HotDAG: Hybrid Consensus via Sharding in the Permissionless Model

Chun-Xuan Zhou, Qiang-Sheng Hua(✉), and Hai Jin

National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
{chxzhou,qshua,hjin}@hust.edu.cn

Abstract. A major design to improve scalability and performance of blockchain is sharding which maintains a distributed ledger by running classical Byzantine Fault Tolerance (BFT) protocols through several relatively small committees. However, there are several drawbacks with the existing sharding protocols. First, the sharding mechanism which ensures that each committee is strongly bias-resistant either weakens the decentralization or reduces the performance of the protocol. Second, BFT protocols are either unresponsive or take quadratic communication complexities under a byzantine leader. Third, they cannot defend against transaction censorship attacks. Finally, nodes do not have enough motivation to follow the protocol and selfish nodes can obtain more rewards through collusive behaviors. A recent study proposes HotStuff –a BFT protocol that achieves linear view-change and optimistic responsiveness. In this paper, we present HotDAG, a hybrid consensus protocol based on HotStuff via sharding in the permissionless model. By employing the parallel Nakamoto consensus protocol, we present a decentralized and bias-resistant sharding mechanism. HotDAG has a linear communication complexity on transaction confirmation by introducing a scalable BFT protocol and an inter-committee consensus mechanism based on blockDAG. By achieving an unpredictable leader rotation, HotDAG prevents the censorship attacks. At the same time, HotDAG provides an incentive mechanism that is compatible with the scalable BFT protocol to encourage nodes to actively participate in the protocol. Finally, we formally prove the security and analyze the performance of HotDAG.

Keywords: Blockchain · Consensus · Byzantine Fault Tolerance · Scalability · Sharding · blockDAG.

1 Introduction

Since the advent of Bitcoin [1] in 2008, cryptocurrency and the underlying technology behind Bitcoin (blockchain) have attracted widespread attention in the finance and academia. The blockchain is essentially a distributed database system that provides a decentralized, open, and Byzantine fault-tolerant protocol. Each participant maintains a distributed ledger which provides the total order of transactions and runs a distributed consensus protocol (Nakamoto consensus) to ensure the consistency of the distributed ledger. Although the blockchain is a fully decentralized and securely designed protocol, it still faces scalability barriers such as low transaction throughput and high latency. Bitcoin and Ethereum process at most 7 transactions per second (tx/s) and 25 tx/s respectively, which is far from meeting the actual market demand. Therefore, many recent research efforts such as [2–6] have been devoted to scaling the Nakamoto consensus.

Variants of Nakamoto consensus still suffer from low throughput and high latency due to the probabilistic consistency and decentralization. It is an inherently tradeoff between security and performance. In contrast, classical BFT protocols have deterministic consistency and the period of blocks is much shorter. Thereby they obtain higher transaction throughput and lower latency. However, these protocols work in a permissioned setting where the set of participants is fixed and the identity of each participant is known (we define the participant as the validator, and the set of validators as a committee). It is easy to be broken in a permissionless setting when suffering from the sybil-attack [7] and it can't run in the dynamic network [9, 10]. Therefore, several hybrid protocols that combine the classical BFT protocol with Proof-of-Work (PoW) [8] or other proof-based protocols have been proposed [11–15]. However, BFT protocols have poor scalability, that is, an increase in the size of committee reduces transaction throughput. It motivated a design of protocols based on multiple committees so that transaction throughput scales linearly as the network size increases [16–19]. These protocols are also known as sharding-based protocols.

Unfortunately, there are some limitations in previous sharding-based protocols. First, to ensure that committees are bias-resistant, these protocols need to run a distributed randomness generation protocol to generate a seed for sharding securely (assign validators to committees). It causes additional communication overhead. In addition, it will cause temporary inoperability of the system for sharding protocols that use threshold signature schemes. Second, these protocols usually remain a fixed leader unless it fails. It makes them more vulnerable to transaction censorship (transactions are suppressed by the malicious leader) and the cost for the leader replacement to take a quadratic communication complexity. Other protocols [18–20] follow the leader rotation regime. However, they might be risky to deploy over the Internet because of the synchronous network assumption, and they forego the optimistic responsiveness. Responsiveness requires that an honest leader can drive the protocol to consensus in time depending only on the actual delay of the network. Third, cross-shard transactions rely on honest clients or leaders and require multiple committees to run the BFT protocol to resist the double-spend attack. Finally, previous work does not give incentive for nodes to participate in the protocol very well or causes serious resource monopoly problems.

To solve the above issues, we introduce HotDAG. It is a novel sharding protocol that ensures security in the perminssionless model. At the same time, HotDAG is scalable and strongly censorship-resistant. First, by introducing a parallel Nakamoto consensus protocol that is proposed by OHIE [6], HotDAG chooses a set of representative validators periodically via Proof-of-Work. Then HotDAG automatically and securely assigns validators to committees without any distributed randomness generation protocol or any honest third party. Second, HotDAG builds on the HotStuff [21] and provides an unpredictable leader rotation to guarantee censorship-resistance. Then, HotDAG extends the blockchain to the directed acyclic graph(blockDAG) and provides a safe total-ordering protocol that all honest nodes are agreed upon the total order of transactions. Finally, we introduce an incentive mechanism that is based on the epoch reputation to provide validators with enough motivation to participate in consensus. The rewards are distributed through the current committee's dynamic block reward coefficient and the node's contribution to processing transactions. It solves the problems of collusive behaviors and resource monopoly. Assume that any node outputs a total ordering of transactions at any time, we define it as a LOG. HotDAG achieves the following properties:

- **Decentralization**: Our protocol runs in a permissionless setting and does not rely on any trusted third party.
- **Consistency**: Suppose that an honest node p_i outputs LOG at time t and an honest node p_j outputs LOG' at time t' (i may equals to j and t may equals t'). Then with high probability,

either LOG is prefix of LOG' or LOG' is prefix of LOG. We use $\prec$ to represent the relationship of the prefix.
- **Scalability**: The throughput increases linearly with the number of committees.
- **Strongly censorship-resistance**: The probability that a malicious validator is selected as the leader is equal to the percentage of malicious validators in a committee.
- **Incentive**: Rational nodes have incentive to follow the protocol and selfish nodes will not get more rewards through collusion.

