SoK: Sharding on Blockchain

Gang Wang, Zhijie Jerry Shi
University of Connecticut
{gang.wang,zshi}@uconn.edu

Mark Nixon
Emerson Automation Solutions
mark.nixon@emerson.com

Song Han
University of Connecticut
song.han@uconn.edu

ABSTRACT

Blockchain is a distributed and decentralized ledger for recording transactions. It is maintained and shared among the participating nodes by utilizing cryptographic primitives. A consensus protocol ensures that all nodes agree on a unique order in which records are appended. However, current blockchain solutions are facing scalability issues. Many methods, such as Off-chain and Directed Acyclic Graph (DAG) solutions, have been proposed to address the issue. However, they have inherent drawbacks, e.g., forming parasite chains. Performance, such as throughput and latency, is also important to a blockchain system. Sharding has emerged as a good candidate that can overcome both the scalability and performance problems in blockchain. To date, there is no systematic work that analyzes the sharding protocols. To bridge this gap, this paper provides a systematic and comprehensive review on blockchain sharding techniques. We first present a general design flow of sharding protocols and then discuss key design challenges. For each challenge, we analyze and compare the techniques in state-of-the-art solutions. Finally, we discuss several potential research directions in blockchain sharding.

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Distributed systems security**.

KEYWORDS

SoK, Blockchain, Sharding, Consensus Protocol

ACM Reference Format:
Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. 2019. SoK: Sharding on Blockchain. In *1st ACM Conference on Advances in Financial Technologies (AFT '19), October 21–23, 2019, Zurich, Switzerland*. ACM, New York, NY, USA, Article 4, 21 pages. https://doi.org/10.1145/3318041.3355457

1 INTRODUCTION

The blockchain has become a key technology for implementing distributed ledgers. It allows a group of participating nodes (or parties) that do not trust each other to provide trustworthy and immutable services. Distributed ledgers were initially used as tamper-evident logs to record data. They are typically maintained by independent parties without a central authority, for example, in systems like SUNDR [95], SPORC [66], and Tamper-Evident Logging [53]. The blockchain became popular because of its success in crypto-currencies, e.g., Bitcoin [107]. Blockchain stands in the tradition of distributed protocols for both secure multiparty computation and replicated services for tolerating Byzantine faults [101]. With blockchain, a group of parties can act as a dependable and trusted third party for maintaining a shared state, mediating exchanges, and providing a secure computing engine [34].

Consensus is one of the most important problems in blockchain, as in any distributed systems where many nodes must reach an agreement, even in the presence of faults. Current consensus algorithms are only applicable to small-scale systems because of complexity, e.g., the Practical Byzantine Fault Tolerance protocol (PBFT) [37] with less than 20 participating nodes. Scalability is an issue that has to be addressed before adopting blockchain in large-scale applications. Recently, many solutions have been proposed to achieve the scale-out throughput by allowing participating nodes only to acquire a fraction of the entire transaction set, for example, an Off-chain solution [114], Directed Acyclic Graph (DAG) [115] and blockchain sharding [100]. However, the off-chain solution is more subject to forks and the transactions in the DAG layout are not organized in a chain structure. Among all proposed methods, sharding schemes seem to be the most effective candidate as it can overcome both performance and scalability problems. A sharding scheme splits the processing of transactions among smaller groups of nodes, called *shards*. As a result, shards can work in parallel to maximize the performance and improve the throughput while requiring significantly less communication, computation, and storage overhead, allowing the scheme to work in large systems [141].

Particularly, sharding technology utilizes the concept of committees. The term committee is also used to refer to a subset of participating nodes that collaborate to finish a specific function. The notion of committees in the context of consensus protocols was first introduced by Bracha [25] to reduce the round complexity of Byzantine agreement. Using committees to reduce the communication and computation overhead of Byzantine agreement dates back to the work

of King *et al.* [88, 89]. However, they provided only theoretical results and the techniques cannot be directly used in a blockchain setting. Sharding-based blockchain protocols can increase the transaction throughput when more participants join the network because more committees can be formed to process transactions in parallel. The total number of transactions processed in each consensus round by the entire network is multiplied by the number of committees. For security reasons, a sharding scheme needs to fairly and randomly divide the network into small shards with the vanishing probability of any shard having an overwhelming number of adversaries.

Although sharding is promising, it still faces many specific design challenges. We need to identify key components in blockchain sharding, understand the challenges in each component, and systematically study potential solutions to each challenge. To date, there has been no systematic and comprehensive study or review on blockchain sharding. To fill the gap, this paper presents a comprehensive and systematic study of sharding techniques in blockchain. We identify the key components in sharding schemes and the major challenges in each component. As a systematization of knowledge on blockchain sharding, we also analyze and compare the state-of-the-art solutions.

The rest of the paper is organized as follows. Section 2 introduces various models and taxonomies of blockchain systems. Section 3 gives an overview of sharding. Section 4 discusses consensus protocols. Section 5 presents the approaches to generating epoch randomness. Section 6 discusses how to deal with cross-sharding transactions. Section 7 discusses the reconfiguration of epochs. Section 8 compares the state-of-the-art sharding protocols. Section 9 concludes this paper.

2 PRELIMINARIES

This section introduces various models and taxonomies for blockchain protocols, followed by discussion on typical blockchain settings and scalability issues. In this paper, we consider the terms *node*, *replica*, *party*, *entity*, and *participant* having the same meaning as *participating node*.

2.1 Models in Blockchain

2.1.1 Communication Models. A consensus protocol for distributed systems is greatly dependent on the underlying communication network. Typically, we can categorize communication networks into three types [5]: strongly synchronous, partially synchronous, and asynchronous. A network is said to be *strongly synchronous* if there exists a known fixed bound, δ, such that every message takes at most δ time units to travel from one node to another in the network. A network is said to be *partially synchronous* if there exists a *fixed* bound, δ, on the network delay and one of the following conditions holds: 1) δ always holds, but is unknown; 2) δ is known, but only starts at some unknown time. A network is said to be *asynchronous* if there is no upper bound on the network delay. It is worth mentioning that the communication network models also vary by the network adversarial models, e.g., adversarial network scheduling models and oblivious adversarial models [6].

A consensus protocol must meet three requirements [103]: (a) *Non-triviality.* If a correct entity outputs a value v, then some entity proposed v; (b) *Safety.* If a correct entity outputs a value v, then all correct entities output the same value v; (c) *Liveness.* If all correct entities initiated the protocol, then, eventually, all correct entities output some value. Note that Fisher, Lynch and Paterson (FLP) [68] proved that a deterministic agreement protocol in an asynchronous network cannot guarantee liveness if one entity may crash, even when links are assumed to be reliable. In an asynchronous system, one cannot distinguish between a crashed node and a correct one. Theoretically, deciding the full network's state and deducing from it an agreed-upon output is impossible. However, there exist some extensions to circumvent the FLP result to achieve an asynchronous consensus, e.g., randomization, timing assumptions, failure detectors, and strong primitives [6].

2.1.2 Fault Models. We distinguish two types of fault consensus: crash fault-tolerant consensus (CFT) and non-crash (Byzantine) fault-tolerant consensus (BFT) [98]. Different failure models have been considered in the literature, and they have distinct behaviors. In general, a crash fault is where a machine simply stops all computation and communication, and a non-crash fault is where it acts arbitrarily, but cannot break the cryptographic primitives, e.g., cryptographic hashes, MACs, message digests, and digital signatures. For instance, in a crash fault model, nodes may fail at any time. When a node fails, it stops processing, sending, or receiving messages. Typically, failed nodes remain silent forever although some distributed protocols have considered node recovery. Tolerating the crash faults (e.g., corrupted participating nodes) as well as network faults (e.g., network partitions or asynchrony) reflects the inability of otherwise correct machines to communicate among each other in a timely manner. This reflects how a typical CFT fault affects the system functionalities. At the heart of these systems typically lies a CFT-based state-machine replication (SMR) primitive [39]. However, these systems cannot deal with non-crash faults, which is also called Byzantine failure. In Byzantine failure models, failed nodes may take arbitrary actions, including sending and receiving messages that are specially crafted to break the consensus process.

Classic CFT and BFT explicitly model machine faults only. These are then combined with an orthogonal network fault model, for either synchronous or asynchronous networks. Thus, the related work can be classified into four categories: synchronous CFT [52], asynchronous CFT [109], synchronous BFT [59], and asynchronous BFT [76] [36]. The Byzantine setting is of relevance to security-critical settings and traditional consensus protocols that tolerate crash failures only.

2.2 BFT Consensus Scalability

Sharding a blockchain largely relies on BFT consensus protocols to reach consensus. However, most BFT protocols are limited in their scalability, either in terms of network size (e.g., number of nodes) or the overall throughput. The design space for improving them is vast. We will use Practical BFT (PBFT) [37] as an example to explain BFT scalability. The original PBFT protocol requires $n = 3f + 1$ nodes to tolerate up to f Byzantine faults. It has been shown not to scale beyond a dozen nodes due to its quadratic communication complexity [58]. Typically, scaling protocols for BFT focuses on either reducing the number of nodes required to tolerate f Byzantine faults [15, 44], or reducing the protocol's communication complexity to allow larger network sizes [90].

