$ is the Kullback-Leibler divergence. Here $t < \frac{n}{3}$ and $f < \frac{1}{3} + \frac{1}{c}$, thus,

$$\Pr[X \geq \frac{c}{2}] \leq e^{-\frac{c}{12}}. \quad (4)$$

When the expected committee size is $c = O(\log^2 n)$, we derive that the probability that a committee is insecure is less than $n^{-\frac{\log n}{12}}$, which is negligible of n .

Fig. 5 visualizes (4). Namely, it shows the probability of failure calculated using the hypergeometric distribution to uniformly sample a committee when the population of the whole network is 2000. The amount of malicious nodes is 666, exactly less than one-third of the size of the network.

Particularly, when $c = 240$, the error probability for a single committee is less than 2.1×10^{-9} . Applying union bound, when m is less than 20, the error probability is no more than 5×10^{-8} .

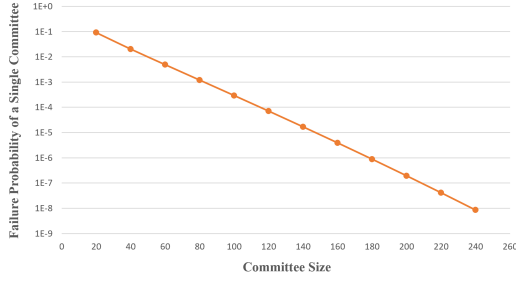


Fig. 5. Probability of failure in sampling one committee from a population of 2000 nodes in CycLedger. The amount of malicious nodes is set to 666.

C. Security on Partial Sets

We say a partial set is secure when at least one node in the set is honest. As no more than $1/3$ validators are faulty, when the size of the partial set is set to 40, the probability that a partial set is insecure at most:

$$\left(\frac{1}{3}\right)^{40} < 8 \times 10^{-20}.$$

Associated with union bound, when the number of committees is 20, the probability that at least one partial set is insecure is no more than 2×10^{-18} .

D. Security on Semi-Commitments

Recall that we use the hash of the member list as a semi-commitment of a committee. To start with, we show that $SEMI_COM_k^r = H(S)$ satisfies the computational binding property, where S is the member list.

Lemma 1. *When H is modeled as a collision-resistant hash function (CRHF), $SEMI_COM_k^r$ satisfies the computational binding property, i.e., once the semi-commitment is released, a probabilistic polynomial-time malicious leader cannot forge another member list which corresponds to the same semi-commitment with non-negligible probability.*

Proof. The lemma can be directly derived from the collision-resistance property of a CRHF. \square

With the above lemma, we come up with the following theorem.

Theorem 2. *A malicious leader cannot deceive a trustful leader by forging a member list of his/her committee as long as C_R has an honest majority.*

Proof. There are only two opportunities when a betrayer can lie to a loyalist on the member list. The first chance is when he/she tries to broadcast the semi-commitment. However, because all members in C_R , as well as the partial set members of the committee, see the exact member list, any false hash on the list will be perceived. Thus the liar will be detected. The other chance for the malicious leader is when the semi-commitment is revealed. However, according to the Lemma 1, misleading behavior will not take effect in this phase. \square

We emphasize that the hiding property of a commitment is not necessary for our protocol. For a vicious leader, even if he/she figures out the members of a committee in the semi-commitment exchanging phase, he/she can do nothing under our threat model as the adversary cannot get control of a trusty node immediately.

Here we introduce the leader re-selection procedure in CycLedger. The program is invoked when an honest partial set member notices that his/her leader is malicious or any participant of C_R notice that some leader is vicious. In the semi-commitment case, the event happens when a loyal party (C_R or a partial set member) discovers inconsistency between the member list he/she owns and the leader claims.

If a partial set member wants to accuse his/her leader, he/she would broadcast his/her *witness* to all members in the committee and ask them to vote on the impeachment. Here, a witness is a pair of messages $W = (m_l, m_0)$ where m_l should be sent and signed by the leader. We say a witness is *valid* if and only if the pair can derive dishonest behaviors of the leader. E.g., m_l be the member list that the leader sends, and m_0 be the semi-commitment of the committee where $m_0 \neq H(m_l)$. If the proposal is approved by more than half of the validators, the prosecutor will forward the voting result as well as his/her witness to everyone in the referee committee.