The remainder of this paper is organized as follows. In section 2, we review the related work. In section 3, we give the system, the network and the adversary models. The protocol details are present in section 4. Then, we formally prove the security of HotDAG in section 5. In section 6, we analyze the theoretical performance and incentive. Finally, we conclude the paper in section 7.

2 Related Work

In this section, we will review hybrid consensus protocols and sharding-based protocols.

Due to the strong consistency of BFT protocols, several hybrid consensus protocols [11,13] have been proposed to scale up the throughput. Usually, hybrid consensus protocols select committees by PoW and transactions are confirmed by the PBFT protocol. In addition, there are other hybrid consensus protocols based on Proof-of-Stake(PoS), including Tendermint [20], Alogorand [14].

Although hybrid consensus protocols significantly improve performance over Nakamoto consensus, there still exists a major limitation: the size of committee is not scalable. It inspires protocols based on multiple committees that allow multiple committees to process transactions in parallel. Elastico [16] is the first permissionless blockchain protocol based on transaction sharding. Elastico has a hierarchical committee topology where normal committees propose blocks, and the final committee combines blocks received from normal committees. Based on this work, some protocols that adopt the state sharding have been proposed, such as [17] and [18]. They have a flat committee topology that all committees are at the same level. However, these protocols need to deal with cross-shard transactions. [17] depends on the assumption that honest clients participate actively when handling cross-shard transactions. And [18] relies on honest leaders, which is an unrealistic assumption in practice. CycLedger [19] is the reputation-based sharding protocols. Unfortunately, they both face serious reputation monopoly issues. We present a comparison of HotDAG with previous sharding blockchain protocols in Table 1.

3 System Overview

3.1 System Model

HotDAG adopts the UTXO (unspent transaction outputs) model which is similar to Bitcoin. Our system has a public key infrastructure (PKI) and each node holds a public/private key pair. We adopt a double-layer architecture that consists of the identity chain and the transaction chain. Multiple transaction chains form a blockDAG. At a high level, HotDAG has a flat committee topology. All blocks generated by a committee constitute a transaction chain. Blocks on all transaction chains and references between blocks together form a blockDAG. Consider that each committee adds blocks to the corresponding transaction chain, we will use the transaction chain to illustrate our protocol in subsequent sections, instead of using the blockDAG.

Table 1: COMPARISON BETWEEN HOTDAG AND THE EXISTING WORKS

	Elastico	OmniLedger	RapidChain	CycLedger	HotDAG
Resiliency	1/4	1/4	1/3	1/3	1/4
Responsive	√	√	×	×	√
Decentralizaton[1]	√	×	×	×	√
Anti-censorship	×	×	×	×	√
Incentive	×	×	×	√	√
Operability[2]	×	×	×	×	√

[1] There are no always honest parties in Elastico and HotDAG. OmniLedger, RapidChain and CycLedger rely on honest clients, honest reference committee and honest leaders, respectively.
[2] The system can process transactions during the transition phase when the threshold signature scheme is used.

3.2 Network Model

We assume a δ-partially synchronous network model as in HotStuff. After an unknown Global Stabilization Time (GST), any message broadcast by an honest node at time t will arrive at all honest nodes at time $t + \delta$ (δ is known). In addition, we assume a priori loose upper bound Δ of network delay. The confirmation time of a block only depends on the network's actual delay δ.

3.3 Adversary Model

We adopt the adversary model which is the same as [13]. We regard Byzantine nodes as being controlled by an *adversary*, denoted as $\mathcal{A}$. Honest nodes strictly follow the protocol while $\mathcal{A}$ can run away from protocol, such as reordering messages and delaying up to the maximal network delay δ. We stress that $\mathcal{A}$ cannot drop or modify messages by honest nodes. Honest nodes may be corrupted due to the adversary's attack. We adopt a τ-corruption model [13] where it takes τ time for honest nodes to be corrupted after the attack. We assume that $\mathcal{A}$ can have up to 25% of the total hashpower at any given moment.

3.4 Cryptographic Primitives

Our protocol makes use of a threshold signature scheme [22–24]. The (n, k)-threshold signature scheme can tolerate up to $n - k$ malicious nodes when there are n nodes. The node can generate a partial signature by using the private key and can combine a signature of the message m with k partial signatures on m. The signature can be verified by the single public key. It is computationally infeasible to forge a signature or to modify a signature to match a modified message.

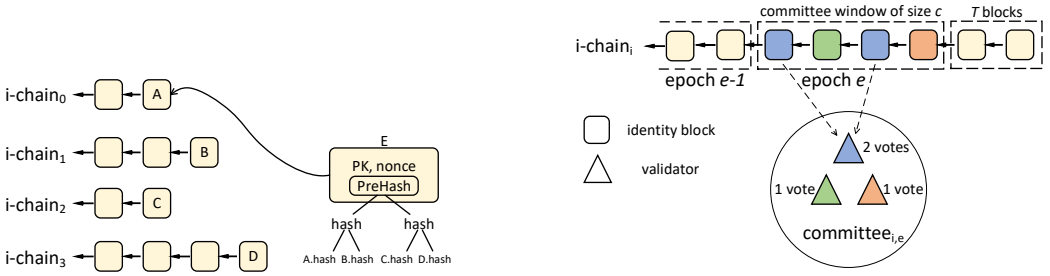
4 Main Protocol

4.1 Identity Establishment and Committee Configuration

To ensure a negligible probability that any committee is compromised, committees need to be re-formed periodically (we assume that committees are re-formed per epoch). To reduce the communication overhead and to make committees re-form in an independent epoch, we aim to extend

the identity chain in [17] to parallel identity chains. In this way, validators are automatically assigned to committees while identities are established. To securely assign validators to committees, we need to ensure that $\mathcal{A}$ gains no significant advantage in trying to bias its computational power towards any identity chain. We adopt a parallel Nakamoto consensus model to force $\mathcal{A}$ to evenly split its power across all identity chains and to guarantee the safety of each committee.