Reducing the number of nodes. To tolerate f Byzantine nodes that can *equivocate* in a *quorum* system like PBFT, quorums must be intersected by at least $f + 1$ nodes [102]. Consequently, if a BFT protocol requires $n = 3f + 1$, its quorum size is at least $2f + 1$. The smaller n means the lower communication cost incurred in tolerating the same number of faults; it also means that for the same number of nodes n, the network can tolerate more faulty nodes. One way to reduce the number of nodes is to randomly select a small set of consensus nodes, as a committee, to run a consensus process. A smaller consensus committee can lead to better throughput, as a smaller committee attains higher throughput due to lower communication overhead. Sharding technology reduces the consensus process within one shard. However, in this scenario, the security of each shard, e.g., the ratio of the number of faulty nodes to the size of a shard, will be the top concern. It can be mitigated by utilizing some mechanisms, e.g., the epoch randomness, to guarantee the "good majority" for each shard with a high probability [100].

Another way to reduce the number of nodes is to utilize techniques to get down the n from $3f + 1$ to $2f + 1$. Those techniques are mainly based on leveraging external components (e.g., the trusted hardware) or lessening the system models. For example, *BFT-TO* [48], a hardware-assisted BFT, with less replicas, shows that it is possible to implement a Byzantine SMR algorithm with only $2f + 1$ replicas by expending the system with a simple trusted distributed component. Similarly, there exist a few other algorithms to achieve the consensus with less replicas, such as A2M-BFT-EA [44], MinBFT [133], MinZyzzyva [133], EBAWA [132], CheapBFT [82], and FastBFT [97]. Besides, there also exist some other work to achieve the same purpose by lessening the system models. For example, the work in [1] improves the BFT threshold to $2f + 1$ by utilizing a relaxed synchrony assumption.

Reducing communication complexity. PBFT protocol has been perceived to be a communication-heavy protocol. There is a long-standing myth that BFT is not scalable to the number of participants n, since most existing solutions incur the message transmission of $O(n^2)$, even under favorable network conditions. As a result, existing BFT chains involve very few nodes (e.g., 21 in [75]). Even with a reduced network size, PBFT still has a communication complexity of $O(n^2)$. Byzcoin [90] proposed an optimization wherein the leader uses a collective signing protocol (CoSi) [128] to aggregate other node's messages into a single authenticated message. By doing so, each node only needs to forward its messages to the leader and verify the aggregate message from the latter. In this way, by avoiding broadcasting, the communication complexity is reduced to $O(n)$. Besides, there is some work [56] on utilizing trusted execution environments (TEEs) (e.g., Intel SGX [50]) to scale distributed consensus. TEEs provide a protected memory and isolated execution so that the regular operating systems or applications can neither control nor observe the data being stored or processed inside them [64]. Generally, a trusted hardware can only crash but not be Byzantine. However, introducing trusted hardware into consensus nodes is expensive, and specific knowledge is needed to implement the protocol. Similarly, the security in this category can be mitigated by using cryptograhic primitives, such as threshold signatures [23] [125].

By splitting a network into multiple committees, sharding technology reduces the number of consensus nodes within committees and further reduces the communication complexity.

2.3 Scalability in Sharding Blockchain

The blockchain scalability can be evaluated by two metrics: transaction throughput (e.g., the maximum rate at which the blockchain can process transactions) and latency (e.g., the time to confirm that a transaction has been included in the blockchain). Blockchain with message communication complexity $O(n)$ per node, where n is the number of participating nodes, is typically referred to as a "scalable" blockchain since its throughput will not decrease with the number of participating nodes and the communication capacities in the

network. Sharding is one such solution that fairly and randomly divides the network into small shards with vanishing probability of any shard having an overwhelming number of adversaries.

In general, when considering scalability in sharding, it is restricted to approaches targeting the blockchain's core design, e.g., on-chain solutions, rather than techniques that delegate to parallel off-path blockchain instances such as sidechains (one of the off-chain solutions) [12]. Sharding based blockchain systems typically operate in *epochs* (e.g., one epoch specifies the maximum time to form one block): the assignment of nodes to committees is valid only for the duration of that epoch. The number of committees scales linearly to the amount of computational power available in the system, and the number of nodes within a committee can be flexible. Thus, as more nodes join the network, the transaction throughput increases without adding to the latency, since messages needed for consensus are decoupled from computation and broadcast of the final block to be added to the blockchain. However, sharding a blockchain is difficult because it must ensure some properties, e.g., a transaction (i.e., spending some cryptocurrencies) is only executed once on the entire network. If a transaction that should happen only once executes more than once, it goes into a situation of *double spending* [116]. Thus, we need to understand the essential components on sharding-based blockchain system.

3 SHARDING OVERVIEW

Originally, sharding is a type of database partitioning technique that separates a very large database into much smaller, faster, more easily managed parts called data shards [99]. The term *shard* represents a small part of the whole set. Technically, sharding is a synonym for horizontal partitioning, which makes a large database more manageable. The key idea of sharding in blockchain is to partition the network into smaller committees, each of which processes a disjoint set of transactions (or a "shard"). Specifically, the number of committees grows linearly in the total computational power of the network. And each committee has a reasonably small number of members so they can run a classic Byzantine consensus protocol to decide their agreed set of transactions in parallel.

3.1 Problem Definition

Assume that there exist n participating nodes having the same computational power, a fraction f of which is controlled by a Byzantine adversary. The network accepts transactions per block, e.g., a transaction i in block j is represented by an integer $x_i^j \in Z_N$, where Z_N [38] is the ring of integers modulo N. All nodes have access to an externally-specified constraint function $C : Z_N \rightarrow \{0, 1\}$ to determine the validity of each transaction. The sharding protocol is to seek a protocol Π running between nodes which outputs a set X which contains k separate "shards" or subsets $X_i = \{x_i^j\}(1 \leq j \leq |X_i|)$ such that the following conditions hold:

- *Agreement.* Honest nodes agree on X with a probability of at least $1 - 2^{-\lambda}$, for a given security parameter λ.
- *Validity.* The agreed shard X satisfies the specified constraint function C, e.g., $\forall i \in \{1...k\}, \forall x_i^j \in X_i, C(x_i^j) = 1$.
- *Scalability.* The value of k grows almost linearly with the size of the network.

The goal of sharding is to split the network into multiple committees, each processing a separate set of transactions (e.g., X_i) called a shard, and the number of shards k grows near linearly on the size of a network. Each shard needs to get an agreement localized within a small committee, which makes the consensus procedure more efficient. Typically, the computation and bandwidth used per node stay constant regardless of n and k. For instance, in blockchain, once the network agrees on the set X, it can create a cryptographic digest of X and form a hash-chain with previously agreed sets in the previous runs of Π, which serve as a distributed ledger of transactions.

3.2 Sharding Overview

Typically, the sharding protocol proceeds in epochs, each of which decides on a set of values $X = \bigcup_{i=1}^{2^s} X_i$ where 2^s is the number of subsets X_i. The key idea is to automatically parallelize the available computation power, dividing it into several smaller committees, each processing a disjoint set of transactions or shards. We take Elastico [100] as an example. The number of committees grows proportionally to the total computation power in the network. All committees, each of which has a small constant number c of members, run a classical BFT consensus protocol internally to agree on one block. For a decentralized system, it needs first to define the membership, and there exist several ways to resolve a membership, e.g., proof-of-work (PoW) [62], proof-of-stake (PoS) [87], proof-of-storage [142], and proof-of-personhood [24]. A permissionless sharding protocol typically consists of five critical components in each consensus round.

1). Identity establishment and committee formation. To join in the protocol, each node needs to establish an identitye.g., an identity consisting of a public key, an IP address and a proof-of-work (PoW) solution. Each node then is assigned to a committee corresponding to its established identity. In this process, the system needs to prevent the Sybil identity [60]. However, for a permissioned blockchain, it does not require this process.

2). Overlay setup for committees. Once the committees are formed, each node communicates to discover the identities of other nodes in its committee. For a blockchain, an overlay of a committee is a fully connected subgraph containing all the committee members. Typically, this process can be done with a gossip protocol [70].

3). Intra-committee consensus. Each node within a committee runs a standard consensus protocol to agree on a single set of transactions. In this process, all honest members must agree on the proposed block within its committee.

4). Cross-shard transaction processing. The transaction should be atomically committed in the whole system. For cross-shard transactions, the related shards need to get consistency. Typically, this process requires a kind of "relay" transaction to synchronize among related shards.

5). Epoch reconfiguration. To guarantee the security of the shards, the shards need to be reconfigured, requiring a randomness. This randomness will be used for the next epoch.

The above five points are the most critical components for a permissionless blockchain sharding.

To design a sharding protocol, it needs to deal with several key challenges. The first challenge is how to *uniformly* split all nodes into several committees so that each committee has the majority honest with high probability. Good randomness is a critical component to partially address this challenge, which provides high-entropy output [49]. However, achieving good randomness in a distributed network is a known hard problem. Section 5 will provide a detailed discussion on epoch randomness. The state-of-the-art solution can only tolerate a small fraction of maliciousness (e.g., 1/6), with excessive message complexity [7]. Typically, the adversary is not static and can adaptively observe all the protocol runs. The second challenge is how to guarantee that the adversary does not gain a significant advantage in biasing its operations or creating Sybil identities (if in public blockchain). Thus, due to the Byzantine faults and network delays in real networks, the sharding protocol must tolerate a varied rate of nodes creation and inconsistency in views of committee members. For a permissionless blockchain, the protocol also needs to deal with one more challenge since the nodes have no inherent identities or external PKI to trust. A malicious node can simulate many virtual nodes, thereby creating a large set of *sybils [108]*. Thus, the protocol must provide an effective mechanism to establish their identities to limit the number of Sybil identities created by malicious nodes.

4 CONSENSUS PROTOCOLS

Sharding on blockchain requires consensus protocols to agree on the proposed blocks. However, capturing a representative and longitudinal view of a topic in blockchain consensus is challenging [13]. Different consensus protocols function differently in the overall sharding procedure. This section presents the state-of-the-art consensus protocols for blockchain sharding in a *general* way.