When any node in C_R receives a witness W and a signature list $Cert$ approving the prosecution from a partial set member pm from committee C_k , he/she starts Algorithm 6 to re-select a committee leader.

Algorithm 6 Leader Re-selection

- 1: **procedure** RE-SELECTION($r, pm, SIG_{pm} < k, W, Cert, pm >$)
 - 2: **For any referee member** rm :
 - 3: $SigList \leftarrow Inside_Consensus(r, < k, pm >, SIG_{pm} < k, W, Cert, pm >)$;
 - 4: **for each** $i \in C_k$ **do**
 - 5: $Send(i \mid SIG_{rm} < NEW, pm >, SigList)$;
-

Fig. 6 shows the above process. Afterwhile, the new leader needs to make a new semi-commitment of the committee via the semi-commitment exchanging protocol (Algorithm 4). When a participant of C_R receives the new semi-commitment, he/she informs every committee leader the new semi-commitment and leader's address, so that cross-shard transaction handling may start safely.

Now we claim that the given procedure is both complete and sound:

Claim 3. *A faulty leader is always detected and thus evicted via the leader re-selection procedure, as long as C_R has an honest majority.*

Proof. Note that there is at least one credible node in any partial set. Therefore, as a leader's action is always monitored by the partial set during the execution of the whole protocol (see further analysis in later subsections), any irregular be-

havior from the leader will be detected and a witness will be inevitably grasped by the non-faulty partial set member. At the same time, as the evidence is signed by the "criminal", no erroneous judgment will occur. \square

Claim 4. *A trustful leader will never be framed up by a faulty partial set member, as long as C_R has an honest majority.*

Proof. We mention that a witness is valid if and only if the first part of it is a message signed by the leader. For the security of the digital signature scheme, a vicious partial set member cannot counterfeit a shred of evidence as the latter has to be a leader's signed message. Therefore, a loyal leader will never be unjustly accused. \square

As proved above, the probability that more than half members in C_R are faulty is negligible. Thus, we claim that our recovering procedure remains complete and sound with high probability.

E. Security on Intra-committee Consensus

Resembling Claim 3 and 4, We assert that a faulty leader is always detected and expelled in the intra-committee consensus phase.

Theorem 5. *In Algorithm 5, a faulty leader can always be detected, meanwhile, malicious members can never calumniate a non-faulty leader.*

F. Security on Inter-committee Consensus

Finally, we show that cross-shard transactions are safely processed by our protocol.

Lemma 6. *A malicious leader who tries to imitate or conceal some cross-shard transactions is always detected by a trustful partial set member and thus evicted via the leader re-selection procedure.*

Proof. We prove this lemma conditioned on the fact that the messages are unforgeable. If the malicious leader imitates or conceals some cross-shard transactions, a trusty partial set member can challenge the leaders honesty by checking signatures from members of the departing committee. \square

Lemma 7. *A malicious leader can never frame up a trustful leader by misleading partial set members, as long as the delay of communication between shards is Γ .*

Proof. Notice that if the faulty leader tries to frame up a credible leader, the only way is to sending a message to the partial set members, meanwhile do not send anything to the honest leader, misleading that the non-faulty leader hides all transactions. However, if a non-faulty partial set member does not receive transactions from his/her leader after 2Γ time since he/she receives the transactions set from another committee leader, he/she can send the transactions set to his/her leader and continues running consensus protocol. \square

Combining the above two lemmas, we give the following theorem, which shows that our protocol guarantees security when processing cross-shard transactions.

Theorem 8. *In the inter-committee consensus phase, a faulty leader can always be detected, in the meantime, non-faulty leaders can never be framed up.*

Proof. This theorem can be derived by Lemma 6 and 7. \square

VI. PERFORMANCE ANALYSIS

We claim that our protocol is efficient in intra-shard and cross-shard transactions processing.