In (λ, p, T)-Nakamoto consensus, there is a security parameter λ and a mining hardness parameter p which is the probability of successful execution of a random oracle query. All blocks on blockchain except the last T blocks are confirmed. Nodes iterate through a nonce which makes the hash digest of a block to include a certain number of leading zeros, and the length of the hash is λ. We assume that there are k identity chains corresponding to k different committees. In HotDAG, we define that the hash of a block is $\mathcal{H}(preHash|nonce|PK)$ where $\mathcal{H}$ is a cryptographic hash function ($\mathcal{H}$ is modeled as random oracles in the analysis), $preHash$ is the hash of the last block that is determined by the Merkle Tree, $nonce$ is a random number and PK is the public key. It inputs hashes of the last blocks on k identify chains as leaves, and outputs the root's hash as $preHash$. The hash of a valid block should have $log_2 \frac{1}{kp}$ leading zeros. We use the last log_2^k bits of the block hash to specify the id of the identify chain. As shown in Fig. 1a, the last blocks on all identity chains are A, B, C, D. When a node mines the block E and we assume that the last log_2^k bits of $E.hash$ is 0, then the block E is added to $i\text{-}chain_0$. For any identity chain, the probability of successful execution of a random oracle query is the same as in Nakamoto consensus ($log_2 \frac{1}{p} = log_2 \frac{1}{kp} + log_2^k$) [6].



(a) Merkle Tree

(b) Committee configuration.

Fig. 1: Structure of identity chains.

From *Lemma* 3 in [6], the existing properties on (λ, p, T)-Nakomato consensus can directly carry over to each identity chain. Therefore, it guarantees that the proportion of malicious validators in each committee is less than $\frac{1}{3}$ with high probability. The committee reconfiguration is triggered when the identity chain grows by c blocks (c is the committee size), as shown in Fig. 1b. We define $i\text{-}chain_i[:-T]$ to denote the chain that removes the last T blocks on the $i\text{-}chain_i$. Let's take the switch from $committee_{i,e-1}$ to $committee_{i,e}$ as an example. When the height of $i\text{-}chain_i[:-T]$ is $(e+1) * c$, miners in the committee window form a new committee to take place of $committee_{i,e-1}$. Validators in $committee_{i,e}$ send *stop* messages to validators in $committee_{i,e-1}$. The color of the identity block indicates different miners. Validators' shares or voting power are directly proportional to their commitment to hashpower in the *committee window*. When the epoch $e-1$ is terminated, $committee_{i,e}$ starts processing transactions by running the intra-committee consensus protocol.

4.2 Intra-committee Consensus

Hotstuff works in a succession of views which are monotonically increasing. The leader of a view needs to collect votes in three phases: *pre-commit*, *commit* and *decide*. . We refer to a quorum of $2f+1$ votes over a given tuple$< view, block >$ as a quorum certificate (QC), and it is generated by the threshold signature scheme. The QC generated in *pre-commit* and *commit* phases are respectively denoted as *prepareQC* and *lockedQC*. A leader starts *prepare* phase when it collects $2f+1$ *new-view* messages with *prepareQC* and the leader selects the *prepareQC* with the highest view number as *highQC*. Then the leader broadcasts a new block with *highQC* and enters the *pre-commit* phase. The number of views is incremented by finishing all voting phases or by triggering a timeout of any phase.

It takes three similar voting phases for a leader to commit a block. Therefore, [21] also proposes the Chained-HotStuff to pipeline these phases. Chained-HotStuff works in a succession of rounds. The leader collects votes for a block b and proposes a new block b', and the work of collecting votes for b' will be handed over to the next leader. Votes for the block b' also serve as votes for the second phase of b. The Chained HotStuff requires only one round of communication on average to reach a consensus on a block. Blocks are added to the blockchain first, and will not be confirmed until the expansion of three blocks (due to three voting phases). Its chain structure is similar to Bitcoin, as shown in Figure 2. $QC.block$ means that QC is generated by enough votes for *block* and the round of QC is the same as the round of $QC.block$. Each validator keeps track of the following state variables: (1) qc_{high}: the QC with the highest round, the validator updates it when it receives a new block(or QC); (2) b_{lock}: the lastest block that finished the second phase; (3) b_{commit}: the lastest block that finished all voting phases. (4) v: the round of last voted block. For example in Fig. 2, $B_0.round = r_0$, $B_1.round = r_1$, $B_2.round = r_2$. We have $qc_{high} = B.QC$, $b_{lock} = B_1$, $b_{commit} = B_0$.

Our intra-committee consensus protocol is based on the Chained- HotStuff. We extend the chain structure of Chained-HotStuff to k chains and each validator maintains k sets of state variables. We define $t\text{-}chain_i$ as the i-th transaction chain. We consider a committee consisting of $c = 3f+1$ validators where f is the number of Byzantine validators and each round has a unique dedicated leader. We add the termination condition for timely termination [13] to ensure that the current committee will not generate any valid block upon all honest validators reach a consensus on entering the next epoch.

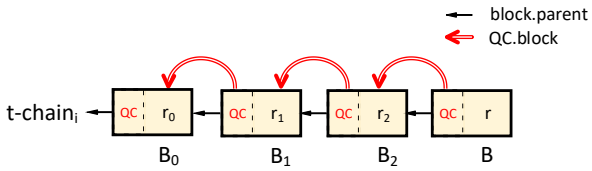


Fig. 2: Structure of Chained-HotStuff.

Protocol Specification. The intra-committee consensus protocol has four phases:

1. Startup phase: At the beginning, the validator sends a *new-view* message to the leader, including its qc_{high}. Then the committee enters the normal phase.

2. Normal phase: Upon receiving enough votes or *new-view* messages, the leader generates a QC. The leader selects transactions according to the transaction hash and proposes a new block b. Then it sends b to validators in the committee(including itself). At the same time, the leader also broadcasts QC and the block in QC to other committees. Upon receiving b and b is valid, the validator sends a vote (with qc_{high}) to the next leader and enters the next round.
3. Timeout phase: To guarantee the liveness, honest validators cannot keep the same round all the time. The validator starts a timer T_1 when it enters a new round. When T_1 expires, it sends a *new-view* message with qc_{high} to a new leader and enters the next round. We make use of an exponential back-off mechanism. The timer doubles value every time it expires [25].
4. Stop phase: When the leader receives enough *stop* messages from the next committee, the leader proposes a *stop* block and the current committee enters the stop phase. When the *stop* block finishes all voting phases, the current epoch is terminated and the next committee enters the startup phase.