4.1 Consensus Classification

In general, protocols can be put in two categories when being used in the blockchain sharding: *PoX* and *BFT*. We know *Proof-of-Work (PoW)* mechanism on Bitcoin [107] and *Proof-of-Stake (PoS)* on Ethereum [85]. Technically speaking, PoW and PoS are not the *decent* "consensus protocol", whose mechanisms are used for determining the membership or the stake in a Sybil-attack-resistant fashion. Due to historical reasons, e.g., Bitcoin used PoW as a "consensus" protocol to build a bitcoin blockchain, we literally categorize them into consensus protocols. For example, in a hybrid consensus (e.g., ByzCoin [90] and *Hybrid Consensus* [110]), the decent consensus protocol (the algorithm for agreement on a shared history) is separable from and orthogonal to the membership Sybil-resistance scheme (e.g., PoW). Here we use *Proof-of-X (PoX)* to represent all alternatives of proof-of-something (including PoW and PoS), and use *BFT* to represent Byzantine-based consensus protocols. In a sharding scheme, both PoX and BFT work together to achieve the consensus process. Roughly speaking, both protocols have different tasks in an *overall* sharding scheme, which is a *dynamic* committee based scheme. PoX is typically used for committee formation (e.g., PoW in Elastico [100]) to establish the committee members and these corresponding identifies, while BFT is used for the intra-committee consensus, which is used within a committee to form the blocks. Thus, it is necessary to introduce both PoX and BFT separately.

4.1.1 PoX. Most PoX-based consensus protocols require that the participating node has some kinds of efforts or resources to prove its validity as a miner. We take PoW and PoS as examples to illustrate the PoX mechanisms.

PoW is also called *Nakamoto* consensus in blockchain after its originator [62], proposed in 1992, for spam Email protection. In PoW, the nodes that generate hashes are called *miners* and the process is referred to as *mining*. When applying PoW as a general consensus in blockchain, it is subject to various kinds of attacks [107], such as forks, double-spending attacks, and 51% attacks. These are the general problems in PoW consensus. However, when implementing PoW into blockchain sharding protocols, due to running PoW locally, special care is required, e.g., *selfish mining* [65]. Selfish mining allows colluding miners to generate more valid blocks than their computing power would normally allow if they were following the standard protocol. These valid blocks are typically generated ahead of time, so that the colluding miners withhold blocks that they have found, and then select

a favorite one to maximize these advantages, e.g., controlling one shard. Thus, applying PoW into blockchain sharding requires an agreed epoch randomness for each epoch. Still, most of the state-of-the-art sharding protocols use PoW to establish the membership for a shard.

Compared to PoW, PoS protocols replace wasteful computations with useful "work" derived from the alternative commonly accessible resources. For example, participants of PoS vote on new blocks weighted by their in-band investment such as the amount of currency held in the PPCoin blockchain [87]. In general, PoS has a candidate pool which contains all qualified participants called stakeholders (e.g., the amount of stake is larger than a threshold value) [17] [57]. A common approach is to randomly elect a leader from the stakeholders, which then appends a block to the blockchain. However, in blockchain sharding, PoS may be subject to the *grinding* attacks [45], in which a miner re-creates a block multiple times until it is likely that the miner can create a second block shortly afterward. We should mention that PoS is not just one but instead a collection of protocols. There exist many PoS alternatives, such as Algorand [72], Ouroboros [85], Ouroboros Praos [57], Ethereum [139], etc.

Besides the main PoS protocol, there exist other PoX-based alternatives, which require *miners* to hold or prove the ownership of assets. We list three alternatives: *proof-of-deposit (PoD)* [93], *proof-of-burn (PoB)* [111] and *proof-of-coin-age (PoCA)* [86]. Readers are referred to the corresponding papers for their details.

4.1.2 BFT. Most shard-based systems use classic BFT consensus protocols, e.g., PBFT, as its intra-shard consensus protocol. In this section, we focus on discussing the potential BFT consensus protocols, or their novel compositions which can be tailored for use as the consensus protocols, in blockchains. Roughly speaking, BFT protocols can be classified into two categories: leader-based BFT and leaderless BFT. Most BFT protocols are leader-based, e.g., PBFT or BFT-SMaRt [18]; and leaderless protocols include SINTRA [32] and HoneyBadger [106].

Actual systems that implement PBFT or its variants are much harder to find than systems which implement Paxos/VSR [131]. *BFT-SMaRt* [126], launched around 2015, is a widely tested implementation of BFT consensus protocols. Similar to Paxos/VSR, Byzantine consensus, such as PBFT and BFT-SMaRt, expects an eventually synchronous network to make progress. Without this assumption, only randomized protocols for Byzantine consensus are possible, e.g., SINTRA (relying on distributed cryptography) [32] and HoneyBadger [106], which can achieve ennventual consensus on an asynchronous network.

Still, many well-known blockchain projects use PBFT and BFT-SMaRt protocols. For example, *Hyperledger Fabric* [3] and *Tendermint Core* [26] implement PBFT as these consensus protocols; *Symbiont* [129] and *R3 Corda* [80] use BFT-SMaRt as their consensus protocols. We briefly discuss these two leader-based BFT consensus protocols, which can be used as intra-shard consensus process.

PBFT. PBFT can tolerate up to 1/3 Byzantine faults. We briefly describe its consensus procedures. One replica, the *primary*/*leader* replica, decides the order for clients' requests, and forwards them to other replicas, the *secondary* replicas. All replicas together then run a three-phase (pre-prepare/prepare/commit) agreement protocol to agree on the order of requests. Each replica processes every request and sends a response to the corresponding client. The PBFT protocol has the important guarantee that safety is maintained even during periods of timing violations, progress only depends on the leader. On detecting that the leader replica is faulty through the consensus procedure, the other replicas trigger a *view-change* protocol to select a new leader. The leader-based protocol works very well in practice and is suitable in blockchain, however, it is subject to scalability issues.

BFT-SMaRt. BFT-SMaRt implements a BFT total-order multicast protocol for the replication layer of coordination service [18]. It assumes a similar system model as BFT SMR [36] [46]: $n \geq 3f + 1$ replicas to tolerate f Byzantine faults, and unbounded number of faulty-prone clients and eventual synchrony to ensure liveness. Typically, the BFT-SMaRt consists three key components: Total Order Multicast [123], State Transfer [19], and Reconfiguration [29]. We refer interested readers to [19, 29, 123] for the details.

Besides the above legacy leader-based BFT protocols and the mentioned BFT protocols in Section 2.2, there exist several variants or newly invented algorithms, e.g., Hotstuff [140], Tendermint [26], and Ouroboros-BFT [84]. Due to the page limit, we refer interested readers to the corresponding references for the details.

We now briefly discuss the leaderless BFT protocols. This type of BFT protocols mainly target on the asynchronous settings, which are based on the randomized atomic broadcast protocols. Unlike existing weakly/partially synchronous protocols, in an asynchronous network, messages are eventually delivered but no other timing assumption is made, as defined in Section 2.1. We take SINTRA [32] and HoneyBadger [106] as examples to describe the leaderless BFT protocols.

SINTRA [32]. SINTRA is a Secure INtrusion-Tolerant Replication Architecture for coordination in asynchronous networks subject to Byzantine faults. It is a system implementation based on the asynchronous atomic broadcast protocol [30], which consists of a reduction from atomic broadcast (ABC) to common subset agreement (ACS), as well as a reduction from ACS to multi-value validated agreement (MVBA).

Security is achieved through the use of threshold public-key cryptography, in particular through a cryptographic common coin based on the Diffie-Hellman problem that undelies the randomized protocols in SINTRA.

HoneyBadger [106]. HoneyBadgerBFT essentially follows asynchronous secure computing with optimal resilience [16], which uses reliable broadcast (RBC) and asynchronous binary Byzantine agreement (ABA) to achieve ACS. HoneyBadger cherry-picks a bandwidth-efficient, erasure-code RBC (AVID broadcast) [33] and the most efficient ABC to realize. Specifically, HoneyBadger uses threshold signature to provide common coins for randomized ABA protocol, which achieves a higher throughput by aggressively batching client transactions.

Besides the above two leaderless BFT protocols, there exist some other peer-reviewed and non-peer-reviewed works, such as BEAT [61], and DBFT [51].

4.2 Committee Configuration

In the sharding protocol, the membership of a shard is dynamically changed in each epoch to guarantee safety and security. A reconfigurable committee needs some mechanisms to track committee membership. This is related to how to configure the committees. Typically, there are four ways to configure a committee within the consensus process: static, rolling (single), full, and rolling (multiple).

Static: In a static setting, the committee members are not periodically changed, which is a typical configuration in permissioned systems. For example, Hyperledger [3] and RSCoin [55] are based on this setting, where committee members have known and trusted identities and its threat model does not include Sybil attacks.

Rolling (Single): The committee is updated in a sliding window fashion, where new nodes are added to the current committee and the oldest members are ejected. ByzCoin [90] adopts this scheme, in which each node has a voting power proportional to the number of mining blocks it has in the current window.

Full: This scheme is a lottery-based mechanism, such as Algorand [72] and SnowWhite [54], to select the committee members for each epoch using randomness generated based on previous blocks.

Rolling (Multiple): The committee swaps out multiple members each time. For example, Omniledger [91] uses cryptographic sortition to select a subset of committees to be swapped out and replaced with new members. This is done in a way that the ratio between honest and Byzantine members in a committee is maintained.