Recall that we use n to denote the number of nodes in the network, m to denote the number of committees and c to denote the expected amount of participants in a committee.

A. Complexity of Committee Configuration

In this phase, all members in any committee except C_R will recognize each other, which imposes a communication, computation and storage complexity of $O(c)$ for all common members. For leaders and partial set members, the communication complexity is multiplied by c as each of them has to deliver i pieces of information to the i^{th} applying participant.

B. Complexity of Commitment Exchanging

For any leader, he/she is obliged to produce the commitment of his/her committee and note down commitments from all other committees. Thus, the computation complexity is $O(c)$ while the storage complexity is $O(m)$. However, members in C_R has to suffer from the huge transportation overhead of $O(m^2)$ as they have to propagate all commitments to all committees as intermediaries.

C. Complexity of Reaching Intra-committee Consensus

To reach a consensus on a set of transactions with constant size, one in a committee has to broadcast the set with his/her signature to all members in a committee, which causes a communication complexity of $O(c)$. For key members, they must store the information with at least half of the members' certification to acknowledge the set, hence, a storage complexity of $O(c)$ is induced. At the same time, common members only need to keep their own opinion in reserve.

D. Complexity of Block Generation and Propagation

To propose a block with size $O(n)$ and broadcast it to all committees, an $O(mn)$ communicating burden and an $O(n)$ storage overhead are inevitable for any participant in C_R . However, we point out that the expense also exists in almost all previous protocols. At the same time, for any other attendee, the storage complexity is just $O(c)$, as he/she only needs to maintain the participants, transactions, and UTXOs within the committee he/she belongs to.

We summarize the theoretical analysis of the performance of CycLedger in Table II.

VII. INCENTIVE ANALYSIS

Besides security, reputation is also a highlight of CycLedger. There are two main problems when introducing reputation to our protocol: what reputation reflects and how reputation works. In this section, we give a discussion on the incentive of our protocol, by analyzing these two problems.

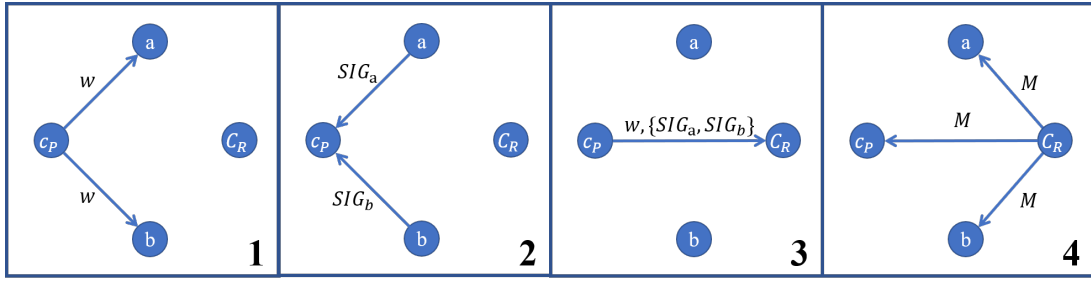


Fig. 6. This figure shows an example of the reporting mechanism and leader re-selection, where C_R is the *Referee Committee*, c_P is a partial set member, a and b are two committee members.

TABLE II
COMMUNICATION, COMPUTATION & STORAGE COMPLEXITY OF CYCLEDDGER

Communication & Computation / Storage Complexity	Common Members	Leaders & Partial Set Members	C_R Members
Committee Configuration	$O(c)/O(c)$	$O(c^2)/O(c^2)$	-
Semi-Commitment Exchanging	-	$O(c)/O(m)$	$O(m^2)/O(m)$
Intra-committee Consensus	$O(c)/O(1)$	$O(c)/O(c)$	$O(n)/O(n)$
Inter-committee Consensus	$O(m)/O(1)$	$O(n)/O(1)$	$O(n)/O(n)$
Reputation Updating	$O(c)/O(1)$	$O(c)/O(c)$	$O(n)/O(n)$
Key Member Selection	-	-	$O(n)/O(n)$
Block Generation & Propagation	$O(m)/O(c)$	$O(n)/O(c)$	$O(mn)/O(n)$

¹ n : total amount of nodes in the network m : amount of committees c : amount of nodes in a committee, we mention again that $n = mc$.