Voting rules. In the normal phase, validators check the validity of a block according to the following rules:

1. The current epoch is not terminated.
2. Each transaction is valid: the sum of outputs is less than the sum of inputs and all inputs are outputs of some transactions on transaction chains.
3. The round of b is greater than the round of the last voted block, that is $b.round > v$.
4. $b.parent.round \geqslant b_{lock}.round$.

Partially-Committed rule. Consider four blocks B, B_0, B_1 and B_2 such that $B_2 = B.parent$, $B_1 = B_2.parent$ and $B_0 = B_1.parent$. If $B_2.round = B_1.round + 1$ and $B_1.round = B_0.round + 1$, we consider the branch led by B_0 is partially-committed. For example, blocks that on the branch led by B_0 are partially-committed in Fig. 2.

Leader selection. To avoid predicting the leader in advance, our protocol makes use of a verifiable random function (VRF) [26] and a pseudo-random function (PRF) to select the leader randomly. While proposing a block b, the leader also generates a seed $s = VRF_{sk}('leader'||e||r)$ and a proof π where sk is the private key of the leader, e is the current epoch and r is the round of $b.QC$. Anyone can verify the seed using the leader's public key and the proof π. Each validator keeps track of the latest seed s and calculates the leader'id of round n by $id = PRF(s, n)\ mod\ c$. In order to ensure that validators will not keep different seeds for a long time (it will cause validators' views on the next leader to be different), we need to setup a timeout T_2 that is used by the validator to synchronize the latest seed. The first block on the transaction chain contains the initial seed.

4.3 Inter-committee consensus

In intra-committee consensus, transactions are partitioned to different committees and multiple committees process transactions in parallel. To detect conflicting transactions in partially-committed blocks to prevent the double-spend attack, each validator needs to output a total-ordering of all blocks. A simple approach to order all blocks is to include the first block of each chain into LOG, and then the second block, etc. That is, $LOG = \{b_{0,0}...b_{k-1,0}, b_{0,1}...b_{k-1,1}...\}$. However, OHIE points out that the difference in the block growth rate among the k transaction chains will cause that some blocks may not be confirmed for a long time. Our protocol is diffierent

from OHIE where the block in OHIE is in a random manner to extend the transaction chain, but this difference still exists in HotDAG due to malicious leaders and the committee reconfiguration.

OHIE provides a total-ordering protocol to solve this issue. However, OHIE may violate happen-before relationships between transactions. In OHIE, each block has a tuple$< rank, nextRank >$. OHIE removes the last T unstabilized blocks and selects the minimum value of the $nextRank$ of the last block in the remaining blocks as $confirm\_bar$. Blocks whose $rank$ is smaller than $confirm\_bar$ can be included in the total order and blocks are ordered by $rank$, with tie-breaking favoring smaller chain ids. In Figure 3, there are two chains and we set T = 2 in this figure. At time t_1, there are five blocks on $Chain$ 1, and the last block contains a transaction $Tx1$. At time t_2, three new blocks are added to $Chain$ 0, and the first block contains the transaction $Tx2$ which spends the output in $Tx1$. At time t_3, we have $confirm\_bar = 5$ according to the confirmation rule of OHIE, and the total order is {B00,B10,B01,B11,B12,B13,B14}. Therefore, $Tx2$ is before $Tx1$ which makes $Tx2$ invalid.

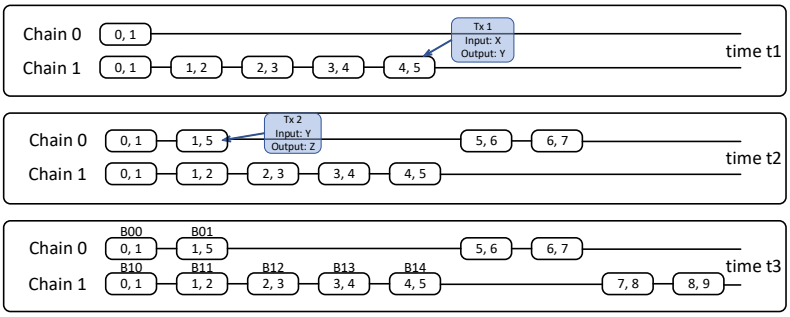


Fig. 3: Structure of OHIE.

To ensure that each non-conflicting transaction can be successfully confirmed, we extend parallel blockchains to a directed acyclic graph (DAG) where blocks specify the happen-before relationships among transactions. Each block b contains three fields $< height, nextHeight, references >$ where $height$ is the height of b and $nextHeight$ represents the $height$ of the next block to compensate for the gap between the current transaction chain and the longest transaction chain, that is, $nextHeight$ is the largest $nextHeight$ of the lastest blocks on transaction chains. $references$ is a set of block hashes and it represents the happen-before relationships among transactions. The height of b should be greater than all reference block, that is, $b.height = max(b.parent.nextHeight, \{r \in b.references | r.height + 1\})$. It should be noted that the cyclic dependencies between blocks do not occur. For example, we define that the transaction $Tx1$ depends on $Tx2$ and the transaction $Tx3$ depends on $Tx4$. $Tx1$ and $Tx4$ are included in block A, and $Tx2$, $Tx3$ are included in block B. Without loss of generality, we assume that A is proposed first, $Tx3$ has not been on the transaction chain at this time. Honest validators will consider the block A to be invalid because the input of $Tx4$ is invalid.