We should mention that many blockchain mechanisms for committee configuration are not orthogonal and potentially complementary, instead of mutually exclusive. For example, a large HyperLedger-like permissioned system could serve as a big "directory" from which an OmniLedger-like random committee selection could take place. Similarly, a ByzCoin-like rolling committee selection mechanism based on PoX (e.g., PoW or PoS) could be used to drive the selection of multiple independent committees for OmniLedger-like sharded consensus, not just a signle committee as in ByzCoin.

In a sharding-based protocol, to maintain the committee's safety and security, it typically adopts either *full* or *rolling (multiple)* committee configuration schemes. To configure or reconfigure the committees, a good epoch randomness is required.

5 EPOCH RANDOMNESS

In blockchain sharding protocols, when multiple nodes are involved in a consensus protocol, an important issue is how the participating nodes are assigned to which committee so that the generated committee is "fair". For example, each generated committee requires that it has a majority of honest nodes, and the ratio of faulty nodes should not exceed a threshold that the consensus protocol specified for that shard. One approach to assigning nodes to committees is done statically according to a specified policy, in which it assumes the existence of a random source or a trusted third party, e.g., RSCoin [55]. However, such approach can be problematic in a permissionless setting, which requires a shared random coin [47] [73]. Another approach is to dynamically allocate nodes to committees. This dynamic allocation should be a randomized process, aiming to stop an adversary from concentrating its presence in one committee, and exceeding the Byzantine tolerance threshold. However, generating good randomness in a distributed manner is a known hard problem. For example, the distributed random number generator in [7] can only tolerate up to 1/6 fraction of Byzantine nodes, while still incurring a high message complexity. There exist other randomness generation schemes with different goals or sychrony [83] settings, such as AVSS [30] and APSS [143] for asynchronous communication model, RandHound and RandHerd [127] for scalability in synchronous communication model. In this section, we discuss the potential epoch randomness for sharding-based protocols, and summarize the start-of-the-art epoch randomness generation for blockchain.

5.1 Properties of Epoch Randomness

To generate a seed for sharding securely without requiring a trusted randomness beacon [55] or binding the protocol to PoX, a good distributed randomness generation is required to

meet with several features: public-verifiability, unbiasability, unpredictability, and availability.

1). *Public-Verifiability:* A third party, e.g., not directly partaking processes, should also be able to verify generated value. As soon as a new random beacon value becomes available, all parties can verify the correctness of the new value using public information only.

2). *Bias-Resistance:* This is the assurance that any single participant or a colluding adversary cannot influence the future randomness beacon values to its own advantage.

3). *Unpredictability:* Participants (either correct or adversarial) should not be able to predict or precompute future random beacon values in advance.

4). *Availability:* This property shows that any single participant or a colluding adversary should not be able to prevent the progress.

5.2 Randomness Generation Methods

Roughly speaking, there exist several ways to generate randomness, which can be considered as the baseline of bias-resistance randomness generation. This section introduces these baselines, including Verifiable Random Function (VRF) [105], Verifiable Secret Sharing (VSS) [67], Public Verifiable Secret Sharing (PVSS) [124], and Verifiable Delay Functions (VDF) [21] [113].

5.2.1 VRF. Intuitively, the idea behind a VRF is that Alice asks Bob to compute a function f_s on some input x. Only Bob is able to compute f_s as its result is dependent on some secret value s, which only Bob knows. The result $v = f_s(x)$ has the property of being unique and computationally indistinguishable from a truly random string v' of equal length. Alice wants to be sure that Bob indeed provided the unique correct result of the computations [14]. Formally, VRFs address the issue of unverifiability of Pseudo-Random Functions (PRFs). Consider the case where a party computing $f_s(x_1), f_s(x_2), ..., f_s(x_n)$ claims the corresponding outputs are $o_1, o_2, ..., o_n$. Without knowledge of s, an observer cannot verify that applying f_s to x_i indeed yields the corresponding output o_i. As soon as s gets published, future output values are not indistinguishable from truly random strings anymore. They get fully predictable and can be efficiently computed by any party.

To obtain verifiability without compromising the unpredictability property of future outputs, a party knowing the seed s publishes $v = f_s(x)$ together with a proof $proof_x$. This proof allows verification of the fact that $v = f_x(x)$ indeed holds without revealing s. It is crucial that a party knowing s can only construct a valid proof for a unique v for every x [105]. However, for the proof itself, there is no uniqueness requirement. Some proposed solution is based on interactive zero-knowledge proofs [105]. However, interactive zero-knowledge proofs incur high communication complexity.

5.2.2 VSS. Secret sharing is a scheme to distribute a secret S among a certain number of participants, each one receiving a part of the secret, called a share. Shares can be combined by collaborating participants to reconstruct the original secret. A (t, n)-secret sharing scheme is that any group of t (or more) out of n participants can recover S from their shares. Shamir's secret sharing protocol [120] is based on polynomial interpolation. The key idea behind it is the fact that given t points $(x_1, y_1), (x_2, y_2), ..., (x_t, y_t)$ with different x-coordinates, there is a unique polynomial $p(x)$ of degree $(t-1)$ going through all of the points. However, Shamir's secret sharing protocol is based on an important assumption: the participants assume that they are given correct shares. And this limits the ability to apply this scheme in, e.g., fault-tolerant or even trust-less distributed systems. For example, this assumption does not hold in Byzantine fault tolerance systems. Thus, a verifiable secret sharing (VSS) is required to protect against malicious dealers/participants.

5.2.3 PVSS. A PVSS scheme [124] [118] makes it possible for any party to verify secret-shares without revealing any information about the secrets or the shares. During the share distribution phase, for each trustee i, the dealer produces an encrypted share $E_i(s_i)$ along with a non-interactive zero-knowledge proof (NIZK) [41] to prove that $E_i(s_i)$ correctly encrypts a valid share s_i of s. During the reconstruction phase, trustees recover s by pooling t properly-decrypted shares. They then publish s along with all shares and NIZK proofs showing that the shares were properly decrypted. There also exist some optimized PVSS schemes, such as SCRAPE [35]. Typically, PVSS runs in three steps:

1). The dealer chooses a degree $t-1$ secret sharing polynomial $s(x) = \sum_{j=0}^{t-1} a_j x_j$ and creates, for each trustee $i \in \{1, ..., n\}$, an encrypted share $\hat{S}_i = X_i^{s(i)}$ of the shared secret $S_0 = G^{s(0)}$. The dealer also creates commitments $A_j = H^{a_j}$, where $H \neq G$ is a generator of g, and for each share a NIZK encryption consistency proof $\hat{P}_i$, Afterwards, the dealer publishes $\hat{S}_i$, $\hat{P}_i$ and A_j.

2). Each trustee i verifies his share $\hat{S}_i$ using $\hat{P}_i$ and A_j, and if valid, publishes the decrypted share $S_i = (\hat{S}_i)^{x_i^{-1}}$ together with z NIZK decryption consistency proof P_i.

3). The dealer checks the validity of S_i against P_i, discards invalid shares and, if there are at least t out of n decrypted shares left, recovers the shared secret S_0 through Lagrange interpolation.

We should notice that VRFs play a different role from VSS and PVSS: VRFs allow individual parties to produce verifiable randomness, while both VSS and PVSS allow groups

of parties to produce collective randomness, a.k.a "common coins".

As a brief comparison between VSS and PVSS, VSS aims to resist malicious share holders, in which there is a verification mechanism for each share holder to verify validity of its share, while in PVSS, not just the participants can verify their own shares, but anybody can verfiy that the participants received correct shares. However, most existing PVSS schemes are complex and inefficient, especially in computation. PVSS schemes are typically "single-use", while VSS schemes and the distributed key generation (DKG) algorithms built from them can produce multi-use distributed threshold key pairs.

5.2.4 VDF. Essentially, a verifiable delay function (VDF) requires a specified number of sequential steps to evaluate, yet produce a unique output that can be efficiently and publicly verified. VDFs have many applications in decentralized systems, including public randomness beacons, leader election in consensus protocols, and proofs of replications. A VDF is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that takes a prescribed time to compute, even on a parallel computer. However, once computed, the output can be quickly verified by anyone. Moreover, every input $x \in \mathcal{X}$ must have a unique valid output $y \in \mathcal{Y}$. Specially, a VDF that implements a function $\mathcal{X} \rightarrow \mathcal{Y}$ is a tuple of three algorithms:

- $Setup(\lambda, T) \rightarrow pp$ is a randomized algorithm that takes a security parameter λ and a time bound T, and outputs public parameters pp.
- $Eval(pp, x) \rightarrow (y, \pi)$ takes an input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$ and a proof π.
- $Verify(pp, x, y, \pi) \rightarrow \{accept, reject\}$ outputs $accept$ if y is the correct evaluation of the VDF on input x.

If $(y, \pi) \leftarrow Eval(pp, x)$ then $Verify(pp, x, y, \pi) = accept$, for all $x \in \mathcal{X}$ and pp output by $Setup(\pi, T)$. Besides, a VDF must satisfy three properties: ϵ-evaluation time, sequentiality and uniqueness. We refer interested readers to [21, 22, 113] for the details.

Besides the above randomness generation baselines, there exist other works, such as random zoo [94], deterministic threshold signatures [20] and distributed key generation [83].

5.3 Comparison

Epoch randomness generation in sharding protocols can be treated as a separate module to provide randomness, so that the node can be fairly assigned to the shards according to the public randomness. Thus, any efficient randomness generation algorithm can be implemented as a separated module.