A. Incentive on Reputation

In general, reputation is expected to reflect the honest computational resources of each node in CycLedger.

The basic task of nodes in each committee, in short, is to give opinions on the validation of requested transactions. One's reputation is a reflection of his/her behaviors. For a newly joined node, based on his/her blank work experience, the reputation will start from zero. However, as long as he/she begins to work, the reputation matches his/her behavior. Specifically, for an honest node with more computing resources, he/she could make more correct judgments on the validation of transactions, thus, earning a higher reputation. While for the malicious nodes, reputation is closely related to his/her evilness, or in other words, the honest computational resources he/she contributes.

One's reputation determines his/her profit in each round. Nodes with a higher reputation get more payment after a new block is successfully generated. Owing to the scoring mechanism and reward mechanism, reputation provides enough incentive for nodes to work honestly and as hard as they can. In this way, reputation is considered as a reflection of the trustworthy computational resources one node contributes.

In CycLedger, leaders have a higher workload compared with other members. With this in mind, we directly choose nodes with the highest reputation as leaders in each round, thus to enhance the performance and throughput of CycLedger even further. In return, leaders obtain some extra reputation as a bonus for their hard work, which turns into higher

revenue. Therefore, leaders will have enough incentive to behave trustfully.

Certainly, a high reputation is not once and for all. Recalling the proportional design in our reward mechanism, one's revenue depends on the relative value of his/her reputation. Thus for each node, not to advance is to go back. So it appears that the best possible strategy for a node is, to use all his/her computational resources to work within rules.

B. Punishment on Reputation

We have been focusing on how the reputation motivates nodes to work hard. In this part, we will discuss what if someone, especially a leader, breaks the rules.

Intuitively, the scoring mechanism can be translated as, awarding points for right answers, deducting marks for wrong answers and doing nothing for an *Unknown* reply. Thus when giving wrong votes, intentionally or unintentionally, the node will face the corresponding decline in reputation. That is, the scoring mechanism itself contains the punishment on reputation.

Moreover, a leader who violates the protocol faces a more serious penalty. Once a leader is confirmed to commit a fault by the referee committee, his/her reputation will be decreased to the cube root. Recall that all leaders are those nodes with the highest reputation. We believe that the reputation of each leader is larger than 0, including malicious ones. Combining this punishment with (2), the mapped value, which is closely related to his/her revenue, will reduce to about one-third of the original mapped value. That is, the higher the reputation

a leader has, the stronger the punishment he/she will suffer when he/she behaves maliciously.

VIII. FUTURE WORKS

In this section, we introduce two skills that may enhance the efficiency of our protocol.

A. Excluding Low-value Transactions through Extra Communication

According to our protocol, if l_i^r wants to pack transactions that are related to UTXOs managed by C_j^r , l_i^r should pack up those transactions while nodes in C_i^r should run Algorithm 3 to generate a package *PACK*. Then nodes in C_j^r should also run Algorithm 3 again to confirm if *PACK* is valid. However, in some situations, most transactions in *PACK* may be invalid, for example, when the network suffers a Denial-of-Service (DoS) attack. In this case, this interaction process may be a waste of computational resources.

We hope that transactions chosen by l_i^r have a high probability to be accepted by C_j^r to enhance efficiency. One practical way is that leaders can communicate with each other before sending packages. For instance, l_i^r can enquire l_j^r which transactions are valid, and receives a preference directly from l_j^r rather than the agreement of C_j^r . This extra step of communication reduces the number of invalid transactions being packaged as long as both leaders trust each other.

We can set up a mechanism to mitigate the possibility that either leader lies. To achieve this, l_i^r can record the response of l_j^r and then generate a packet by Algorithm 3, including all transactions mentioned by l_j^r . Thus, if l_j^r lies to l_i^r on the validity of a transaction, he/she gets punished, such as a reduction on reputation.