Eventually-Committed rule. We define that blocks that meet Eventually-Committed rule can be included in LOG: Consider an honest node and a blockDAG $\mathcal{G}$ consisting of k transaction chains at time t, we define B is a set of b_{commit} of k transaction chains. Let $Height_{confirm} = min(\{b \in$

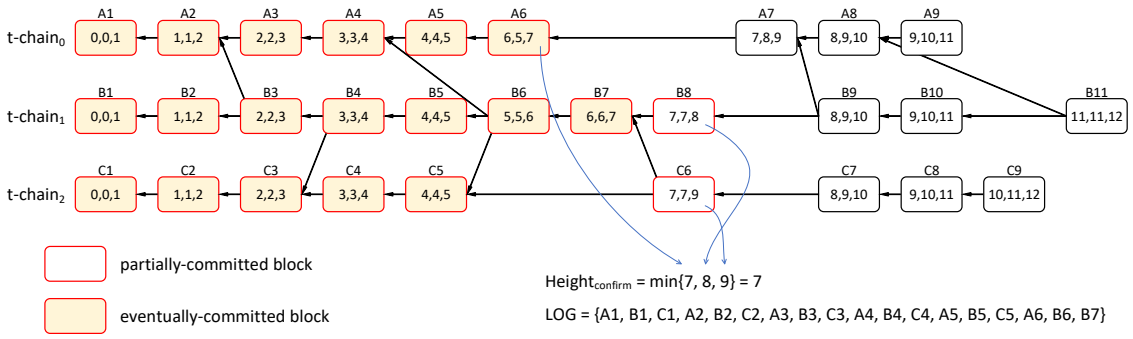


Fig. 4: Structure of transaction chains. Each block has a tuple $< round, height, nextHeight >$.There are three transaction chains and the lastest *partially-committed* blocks on each chain are A6, B8 and C6 according to the partially-committed rule.. Then, $Height_{confirm} = 7$.

$B|b.nextHeight\}$), then blocks in the set $\{b \in \mathcal{G}|b.height < Height_{comfirm}\}$ are *eventually-committed*. *Eventually-committed* blocks are ordered by the *height*, with tie-breaking favoring smaller transaction chain ids.

The function $Order()$ in *Algorithm* 1 gives the pseudo-code to generate a LOG. For example in Figure 4, each block has a tuple$< round, height, nextHeight >$. We assume that there are three transaction chains and the lastest *partially-committed* blocks of each chain are A6, B8 and C6. The *eventually-committed* blocks are the blocks whose *height* are smaller than $Height_{confirm}$. $LOG = \{A_1,B_1,C_1,A_2,B_2,C_2,A_3,B_3,C_3,A_4,B_4,C_4,A_5,B_5,C_5,A_6,B_6,B_7\}$. If there are two conflicting transactions in A_1 and B_2, then the transaction in B_2 is invalid.

Algorithm 1 Total ordering protocol

Require: B_{commit}; $LOG = \emptyset$; $Height_{comfirm} = 0$; $Index = \{i \in [0, k)|index_i\}$;
1: **function** ORDER()
2: $h = Height_{comfirm}$;
3: $Height_{comfirm} =$
4: $min(b \in B_{commit}|b.nextHeight)$;
5: **while** $h < Height_{comfirm}$ **do**
6: **for** $i = 0 \rightarrow k - 1$ **do**
7: **if** $t\text{-}chain_i[index_i].height = h$ **then**
8: $LOG = t\text{-}chain_i[index_i] \cup LOG$;
9: $index_i = index_i + 1$;
10: **end if**
11: **end for**
12: $h = h + 1$;
13: **end while**
14: return LOG;
15: **end function**

4.4 Incentive Mechanism

The final challenge is to design a fair rewards allocation mechanism to enhance the incentive property of HotDAG. The key to our design is to use the *epoch reputation* to represent the performance of validators in committees, including the resources and behaviors. Our incentive mechanism includes three parts: malicious behaviors, epoch reputation and rewards allocation.

Malicious behaviors. HotDAG can detect three malicious behaviors: malicious votes, conflicting blocks and double-spending transactions. We can judge whether a validator is malicious through the first two behaviors. If a leader receives a vote that violates the voting rules, it can conclude that the validator who sends the vote is malicious. If a node receives two different blocks with the same round that are proposed by the same validator, it can conclude that this validator proposes conflicting blocks. The leader can propose a block with the evidence of malicious behaviors. Finally, double-spending transactions can be easily detected in *LOG*: transactions in *LOG* that cost the same input. For nodes who have sent double-spending transactions, we can increase the cost of malicious behaviors by increasing its transaction fee.

Epoch Reputation. Inspired by Accountable-Subgroup Multisignatures Scheme [28], we assume that a quorum certificate(QC) contains the identity information of $2f + 1$ voters. [28] guarantees that leaders cannot forge the identity information. There is an identity array id in QC that is used to record whether the validator in the committee has voted. When the leader receives a valid vote from the validator i, it sets $id[i]$ to 1. The leader generates an aggregate signature containing the identity array id when it receives enough votes. Everyone can verify it by aggregating all public keys pk_i that are 1 in the array id. Accountable-Subgroup Multisignatures Scheme guarantees that leaders cannot forge the identity information of voters.

After the epoch ends for a period of time, any node can calculate the epoch reputation because the proposer and voters of a block are recorded on the transaction chains. The epoch reputation of the validator i is calculated according to *Algorithm* 2 and notations are defined in Table 2 . It is worth noting that $Y \in \{0, 1\}$ represents the honesty of the validator. It is $'1'$ for each new validator and it is set to $'0'$ if the validator has misbehaved[1]. For the validator i, x_i is the standard score (z-score) and r_i is the epoch reputation. It should be noted that the epoch reputation will not be accumulated, that is, the epoch reputation previously obtained by the validator will not have any impact on rewards allocation of the current epoch (unless malicious behaviors of the node has been detected).

Block rewards and Transaction Fees. Similar to Bitcoin, there are two kinds of rewards in HotDAG: block rewards and transaction fees. We assume that a committee confirmed L blocks in epoch e, and *LOG* grew S blocks during this period. The basic block rewards is R_b and the total transaction fees in epoch e are F. We assume that A ($A > 1$) is the rewards parameter. The rewards for the validator i is:

$$R_i = F * \frac{r_i}{\Sigma_{i=1}^{c} r_i} + (1 + \frac{L}{S} * A) * R_b * l_i \tag{1}$$

[1] It includes sending malicious votes and proposing conflicting blocks.

Table 2: Notations of incentive mechanism

Natation	Explanation
γ	epoch reputation parameter
c	the size of committee
L	the length of the *t-chain* generated in the current epoch
v_i	the number of the i-th validator's votes[1]
l_i	the number of blocks that are proposed by the i-th validator
Y	whether the validator is honest("1") or not ("0")

[1] The vote is not included when it is the leader.

Algorithm 2 Epoch reputation

Require: $c, L, \{v_i\}_{i=1}^{c}, \{l_i\}_{i=1}^{c}, \{Y_i\}_{i=1}^{c}, \gamma$.