We provide a comparison of the state-of-the-art epoch randomness generation schemes, and discuss these approaches. In our comparison, we do not only consider the protocols specifically targeted at implementing random beacons, but also by including approaches that can readily provide random beacon functionality as a product of their intended applications, such as a provision of a distributed public ledger. Our comparison mainly focuses on the network models, its achieved properties, complexity evaluation metrics, and the baseline technology. However, we must mention that some characteristics were not specified or not available, so we left them blank. Table 1 shows a comparison for generating public-verifiable randomness for blockchain. About the complexity evaluation, n refers to the number of the participants in the overall network, and if the protocols are based on clusters/subsets, c denotes the size of some subset of nodes. And then the value c is protocol dependent, and is typically a constant and negligible factor for the resulting complexity in practice.

6 CROSS-SHARD TRANSACTIONS

To scale blockchain, transactions need to be distributed among multiple committees (or shards), and each shard processes a subset of transactions in parallel. Typically, a transaction may have multiple inputs and outputs. However, due to sharding technology, the inputs and outputs of a transaction might be in different shards, and these transactions are called cross-shard (or *inter-shard*) transactions. Due to random distribution of the transactions in sharding protocols, a cross-shard transaction can be considered as a global transaction, which should be executed by different shards. To achieve a global consistency among different shards, we need to carefully handle the cross-shard transactions. Taking Unspent Transaction-Output (*UTXO*) model as an example, it is expected that the majority of transactions (e.g., more than 90% in [91]) are cross-sharded in a traditional model, where UTXOs are randomly assigned to shards for processing [100] [55]. For the *Account/Balance* transaction model, the cross-shard transactions also can reach up to 90% when the number of shards is more than 64 [137].

To enable value transfer between different shards thereby achieving shard interoperability, supporting for cross-shard transactions is crucial in any sharded-ledger system. In this section, we first describe a general transaction model, Unspent Transaction-Output (*UTXO*), and present its potential issues in blockchain sharding protocols. Then we discuss potential techniques (e.g., atomic commit) to deal with cross-shard transactions. Finally, we present the state-of-the-art approaches to cope with the cross-shard transactions in sharding.

Table 1: A comparison for generating public-verifiable randomness for blockchain

	Communication Model	*Trusted Dealer or DKG Required*	*Crypto-Primitive*	*Liveness/Failure Probability*$^{\triangledown}$	*Unpredictability*	*Bias-Resistance*	*Comm. Complexity (overall protocol)*	*Comp. Complexity (per node)*	*Verif. Complexity (per passive verifier)*
Cachin et al. [31]	Async	yes	uniq. thr. sig.	✓	✓	✓	$O(n^2)$	$O(n)$	$O(1)$
RandShare [127]		no	PVSS	✗†	✓	✓	$O(n^3)$	$O(n^3)$	$O(n^3)$
Algorand [72]	Semi-Syn	no	VRF	10^{-12}	↗	✗	$O(cn)^*$	$O(c)^*$	$O(1)^*$
Ouroboros Praos [57]		no	VRF	✓	↗	✗	$O(n)^*$	$O(1)^*$	$O(1)^*$
Ouroboros [85]	Syn	no	PVSS	✓	✓	✓	$O(n^3)$	$O(n^3)$	$O(n^3)$
Proof-of-Work [107]		no	hash func.	✓	↗	✗	$O(n)$	very high$^{\wr}$	$O(1)$
Proof-of-Delay [27]		no	hash func.	✓	✓	✓	$O(n)$	very high$^{\wr}$	$O(log\Delta)^{\circ}$
Caucus [10]		no	hash func.	✓	↗	✗	$O(n)$	$O(1)$	$O(1)$
Dfinity [79]		yes$^{\oplus}$	BLS sig.	10^{-12}	✓	✓	$O(cn)$	$O(c)$	$O(1)$
Scrape [35]		no	PVSS	✓	✓	✓	$O(n^3)$	$O(n^2)$	$O(n^2)$
RandHound [127]		no	PVSS	0.08%	✓	✓	$O(c^2n)$	$O(c^2n)$	$O(c^2n)$
RandHerd [127]		yes$^{\oplus}$	PVSS/Cosi	0.08%	✓	✓	$O(c^2logn)^{\ddagger}$	$O(c^2logn)$	$O(1)$
HydRand [117]		no	PVSS	✓	✓↗	✓	$O(n^2)$	$O(n)$	$O(n)$

$^{\triangledown}$ provides an upper bound of failure probability for the parameterized protocol.

* represents that the randomness generation approach is not in a standalone way, it requires additional communication and verification steps for underlying consensus protocols or implementation of e.g., bulletin board. In this table, these steps are not counted into the complexity.

↗ provides the probabilistic guarantees for unpredictability, which quickly, e.g., exponentially in the waiting time, get stronger as the longer a client waits after it commits to using a future protocol output. However, in HydRand, the unpredictability can be reached with certainty only after f rounds.

$^{\oplus}$ In Dfinity and RandHerd, nodes are split into smaller groups, and within each of these groups, a distributed key generation protocol is required.

† means that the protocol only provides liveness with additional synchrony assumption.

‡ depends on the relation between n and c. For example, assume that each node only sends a single message during the process of generating a round's randomness, already yields a complexity of $O(n)$, which is higher than the stated $O(c^2logn)$ for a constant group size c and large n.

$^{\circ}$ means the verification process is executed within a smart contract via an interactive challenge/response protocol. The logarithmic complexity $O(\Delta)$ depends on security parameter Δ.

$^{\wr}$ shows the complexity is not dependent on the number of nodes n.

6.1 Transaction Model

UTXO model is adopted by most blockchain protocols and distributed applications. It represents each step in the evaluation of a data object as a separate *atomic state* of the ledger. Such a state is created by a transaction and destroyed (or "consumed") by another unique transaction occurring later [3]. More specifically, in a typical UTXO model, an input represents the value that is to be spent and output represents the new value that is created in response to the input values' consumption. We can think of inputs and outputs representing different phases of the state of the same asset (e.g., in asset management), where state includes its ownership (or shares). Clearly, an input can be used only once, and stops being considered in the system.

In a UTXO model, *input* fields implicitly or explicitly refer *output* fields of other transactions that have not yet been spent. At the validation time, verifiers need to ensure that the outputs referenced by the inputs of the transactions have not been spent and upon transaction-commitment we see them as spent. However, in a multi-shard system, some transactions might involve a coordination between multiple shards. Such transactions might require to access or manipulate the state that is handled by different shards. The inter-shard consensus ensures that this takes place consistently and atomically across all involved shards.

A simple but inadequate strawman approach to a cross-shard transaction, is to concurrently send a transaction to all the corresponding shards for processing. However, for

a cross-shard transaction, due to the separated verification processes, some shards might commit this transaction while others might abort it. In such a case the UTXOs at the shard who accepted the transactions are lost as there is no straightforward way to roll back a half-committed transaction, without adding exploitable race conditions. Thus, we require to ensure the consistency of transactions between shards, to prevent double spending and to prevent unspent funds from being locked forever.

6.2 Atomic Commit

In multi-shard blockchain, it requires to guarantee the global transactions with the properties of ACID [134]: Atomicity, Consistency, Isolation, and Durability. Atomic Commitment (AC) protocol was intially proposed to handle the global ACID transactions [78]. To ensure the transaction atomicity in a blockchain sharding, we require the participants to agree on *one* output for the transaction: either commit or abort, but not both.

One of the earliest and most commonly used protocols for atomic commitment is the two-phase commit (2PC) protocol [74]. In a 2PC protocol, the global transaction manager (or called coordinator node) sends a "prepare" message to all local transactions. The local transactions try to become ready to commit, i.e., reach the *ready* state. In this state, a local transaction has successfully finished all its actions. To be able to follow a global commit decision, the changes of the local transactions are written to a stable storage. Different to the committed state, it is still possible to abort a local transaction in the *ready* state [77]. In other words, the local transaction is able to follow either a global commit or abort decision.

When it is required that every correct participant eventually reaches an outcome despite the failure of other participants, the problem is called *Non-Blocking Atomic Commitment (NB-AC)* [11]. Solving this problem enables correct participants to relinquish resources (e.g., locks) without waiting for crashed participants to recover. The 2PC algorithm solves AC but not NB-AC, whereas the three-phase commit (3PC) algorithm [121] [122] solves NB-AC in synchronous systems (when communication delays and process relative speeds as bounded). The 3PC protocol introduces an additional *pre-commit* state between the *ready* and *commit* states, which ensures that there is no direct transaction between the non-committable and committable states. This simple modification makes the 3PC protocol non-blocking under node failure. However, compared to the 2PC protocol, the 3PC protocol acts as the major performance suppressant in the design of efficient distributed systems. It can be easily observed that the addition of the *pre-commit* state leads to an extra phase of communication among the nodes. Thus, it is necessary to design an efficient commit protocol for geo-scale systems.

However, neither 2PC nor 3PC can be directly applied to the blockchain sharding schemes without modification. For different blockchain sharding schemes, they might have different assumptions among the shards, e.g., the trustworthiness among shards. A practical cross-shard commit approach depends on its assumptions and the threat models used. For example, Interledger [130] protocol enables transfers between ledgers, and ledger-provided escrow removes the need to trust these *connectors* (e.g., each connector functions as a trusted third party to provide the service to the payment sender [81]). Analogized to the blockchain sharding scheme, it assumes that different shards (or alternatively blockchain) that we want to perform atomic transactions across are mutually distrustful, e.g., one might fail to be secure and/or live. The mutual distrusts can further lead to DoS "account lockout" attacks, which is why all these Interledger-type protocols require complex timeout-based recovery mechanisms. In contrast, OmniLedger relies on the fact that *all* shards can be assumed "by construction" to be both safe and live, which means that the simple 2PC approach works fine in that context, and the NB-AC problem does not need to be solved in that threat model. But in OmniLedger the shards have to trust each other. If we weaken the security of OmniLedger's shard selection so that shards no longer fully trust each other, then we need to bring back more complex cross-shard commit protocols.