B. Parallelizing Block Generation

In our protocol, C_R is in charge of proposing a block at the end of a round. This causes a huge connection burden on the referee committee. To boost efficiency, we can have each committee broadcast the block. In detail, after receiving enough authentication on a certain set of transactions from committee participants, a leader can forward the set to C_R for verification. After obtaining permission from the referee committee, the leader can immediately broadcast the block to the whole network. Applying this change to our protocol, we can adapt the schedule proposed by [17] in our protocol. We call two transactions are relevant if they follow one of the following properties:

- 1) They use the same UTXO as input;
- 2) One transaction spends the output of the other one.

Observe that those irrelevant transactions can be processed in parallel. Thus, a committee can sequentially produce and broadcast blocks within a round. As a result, two transactions satisfying the second property above may both be accepted in the same round. This event never happens in our original protocol as at least one of them will be regarded as illegal. Thus, by using this mechanism, we can enhance the efficiency and reliability of our protocol.

IX. CONCLUSION

We present CycLedger, a 1/3-resilient sharding-based distributed ledger protocol with scalability, reliable safety, and incentives.

By splitting nodes into parallel committees, we maximize the utilization of the computational resource, bringing high throughput to our protocol. We enable users to trade safely by introducing semi-commitments among committees and a recovery procedure to detect and evict faulty leaders. By evaluating each validator's behavior explicitly, our protocol has a considerable incentive for nodes to follow the instructions. At the same time, the reputation mechanism helps CycLedger locate those nodes with higher trusty computational power. By assigning them to high-workload positions, we further enhance the efficiency of the protocol. Finally, our analysis demonstrates that CycLedger has a nice performance and can provide striking security.

REFERENCES

- [1] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
- [2] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.
- [3] Cardano. Cardano settlement layer documentation. leader selection in cardano sl. <https://cardanodocs.com/technical/leader-selection>.
- [4] I. Cascudo and B. David. SCRAPE: scalable randomness attested by public entities. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Proceedings*, pages 537–556, 2017.
- [5] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186, 1999.
- [6] N. Christin, A. S. Weigend, and J. Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce (EC-2005)*, pages 68–77, 2005.
- [7] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains - (A position paper). In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Revised Selected Papers*, pages 106–125, 2016.
- [8] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002*, pages 207–216, 2002.
- [9] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016*, 2016.
- [10] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 66–98, 2018.
- [11] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC-2004)*, pages 102–111, 2004.
- [12] A. Gervais, G. O. Karame, V. Capkun, and S. Capkun. Is bitcoin a decentralized currency? *IEEE Secur. Priv.*, 12(3):54–60, 2014.
- [13] 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*, pages 51–68, 2017.
- [14] 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. *CoRR*, abs/1901.05741, 2019.

- [15] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International World Wide Web Conference, WWW 2003*, pages 640–651, 2003.
- [16] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Proceedings, Part I*, pages 357–388, 2017.
- [17] 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 2018, Proceedings*, pages 583–598, 2018.
- [18] 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, CCS 2016*, pages 17–30, 2016.
- [19] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 120–130, 1999.
- [20] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.
- [21] M. Nojoumian, A. Golchubian, L. Njilla, K. Kwiat, and C. Kamhoua. Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm. In *Proceedings of the 2018 IEEE Computing Conference, CC 2018, Volume 2*, pages 1118–1134, 2018.
- [22] R. Pass and E. Shi. The sleepy model of consensus. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Proceedings, Part II*, pages 380–409, 2017.
- [23] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology - CRYPTO'99, 19th Annual International Cryptology Conference, Proceedings*, pages 148–164. Springer, 1999.
- [24] E. Syta, P. Jovanovic, E. Kokoris-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 2017*, pages 444–460, 2017.
- [25] Visa. Visa's transaction per second. <https://usa.visa.com/run-your-business/small-business-tools/retail.html>.
- [26] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), Proceedings*, 2006.
- [27] 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, CCS 2018*, pages 931–948, 2018.