1: $mean_v = \frac{\Sigma_{i=1}^{c} v_i}{c}$
2: $mean_l = \frac{\Sigma_{i=1}^{c} l_i}{c}$
3: $s_v = \sqrt{\frac{\Sigma_{i=1}^{c}(v_i - mean_v)^2}{c}}$
4: $s_l = \sqrt{\frac{\Sigma_{i=1}^{c}(l_i - mean_l)^2}{c}}$
5: $mean_v = mean_v - \gamma * L$
6: $x_i = \frac{l_i - mean_l}{s_l} + \frac{v_i - mean_v}{s_v}$
7: $r_i = Y_i * \frac{1}{1+e^{-x_i}}$
8: return r_i;

5 SECURITY ANALYSIS

In this section, we provide security analysis for HotDAG, showing that HotDAG is highly secure with overwhelming probability. We define that some probability p is strongly negligible if there exist some constants c_0>0, c_1, such that for a security parameter λ, $p \leq exp(-(c_0\lambda + c_1))$. If some event happens with high probability, it happens with probability of at least $1 - exp(-(c_0\lambda + c_1))$.

5.1 Security on Committee Configuration

We define a security parameter λ and a parameter T. Define a *slot* to be the total time needed to do a single query to the random oracle by an honest node. We allow $\mathcal{A}$ to potentially make αn queries in a *slot*. Consider any $\varepsilon > 0$, $\alpha < \frac{1}{4} - \varepsilon$, there exits $p_0 < \Theta(\frac{1}{\Delta n})$ for every n. For all $p < p_0$ and any constant $\epsilon > 0$, each identity chain satisfies the following properties with the probability at least 1-$k * exp(-\Omega(\lambda))$-$k * exp(-\Omega(T))$:

Theorem 1. (The *consistency* property in *Theorem* 1 of [6]): Suppose that the identity chain of an honest node i at time t is $Chain$, and the identity chain of an honest node j at time t' is $Chain'$ (i may equal to j and t may equal t'). It holds that $Chain[:T] \prec Chain'$ if $t \leq t'$.

Theorem 2. (The *quality* property in *Theorem* 1 of [6] and *Corollary* 1 in [13]): There are at least $\frac{2T}{3}$ honest blocks in any consecutive T blocks on the identity chain (The honest blocks are

contributed by honest nodes). At any point, the identity chain grows T blocks within $\frac{T}{s}$ slots, where $g(n,p) \leq s \leq g'(\epsilon,n,p)$, $g(n,p) = \frac{3}{4}np$, $g'(\epsilon,n,p) = (1+\epsilon)np$..

5.2 Security on Intra-committee Consensus

Notation 1. For a block b, we define that $b \leftarrow b'$ denotes the parent of b' is b. $b \leftarrow\!\!- b'$ denotes b is in the branch led by b', that is, $b \leftarrow b_0 \leftarrow ... \leftarrow b'$. $b \Leftarrow b'$denotes $b \leftarrow b'$ and $b'.round = b.round + 1$.
Notation 2. We define the *partially − committed rule* in our protocol as $Commit(b_0, b) : b_0 \Leftarrow b_1 \Leftarrow b_2 \leftarrow b$.

Lemma 1. In a fixed set of $n = 3f + 1$ nodes where f is the number of byzantine nodes and n is the total number of all nodes, we define that a quorum consists of Q nodes ($Q = 2f + 1$). The intersection of two quorums must exist an honest node.
Proof. Let Q_1 and Q_2 be two quorums. There are at least $f + 1$ common nodes in two quorums : $M = Q_1 \cap Q_2$, $|M| = 2Q - n = f + 1$. There must be at least one honest node in M because of at most f malicious nodes.

Lemma 2. Consider two blocks b and b' such that $b \neq b'$ and $b.round = b'.round$, it is impossible to generate two valid QCs for b and b', respectively.
Proof. To generate a QC, the block must receive $2f+1$ votes from different nodes. From *Lemma* 1, there must be an honest validator p who votes for both blocks. We assume that the validator p votes for block b first, then the round of the last voted block is $b.round$. When p receives the block b', it cannot vote for b' according to the voting rule 3. Therefore, b and b' cannot both have valid QCs.

Corollary 3. Consider two blocks b and b' such that $b \neq b'$ and $b.height = b'.height$, it is impossible to generate two valid QCs for b and b', respectively.

Lemma 4. We assume four blocks such that $b \Leftarrow b' \Leftarrow b'' \leftarrow b'''$. For a block b^* such that $b^*.round > b''.round$ and there is a QC^* for b^*, we have $b^*.parent.round > b.round$.
Proof. From *Lemma* 1, there must be an honest node who votes for b' and b^*. The node updates $b_{lock} = b$ when it receives b''. Upon receiving b^*, the honest node votes for it if and only if b^* meets voting rules. From the voting rule 4, we have $b^*.parent.round > b_{lock}.round$. Therefore $b^*.parent.round \geqslant b.round$.

Lemma 5. We assume four blocks such that $b \Leftarrow b' \Leftarrow b'' \leftarrow b'''$. For a block b^* that $b^*.round > b.round$ and there is a QC for it, We have $b \leftarrow\!\!- b^*$.
Proof. If $b^*.round \leq b''.round$, we can infer that b^* is b' or b'' from *Lemma* 2. Therefore we have $b \leftarrow\!\!- b^*$. If $b^*.round > b''.round$, we assume b^* is the block with smallest round larger than $b''.round$. Let b_{parent} be the parent of b^*, we have $b_{parent}.round \geqslant b.round$ from *Lemma* 4 and $b_{parent}.round < b^*.round$. By minimality of b^* and *Lemma* 2, $b \leftarrow\!\!- b_{parent}$ and $b_{parent} \leftarrow\!\!- b^*$. Therefore, we have $b \leftarrow\!\!- b^*$.

Notation 3. We define the branch led by the block b as $\Phi(b)$.