Thus, for different blockchain sharding schemes, they might have different mechanisms to deal with the the cross-shard transactions. We will discuss these different solutions for specific sharding schemes.

6.3 Methods to Deal with Cross-shard Transactions

Instead of presenting all possible AC protocols, this section presents several state-of-the-art schemes to deal with cross-shard transactions. Some of these schemes do not use the term "shard" but instead using "committee" to deal with the cross-committee transactions, both have the same meaning, i.e., one transaction involving multiple independent entities. However, some sharding protocols, such as Elastico, do not provide a clear or separated process to deal with the cross-shard transactions.

6.3.1 RSCoin. RSCoin [55] is a cryptocurrency framework in which central banks maintain complete control over the monetary supply, but rely on a distributed set of authorities, or *mintettes*, to prevent double-spending. The mintettes process the *lower-level blocks*, which form a potentially cross-referenced chain. The communication between committee members takes place indirectly through the client, and it also

relies on the client to ensure completion of the transactions. A client first gets signed "clearance" from the majority of the mintettes that manage the transaction inputs. Next, the client sends the transaction and signed clearance to mintettes corresponding to transaction outputs. The mintettes check the validity of the transaction and verify signed evidence from input mintettes that the transaction is not double-spending any inputs. If the checks pass, the mintettes append the transaction to be included in the next block. The system operates in epochs: at the end of each epoch, mintettes send all cleared transactions to the central bank, which collates transactions into blocks that are appended to the blockchain.

However, client/user-driven atomic commit protocols are vulnerable to DoS if the client stops participating and the inputs are locked forever. These systems assume that clients are incentivized to proceed to the unlock phase. Such incentives may exist in a cryptocurrency application where an unresponsive client will lose its own coins if the inputs are permanently locked, but do not hold for a general-purpose platform where inputs may have shared ownership. Besides, RSCoin relies on a two-phase commit protocol executed within each shard which, unfortunately, is not Byzantine fault tolerant and can result in double spending attacks by a colluding adversary.

6.3.2 Chainspace. Chainspace [2] is a recently proposed, sharded smart contract platform with privacy built in by design. To enable scalability on Chainspace, the nodes are organized into shards that manage the state of objects, keep track of their validity, and record transactions committed or aborted. The nodes ensure that only valid transactions, consisting of encrypted or committed data, along with the zero-knowledge proofs that assert their correctness, end up on their shard of the blockchain. The nodes communicate with the other shards to decide whether to accept or reject a transaction via inter-shard consensus. Instead of a client-driven approach, Chainspace runs an atomic commit protocol collaboratively between all the concerned committees. This is achieved by making all the committees act as a resource manager for the transactions they manage. To do this, Chainspace proposes a protocol called *Sharded Byzantine Atomic Commit* or *S-BAC*, which combines existing Byzantine agreement and atomic commit protocols in a novel way. In *S-BAC* Byzantine agreement securely keeps a consensus on a shard of $3f + 1$ nodes in total, containing up to f malicious nodes. Atomic commit runs across all shards that contain objects which the transaction relies on. The transaction is rejected unless all of the shards accept to commit the transaction.

6.3.3 OmniLedger. OmniLedger [91] uses a Byzantine shard atomic commit (Atomix) protocol to atomically process transactions across committees, such that each transaction is either committed or aborted. Since both deploying atomic commit protocols and running BFT consensus are unnecessarily complex, atomix uses a lock-then-unlock process. OmniLedger intentionally keeps the shards' logic simple and makes any direct shard-to-shard communication unnecessary by tasking the client with the responsibility of driving the unlock process while permitting any other party (e.g., validators or even other clients) to fill in for the client if a specific transaction stalls after being submitted for processing. Atomix takes a three-step (*initialize/lock/unlock*) protocol to deal with cross-shard UTXO transactions. More specifically, the client first gossips the cross-shard transactions to all their input shards. Then, OmniLedger takes a two-phase approach to handle atomic commit, in which each input shard first locks the corresponding input UTXO(s) and issues a proof-of-acceptance, if the UTXO is valid. The client collects responses from all input committees and issues an "unlock to commit" to the output shard. Interested readers are referred to [91] for the details.

Both OmniLedger and RSCoin heavily rely on the client to proceed with the cross-shard transactions, thus both protocols assume that the client is the honest part. Typically, OmniLedger allows the output committee to verify transactions independently; the transactions have to be gossiped to the entire network and one proof needs to be generated for a batch of transactions, potentially incurring some communication overhead. Besides, OmniLedger depends on the client to retrieve the proof which incurs extra burden on typically lightweight client nodes.

6.3.4 RapidChain. In RapidChain [141], the user does not attach any proof to transaction. It lets the user communicate with any committee who routes transaction to its output committee via the inter-committee routing protocol. RapaidChain considers a simple UTXO transaction $tx =< (I_1, I_2), O >$ that spends coins I_1, I_2 in shard S_1 and S_2, respectively, to create a new coin O belonging to shard S_3. The RapidChain engine executes tx by splitting it into three sub-transactions: $tx_a =< I_1, I_1' >$, $tx_b =< I_2, I_2' >$, and $tx_c =< (I_1', I_2'), O >$, where I_1' and I_2' belong to S_3. tx_a and tx_b essentailly transfer I_1 and I_2 to the output shard, which are spent by tx_c to create the final output O. All thress sub-transactions are single-shard. In case of failures, when, for example, tx_b fails while tx_a succeeds, RapidChain sidesteps atomicity by informing the owner of I_1 to use I_1' for future transactions, which has the same effect as rolling back the failed tx. The cross-shard transaction in RapidChain has largely relied on the inter-committee routing scheme which enables the users and committee leaders to quickly locate

to which committees they should send their transaction. To achieve this, RapidChain builds a routing overlay network, at the committee level, which is based on a routing algorithm of Kademlia [104]. Specifically, each RapidChain committee maintains a routing table of $log(n)$ records which point to $log(n)$ different committees which are distance 2^i for $0 \leq i \leq logn - 1$ away.

For cross-shard transactions in RapidChain, one drawback is that, for each transaction, it creates three different transactions to exchange information among shards. This inherently increases the number of transactions to be proceeded, and the communication by sending the extra transactions back to its input committees also increases. It uses the committee's leader to produce these transactions without considering the status of a leader (e.g., malicious leader). Also, the input committees include the created new transaction into its leader. This behavior to some extent modifies the originality of transactions. Besides, the cross-shard transaction largely depends on the routing algorithm, which is a potential bottleneck.

6.3.5 Discussion. Sharding protocols reduce the communication, computation and storage requirements of each node by dividing the blockchain into partitions, each stored by one of the committees. The cross-shard transactions, however, makes the verification more challenging. Thus, an efficient mechanism to deal with the cross-shard transactions is crucial in the design of a practical blockchain sharding protocol.

Intuitively, there exist some fallacies about the client (who is a coordinator to handle cross-shard transactions) or the shard consensus leader. Taking OmniLedger and RSCoin as examples, one fallacy is that if the client performs some malicious behaviors, then the protocol could not proceed successfully. This is not the fact. Both RSCoin and OmniLedger have backup "garbage collection" strategies that enable the ledger (or other clients) to complete or abort cross-shard transactions that failed or malicious clients might leave uncompleted. It is not a complicated process, and just a matter of ensuring that the "lock" phase records all the cross-shard transaction information that a future garbage-collector or other interested client needs to complete or abort the transaction that has an account of interest locked. Another fallacy is that the OmniLedger uses the leader of a shard to issue and indicate *acceptance* or *rejection*; this might involve some problems, especially if the leader is a malicious one. This is also not true. An OmniLedger shard's leader is merely the leader of a PBFT-sytle Byzantine consensus group, and has no power to carry out any (malicious) behaviors itself without getting them validated by a majority of honest nodes within the same group. In other words, the "accept" or "reject" decision, like all decisions that an OmniLedger shard makes, are products of (and layered on top of) the PBFT state machine, and thus will always be "correct" and "honest" and "non-malicious" because of PBFT, unless the system's basic security invariants are broken, e.g., leading to fully-compromised with too many corrupted nodes.

How to efficiently handle the cross-shard transactions is a fundamental topic in most blockchain sharding protocols. When designing an efficient mechanism to deal with cross-shard transactions, it requires to consider several significant factors, e.g., the atomic commitment scheme within the shard, the communication complexity among the shards (e.g., the number of message exchanges), and the transaction model. Technically, the transaction model affects the cross-shard transaction mechanism significantly. We should notice that for different applications, they might adopt different transaction models. Currently, most of the state-of-the-art sharding protocols are still based on the traditional cryptocurrency-based UTXO model. However, for different transaction models, it might result different storage requirements [135] [136].

Besides the garbage-collection mechanisms, there exist some blockchain protocols, such as SideCoin [92] and RollerChain [43], utilizing the distributed state snapshotting mechanism [40] to record the blockchain's recent status. And this state snapshotting mechanism can be applied into sharding blockchain, e.g., RapidChain, to check the cross-shard transactions much quicker, and it also can be used to reconfigure the committees of next epoch.

7 EPOCH RECONFIGURATION

Sharding protocols partition the consensus nodes into different shards, so that each shard can process the transactions in parallel, and hence improve the scalability of the whole system. However, partitioning the nodes into shards in blockchain sharding introduces new challenges when dealing with the phenomenon of the churn. For example, corrupted nodes could strategically leave and rejoin the shards, so that eventually they can take over one of the shards and break the security guarantees of the blockchain protocol. Moreover, the adversary can actively corrupt a constant number of uncorrupted nodes in each epoch even if no nodes join/rejoin [141]. Most current sharding protocols did not explicitly provide the approaches to deal with the epoch reconfiguration. However, the epoch reconfiguration is critical to guarantee the security of blockchain system.

Clearly, to prevent attacks from the adversary, e.g, corrupting a specific shard, the adversary should not have the knowledge, *in advance*, how the partition (reconfiguration) process works. This requires that the partition process should not be affected by the adversary who do not know which participating nodes will be assigned to which shard ahead. Also, for each shard working correctly, it must guarantee that the majority of participating nodes within each shard

(e.g., at least $2/3$ of the shard members) are honest and follow the consensus protocol. One simple and naive way is to leverage the randomness, discussed in Section 5. By applying the randomness on epoch reconfiguration, the probability of one shard being bad is negligible (e.g., less than 10^{-7}). In this section, we present several state-of-the-art schemes to deal with epoch reconfiguration, which typically rely on the (modified) epoch randomness and the specific mechanisms together. We call epoch reconfiguration and shard reconfiguration interchangeably in this section.

7.1 Hash + Final Committee

One simple and naive approach for epoch reconfiguration is to re-elect all committees periodically faster than the adversary's ability to generate the churn. A previous approach is used to generate *epoch randomness* [8]. However, this solution tolerates at most $1/6$ fraction of malicious nodes and only works for a small network since it essentially bears an excessive message complexity. The cryptographic hash operations can be used to achieve the same purpose at some extent. In the last step of Elastico [100], it takes a similar but optimized approach via the final committee (or called *consensus committee*) to achieve epoch reconfiguration. The final committee at the final step generates a set of random strings used for next epoch. In general, Elastico consists of two main phases for epoch reconfiguration.

In the first phase of the reconfiguration, each member of the final committee chooses a r-bit random string R_i and sends a hash $H(R_i)$ to everyone in that committee. The final committee then runs an interactive consistency protocol to agree on a single set of hash values S [112] and broadcasts S to everyone in the network. This set S contains at least $2c/3$ (where c is the size of the final committee) hash values and serves as a commitment to the random strings. In the second phase, each member of the final committee broadcasts a message containing the random string R_i itself to everyone (i.e., not just to the final committee). This phase starts only after the agreement of S is done, i.e., having $2c/3$ signatures on S. This is to guarantee that honest members release their commitments only after they are sure that the committee has agreed on S and the adversary cannot change its commitment. After the second phase, each node in this system has received at least $2c/3$ and at most $3c/2$ pairs of R_i and $H(R_i)$ from members of the final committee, since the honest members follow the protocol, while the malicious nodes may choose not to release their commitments. Nodes discard any random strings R_i that do not match the commitments $H(R_i)$. Finally, the agreed-to set S is used to configure the next epoch.

However, there exist several weaknesses in this kind of epoch reconfiguration. First, re-generating all the committees is very expensive due to the large overhead of the bootstrapping protocol. Second, maintaining a separate ledger for each committee is challenging when several committee members may be fully replaced in every epoch. Third, the randomness used in each epoch can be biased by an adversary, and hence, compromise the committee selection process and even allow malicious nodes to precompute PoW puzzles. Besides, Elastico requires a trusted setup for generating an initial common randomness that is revealed to all parties at the same time.

7.2 DRG + PoW + Cuckoo Rule

RapidChain adopts a different approach to handle partial issues in Elastico via Cuckoo rule [9] [119]. In general, the epoch reconfiguration has three components: offline PoW, epoch randomness generation, and reconfiguration process. The reconfiguration process uses Cuckoo rule to re-organize only a subset of shard members during the reconfiguration event that shards are balanced with respect to their sizes as nodes join or leave the network.

RapidChain relies on PoW to protect against Sybil attack by requiring *every* node who wants to join or stay in the protocol to solve a PoW puzzle. In each epoch, a fresh puzzle is generated based on the epoch randomness so that the adversary cannot precompute the solutions ahead of the time to compromise the committees. All nodes in RapidChain solve a PoW offline without making the protocol stop and wait for the solution. Thus, the expensive PoW calculations are performed off the critical latency path. The reference committee (C_R) in RapidChain is responsible to check the PoW solutions of all nodes at the start of each epoch, and then agrees on a reference block consisting of the list of all active nodes for that epoch as well as their assigned committees.

To compute an offline PoW solution, an epoch randomness generation process is needed, in which the members of the reference committee run a *distributed random generation (DRG)* protocol to agree on an unbiased random value. C_R includes the randomness in the reference block so that other committees can randomize their epochs. RapidChain uses a verifiable secret sharing (VSS) of Feldman [67] to generate an unbiased randomness within the reference committee. Any new node who wishes to join the system can contact any node in any committees at any time and request the randomness of this epoch as a fresh PoW puzzle.

To assign the nodes to shards, it first maps each node to a random position in $[0, 1)$ using a hash function. Then the range $[0, 1)$ is partitioned into k regions of size k/n, and a committee is defined as the group of nodes that are assigned to $O(log(n))$ regions, for some constant k. Awerbuch and

Scheideler [9] propose the Cuckoo rule to ensure that the set of committees created in the range $[0, 1)$ remain robust to join-leave attacks. Based on this rule, when a node wants to join the network, it is placed at a random position $x \in [0, 1)$, while all nodes in a constant-sized interval surrounding x are moved (or *cuckoo*'ed) to a new random position in $[0, 1)$. It is proved that given $\varepsilon \leq 1/2 - 1/k$ in a steady state, all regions of size $O(log(n))/n$ have $O(log(n))$ nodes (i.e., they are balanced) of which less than $1/3$ are faulty, with high probability, for any polynomial number of rounds.

7.3 VRF + Global Reconfiguration

Similar to Elastico, OminiLedger also runs a global reconfiguration protocol at each epoch, e.g., once a day, to allow new participants to join the protocol. The protocol *generates* identities and assigns participants to shards using a slow identity blockchain protocol that assumes the synchronous channels. In each epoch, a fresh randomness is generated using a bias-resistant random generation protocol that relies on a verifiable random function (VRF) [105] for unpredicatble leader election in a way similar to the lottery algorithm of Algorand [72]. Then, the protocol uses the elected leader as the client in the RandHound [127] protocol to generate the epoch randomness.

More specifically, at the beginning of an epoch, each validator computes a ticket which contains all properly registered validators of the current epoch (e.g., as stored in the identity blockchain) and the view counter. Validators then gossip these tickets with each other for a time δ, after which they lock in the lowest-value valid ticket they have seen thus far and accept the corresponding node as the leader of the RandHound protocol run. Once the validators have successfully completed a run of RandHound and the leader has broadcast randomness together with its correctness proof, each of the registered validators can verify and use this randomness to compute a permutation, and subdivide the result into approximately equally-sized buckets, thereby determining the assignment of nodes to shards.

8 STATE-OF-THE-ART SHARDING PROTOCOLS

This section summaries a comparison of the state-of-the-art blockchain sharding protocols in a more general way. We first summarize and compare several state-of-the-art blockchain sharding protocols, and then briefly discuss other protocols to deal with the scalability in blockchain.

8.1 Comparision of State-of-the-art Sharding Protocols

Table 2 provides a comprehensive comparison for the current classic blockchain sharding protocols. Instead of considering the individual protocols, we map out the landscape by extracting and evaluating the high-level design themes in blockchain sharding schemes. The system designer can have a general overview on these blockchain sharding schemes. In this section, the terms *committee* and *shard* have the same meaning.

In this comparison, we mainly focus on four aspects: protocol settings, intra-committee consensus, inter-committee consensus, as well as safety and their performances. Note that some properties have already been described in the previous sections. The protocol settings show how the protocols set up in an overall perspective, such as committee formation, network model. The intra-committee consensus shows how to achieve a consensus within a committee, and the inter-committee consensus shows how to achieve an agreement among different committees. Finally, we compare their safety aspects and the achieved performance.

Protocol Settings: *Committee formation* refers to the criteria used to allow nodes to join a committee, which describes the mechanisms to establish the membership, e.g., membership based on PoW or PoS. This is an important aspect of decentralized and permissionless systems to thwart Sybil attacks. However, for permissioned blockchain, e.g., RSCoin, we do not need to deal with Sybil attacks, since permissioned systems operate in a *relatively* trust environment where the participating nodes are granted committee membership based on these organizational policy. *Consistency* shows the likelihood that the system will reach a consensus on the proposed value, typically, it can be either strong or weak. In general, classic BFT protocols offer strong consistency, but are subject to the scalability issue. *Network Model* shows the synchrony of the underlying communication network. Typically, the communication networks can be categorized into three types: strongly synchronous, partially synchronous, and asynchronous.