Theorem 3(Consistency for each transaction chain) . For a transaction chain, consider

$b_{commit} = b_0$ at time t, and $b_{commit} = b'_0$ at time t'. We have either $\Phi(b_0) \prec \Phi(b'_0)$ or $\Phi(b'_0) \prec \Phi(b_0)$.
Proof. We assume that b_0 is committed by $Commit(b_0, b) : b_0 \Leftarrow b_1 \Leftarrow b_2 \leftarrow b$, and b'_0 is committed by $Commit(b'_0, b') : b'_0 \Leftarrow b'_1 \Leftarrow b'_2 \leftarrow b'$. From *Lemma* 2, $b_0.round$ is different from $b'_0.round$. Without loss of generality, we assume $b_0.round < b'_0.round$. Then according to *Lemma* 5, $b_0 \dashleftarrow b'_0$. Therefore $\Phi(b_0) \prec \Phi(b'_0)$.

5.3 Security on Inter-committee Consensus

Lemma 6. Consider a *LOG* of an honest node, two blocks b_0, $b_1 \in LOG$ at time t such that b_0 is before b_1, then at any time $t' > t$, b_0 is before b_1.
Proof. If b_0, $b_1 \in t\text{-}chain_i$, we have at any time $t' > t$, b_0 is before b_1 from *Theorem* 3. If b_0 and b_1 are on different transaction chains and there is no happen-before relationship between them. They are ordered by *height*, and be broken by t-chain id when $b_0.height = b_1.height$. Then b_0 is before b_1. If there exits happen-before relation between b_0 and b_1, we assume that $b_0 \in b_1.references$, w.l.o.g. Let h_0 and n_0 be *height* and *nextHeight* of b_0, h_1 and n_1 be *height* and *nextHeight* of b_1, $Height_{comfirm} = H$ at time t. We have $h_1 = max\{b_1.parent.nextHeight, max_{r \in b.references}(r.height + 1)\}$, that is, $h_1 > h_0$. If $b_1 \in LOG$ at time t, $h_1 < H$. Then $h_0 < h_1 < H$, therefore $b_0 \in LOG$. All blocks whose height is smaller than H are ordered by *height* according to *algorithm* 3, therefore b_0 is before b_1 at any time $t' > t$.

Theorem 4 (Consistency for *LOG*). Suppose that an honest node outputs *LOG* at time t and an honest node outputs LOG' at time t'. Then either $LOG \prec LOG'$ or $LOG' \prec LOG$.
Proof. We assume that $Height_{comfirm} = h$ at time t and $Height_{comfirm} = h'$ at time t'. Let $\Phi_i(b_{commit,i})$ and $\Phi'_i(b'_{commit,i})$ be *partially-committed* blocks on $t\text{-}chain_i$ at time t and t' (abbreviated as Φ_i and Φ'_i). Let $\mathbb{E}_i$ be a set of blocks on Φ_i whose *height* is smaller than h. And $\mathbb{E}'_i$ is a set of blocks on Φ'_i whose *height* is smaller than h'. We have $\mathbb{E}_i \prec \Phi_i$ and $\mathbb{E}'_i \prec \Phi'_i$. From *Theorem* 4, either $\Phi_i \prec \Phi'_i$ or $\Phi'_i \prec \Phi_i$. We assume $\Phi_i \prec \Phi'_i$, w.l.o.g. It is clear that $\mathbb{E}_i \prec \mathbb{E}'_i$. We have $LOG = \{i \in [0, k) | \mathbb{E}_i\}$ and $LOG' = \{i \in [0, k) | \mathbb{E}'_i\}$. Then, $LOG \subseteq LOG'$. For the block $b \in \{\mathbb{E}'_i - \mathbb{E}_i\}$, $b.height \geq h$. Therefore all blocks in the set $\{\mathbb{E}'_i - \mathbb{E}_i\}$ are ordered after $\mathbb{E}_i$. From *Lemma* 6, the sequence of blocks in *LOG* and LOG' are same, therefore we have $LOG \prec LOG'$.

6 PERFORMANCE ANALYSIS AND INCENTIVE ANALYSIS

In this section, we will analyze the performance and incentive of HotDAG and compare it with previous work. We use n to denote the total number of nodes, c to denote the committee size and b to denote the block size.

6.1 Identity Establishment

In this phase, all nodes of HotDAG need to broadcast a valid PoW solution which imposes a communication complexity of $\mathcal{O}(n^2)$. OmniLedger is the same as HotDAG. In RapidChain and CycLedger, nodes send the PoW solution to the reference (or referee) committee, and the communication complexity is $\mathcal{O}(nc)$. RapidChain and CycLedger achieve the lower communication complexity at the cost of weakening decentralization.

6.2 Committee Configuration

HotDAG implements the committee configuration at the same time during the identity establishment, so there is no additional communication overhead in this phase. OmniLedger uses RandHound with a VRF-based leader election algorithm to assign committees which causes a communication complexity of $\mathcal{O}(n^2 + nc^2)$. In RapidChain and CycLedger, the reference committee distributedly generates the next round's randomness and agrees on the assigned committees, then it broadcasts to the whole network. The communication complexity is $\mathcal{O}(nc)$.

6.3 Intra-committee Consensus

Without loss of generality, we analyze the communication complexity of consensus per transaction. We consider a transaction that involves all committees. In HotDAG, each validator in the committee sends a $\mathcal{O}(1)$-size vote to the leader. The leader aggregates the votes into a $\mathcal{O}(1)$-size threshold signature and sends it to all validators together with a block. The communication complexity is $\mathcal{O}(c/b)$ and stays the same when rotating leaders. OmniLedger, RapidChain and CycLedger have the communication complexity of $\mathcal{O}(nc/b)$.[2]

6.4 Inter-committee Consensus

In HotDAG, the leader is responsible for broadcasting the block with a QC to the entire network which causes a communication complexity of $\mathcal{O}(n/b)$. In OmniLedger and RapidChain, the client or the output committee is responsible for broadcasting the transaction to the involved committee, the communication complexity are $\mathcal{O}(n)$ and $\mathcal{O}(c \log n)$, respectively. In CycLedger, the communication complexity is $\mathcal{O}(nc/b)$.

We summarize a comparison of theoretical performance analysis with previous sharding blockchain protocols in Table 3.

Table 3: A COMPARISON OF THEORETICAL PERFORMANCE ANALYSIS

Communication Complexity	OmniLedger	RapidChain	CycLedger	HotDAG
Identity Establishment[1]	$\mathcal{O}(n^2)$	$\mathcal{O}(nc)$	$\mathcal{O}(nc)$	$\mathcal{O}(n^2)$
Committee Configuration	$\mathcal{O}(n^2 + nc^2)$	$\mathcal{O}(nc)$	$\mathcal{O}(nc)$	–[2]
Intra-committee Consensus[3]	$\mathcal{O}(nc/b)$	$\mathcal{O}(nc/b)$	$\mathcal{O}(nc/b)$	$\mathcal{O}(c/b)$
Inter-committee Consensus	$\mathcal{O}(n)$	$\mathcal{O}(c \log n)$	$\mathcal{O}(nc/b)$	$\mathcal{O}(n/b)$

[1] RapidChain and CycLedger achieve the lower communication complexity at the cost of weakening decentralization.
[2] There is no communication overhead.
[3] We assume that a cross-shard transaction is relative with all committees.

[2] There are only 3 shards in the experiment of OmniLedger, and the probability of a transaction involving more than two shards is as high as 96.3%.

6.5 Incentive analysis

To avoid the serious monopoly problem, HotDAG decouples the reputation from consensus and focuses on designing a fair incentive mechanism for the protocol that uses a threshold signature scheme. The goal of HotDAG's incentive mechanism is that the epoch reputation can correctly reflect the resources (including computing resources and network resources) of honest validators, which means that the proportion of rewards obtained by honest validators is close to the proportion of their resources. We analyze the following two attack strategies:

Free-riding Attack. Free-riding is to establish a public goods game [29]. In ByzCoin [11], everyone in committee is rewarded regardless of participation. In this case, keeping silent is the best strategy for the attacker because honest validators will ensure transactions to be confirmed.

Aggregation Attack. A strategy to prevent free-riding attacks is to evenly distribute rewards to participants. The attacker can actively participate in the consensus and collude with other attackers. When a attacker becomes the leader, it prefers the vote from its peers to get more rewards.

In HotDAG, basic tasks of validators are to propose blocks and to give opinions on the block. Therefore, we propose to use the times that the validator successfully proposes a block and the times that the validator's vote is included in the QC to reflect its behaviors. Specifically, a validator with more resources can make a greater contribution to the consensus process, thereby gaining a higher epoch reputation. The proportion of a validator's reputation in the committee determines the transaction fees it received. Therefore, lazy behaviors (keeping silent) will lead to few rewards while malicious behaviors will make validators almost impossible to obtain any revenue. On the other hand, the aggregation attack will increase $mean_v$. HotDAG weakens the advantage of attackers by lowering $mean_v$ to calculate the standard score and using a sigmoid function to map epoch reputation. Finally, we introduce the dynamic block rewards in the Bitcoin's reward model and the dynamic block rewards can incent the validators to behave honestly and confirm more blocks.

7 Conclusion

We present HotDAG, a hybrid consensus protocol in the permissionless model. HotDAG achieves security and scalability through a novel method consisting of four parts. First, we present a decentralized and bias-resistant committee formation protocol for sharding securely. Second, each transaction requires only one committee to run the BFT protocol and the inter-committee consensus has linear communication complexity. Third, HotDAG is strongly censorship-resistant due to the unpredictable leader rotation. Finally, we design an incentive mechanism that is compatible with the scalable BFT protocol and all rational nodes are motivated to honestly follow the protocol. In the future, we will implement and evaluate HotDAG in the real network.

References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2009), http://www.bitcoin.org
2. Yonatan, S., Aviv, Z.: Secure high-rate transaction processing in bitcoin. In: Financial Cryptography and Data Security - 19th International Conference (2015). pp. 507–527

3. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: 13th USENIX Symposium on Operating Systems Design and Implementation. pp. 45–59 (2016)
4. Sompolinsky, Y., Zohar, A.: Phantom: A scalable blockdag protocol. Cryptology ePrint Archive 2018, 104 (2018)
5. Li, C., Li, P., Xu, W., Long, F., Yao, A.C.C.: Scaling nakamoto consensus to thousands of transactions per second. CoRR,abs/1805.03870 (2018)
6. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE:blockchain scaling made simple. In: Proceedings of the 41st IEEE Symposium on Security and Privacy. pp. 112–127
7. Douceur, J.R.: The sybil attack. In: Proceedings of the 1st International Workshop on Peer to Peer Systems(2001). pp. 251–260
8. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Proceedings of the 12th Annual International Cryptology Conference
9. Yu D, Zou Y, Yu J, et al. Implementing Abstract MAC Layer in Dynamic Networks. IEEE Transactions on Mobile Computing, DOI: 10.1109/TMC.2020.2971599, 2020.
10. Hua Q.S, Shi Y, Yu D, et al. Faster Parallel Core Maintenance Algorithms in Dynamic Graphs. IEEE Transactions on Parallel and Distributed Systems. 31(6): 1287-1300(2020).
11. Kokoris, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium. pp. 279–296
12. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Spiegelman, A.: Solida: A blockchain protocol based on reconfigurable byzantine consensus. In: 21st International Conference on Principles of Distributed Systems. pp. 25:1–25:19
13. Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017). pp. 39:1–39:16. Leibniz International Proceedings in Informatics (LIPIcs)
14. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 51–68
15. Zhu S , Cai Z , Hu H , et al. zkCrowd: A Hybrid Blockchain-based Crowdsourcing Platform. IEEE Transactions on Industrial Informatics (TII). 16(6): 4196-4205 (2020).
16. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 17–30
17. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. Cryptology ePrint Archive, Report 2017/406 (2017)
18. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: Scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 931–948
19. Mengqian Zhang, Jichen Li, Z.C.H.C., Deng, X.: Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding. In: Proceedings of the 34th IEEE International Parallel and Distributed Processing Symposium(2020)
20. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis, The University of Guelph, Guelph, Ontario, Canada (June 2016)
21. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing.
22. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) Proceedings of the 9th Annual International Cryptology Conference
23. Shoup, V.: Practical threshold signatures. In: EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques
24. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security
25. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation(1999). pp. 173-186

26. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039) (1999)
27. Zhou, C.X., Hua, Q.S., Jin, H.: HotDAG: Hybrid consensus via sharding in the permissionless model, https://qiangshenghua.github.io/papers/hotdag.pdf
28. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 435–464 (2018)
29. Archetti, M., Scheuring, I.: Review: Game theory of public goods in one-shot social dilemmas without assortment. Journal of Theoretical Biology 299, 9–20 (2012)