Intra-Committee Consensus: *Committee Configuration* represents how the committee members are assigned to the committee in a single committee setting, e.g., either the members serve on the committee permanently (static) or they are changed at regular intervals (rolling or swap) for the epoch-based protocols. *Incentives* show the mechanisms that keep participating nodes motivated to participate in the consensus process and follow its rules. We distinguish the incentives in two aspects: one is the join process, and the other is the participating process. *Leader* indicates, within a specific committee, where the leader comes from. It can be either elected

Table 2: A comparison for sharding blockchain protocols

		RSCoin [55]	*Chainspace [2]*	*Elastico [100]*	*OmniLedger [91]*	*RapidChain [141]*
Committee Formation		Permissioned	Flexible	PoW	Pow/PoX	Offline PoW
Strong Consistency		✓	✓	✓	✓	✓
Network Model		!	Async	Partial Sync.	Partial Sync.	Sync.
Single Intra-committee Consensus	***Committee Configuration***	Static	Flexible	Full Swap	Rolling (subset)	Partical Swap
	Incentives (join, participate)	(-, -)	(✗, ✗)	(✓, ✗)	(✓, ✗)	(✓, ✗)
	Leader	Internal	Internal	Internal	Internal	Internal
	Msg. Compl†	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
Multiple Inter-committee Consensus	***Inter-Committee Configuration***	✗	✗	Dynamic (Random)	Dynamic (Random)	Dynamic (Random)
	Mediated	Client	✗	!	Client	✗
	Incentives	✗	✗	!	✗	✗
Safety	***TX Censorship Resistance***	✓	✓	✗	✓	✓
	DoS Resistance	✓	✓*	✓	✓	✓
	Adversary Model	33%	33%	33%	33%‡	33%
Performance	***Throughput***	2k tx/s [1]	350 tx/s [2]	16 blocks in 110s [3]	≈10k tx/s [4]	≈7,300tx/s [5]
	Scalable	✓	✓	✓	✓	✓
	Latency	<1s	<1s	110s for 16 blocks	≈1s	8.7s for 7300tx

✓: has property; ✗: does not have property; *: partially has property; −: means the property does not apply to the given category; !: means the value is missing; †: means message complexity.

‡: each shard tolerates 1/3-fraction adversary, and the overall protocol tolerates only 1/4.

[1]: 3 nodes/committee and 10 committee in total; [2]: 4 nodes/committee and 15 committees in total; [3]: 100 nodes/committee and 16 committees in total; [4]: 72 nodes/committee (12.5% adversary) and 25 committees in total; [5]: 250 nodes/committee and 4000 nodes in total.

among the current committee (internally), externally, or flexible (e.g., through the specified smart contracts). For the listed schemes, all leaders come *internally* from its committee members. *Msg. Complexity* shows the communication complexity within one committee at the message level, where n refers to the number of participating nodes.

Inter-Committee Consensus: *Inter-committee configuration* shows how the members are assigned to the committees in a multiple-committee setting, which can be either static or dynamic. A dynamic approach is typically based on the randomness generated from the previous epoch. *Mediated* indicates how to mediate the cross-sharding transactions. It can be optionally mediated by an external resource, e.g., the client. *Incentives* indicates, for mediators, whether they will get some rewards for their mediation efforts.

Safety and Performance: For safety, we focus on the resistance against an adversary. *TX Censorship Resistance* shows the system's resilience to the proposed transactions being suppressed (i.e., censored) by malicious nodes involved in consensus process. *DoS Resistance* represents the resilience of the nodes involved in consensus to Denial-of-service (DoS) attacks. If the participants of the consensus protocol are known in advance, an adversary may launch a DoS attack against them. *Adversary Model* represents the fraction of malicious or faulty nodes that the consensus protocol can tolerate (e.g., the protocol still works correctly despite the presence of such nodes). Note that for different adversary models, it might have different resistance rates. In this comparison, the adversary models are all based on the Byzantine setting. For performance, we target at analyzing its throughout, latency and scalability. *Throughput* is the maximum rate at which transactions can be agreed upon by the consensus protocol; *latency* represents the time it takes from when a transaction is proposed until consensus has been reached on it. *Scalability* shows if the system has the ability to achieve greater throughput when consensus involves a larger number of nodes. All the listed schemes in Table 2 can scale.

8.2 Discussion

Besides the sharding-based blockchain protocols summarized in Table 2, there exist other alternatives to deal with scalability issues, which are conceptually similar to the mentioned sharding-based protocols, e.g., Monoxide [137] and SSChain [42].

Monoxide utilizes the concept of asynchronous consensus zones, in which each zone is conceptually a shard. Instead of utilizing UTXO transaction models, this protocol is based on the account/balance transaction model, which is similar to a bank account model. It proposes an *eventual atomicity* scheme, by relying on the relay transactions, to ensure the atomicity of transactions across zones. For the consensus protocol, Monoxide builds on the PoW scheme in general, and it uses the *Chu-ko-nu mining* scheme, which allows a single PoW solution to create multiple blocks at different zones simultaneously, to ensure the effective mining power in each zone to be at the same level of the entire network. Conceptually, Monoxide can be categorized as a kind of blockchain sharding scheme.

SSChain utilizes a two-layer architecture to eliminate the data migration overhead in reshuffling scheme. In SSChain, participating nodes can freely join in one or more shards without reshuffling network periodically. In this two-layer structure, the first layer is the root chain network, which has a significantly large portion (e.g., over 50%) of computing power over the whole network, while the second layer is the shard networks, in which each shard maintains disjoint ledgers and independently processes a disjoint subset of transactions. In the words, the root chain maintains security of the system, while shards improve the throughput and decrease storage requirements.

There also a large number of non-peer reviewed blockchain sharding protocols in the literature, e.g., Aspen [71], Blockclique [69], Ethereum 2.0 [28], etc. Due to the page limit, the interested reader are referred to the provided references for their details.

It is necessary to briefly discuss the techniques to handle the blockchain scalability (including sharding protocols) in general. There exists two main-stream solutions: off-chain solutions [114] [63] and DAG solutions [115].

Off-chain Solutions. In this solutions, each node holds its transactions locally, referred as "off-chain", and only sends a description or the eventual outcome of these transactions to the "main chain", referred as "on-chain". However, there is no guarantee on the validity of the "off-chain" transactions, either validation node are introduced to validate and endorse these transactions, or economical deposit should be provided for the transactions. And, the validity condition might be compromised due to centralization or the economical constraint. There exist several key chellenges in off-chain solutions, e.g., the way to keep the state consistency (and final conformation of transactions) between "off-chains" and the "on-chain" in real-time (or acceptable time) manner, the centralization and security issues in the "off-chains" which rely on intermediaries to aggregate and settle transactions off-chain.

Directed Acyclic Graph (DAG) Solutions. In DAG, the transactions are not structured in a chain, but in a graph. The validity is dependent on the (directly or indirectly) outgoing edges of the transaction, which represents the nodes that have validated it. A scale-out throughput can be achieved if the acquirement of the complete graph is not obligated for all nodes. And, the validity of the transaction might be compromised due to its dependency on the validators. Also, there exist some probability that the valid transactions are appended to the parasite chains [115].

Sharding Solutions. Besides the common issues discussed in this paper, there exist some potential research topics on blockchain sharding, such as horizontal sharding (e.g., Channels [4]) and heterogeneous sharding (e.g., nodes with different capacity), and application-specific blockchain sharding schemes (e.g., sharding schemes targeted to industrial Internet of Things (IIoT) [136] [138]). Sharding based blockchain systems make trade-offs between the scalability of throughout, storage efficiency, and security [96]. A widely open fundamental question is that *Is there a blockchain design that simultaneously scales throughput, storage efficiency, and security?*

9 CONCLUSION

This paper presents a Systematization of Knowledge for sharding on blockchain. We identified key components and challenges in sharding. The publicly verifiable randomness is critical for placing participating nodes uniformly into shards. Within each shard, a consensus protocol is needed to reach an agreement on the blocks. BFT-based protocols are dominating in existing solutions. For the cross-shard transactions, the protocol needs to guarantee the atomic properties. Finally, a reconfiguration process is needed at the end of an epoch. We analyzed several well-known blockchain sharding protocols and then discussed several potential research directions.

REFERENCES

[1] Abraham, I., Devadas, S., Dolev, D., Nayak, K., and Ren, L. Efficient synchronous byzantine consensus. *arXiv preprint arXiv:1704.02397* (2017).

[2] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., and Danezis, G. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778* (2017).

[3] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich,

Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (2018), ACM, p. 30.

[4] Androulaki, E., Cachin, C., De Caro, A., and Kokoris-Kogias, E. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security* (2018), Springer, pp. 111–131.

[5] Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., and Yakira, D. Helix: A scalable and fair consensus algorithm resistant to ordering manipulation.

[6] Aspnes, J. Randomized protocols for asynchronous consensus. *Distributed Computing 16*, 2-3 (2003), 165–175.

[7] Awerbuch, B., and Scheideler, C. Robust random number generation for peer-to-peer systems. In *International Conference On Principles Of Distributed Systems* (2006), Springer, pp. 275–289.

[8] Awerbuch, B., and Scheideler, C. Robust random number generation for peer-to-peer systems. *Theoretical Computer Science 410*, 6-7 (2009), 453–466.

[9] Awerbuch, B., and Scheideler, C. Towards a scalable and robust dht. *Theory of Computing Systems 45*, 2 (2009), 234–260.

[10] Azouvi, S., McCorry, P., and Meiklejohn, S. Winning the caucus race: Continuous leader election via public randomness. *arXiv preprint arXiv:1801.07965* (2018).

[11] Babaoglu, O., and Toueg, S. Understanding non-blocking atomic commitment. *Distributed systems* (1993), 147–168.

[12] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., and Wuille, P. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* (2014).

[13] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936* (2017).

[14] Bao, F., and Deng, R. H. A signcryption scheme with signature directly verifiable by public key. In *International Workshop on Public Key Cryptography* (1998), Springer, pp. 55–59.

[15] Behl, J., Distler, T., and Kapitza, R. Hybrids on steroids: Sgx-based high performance bft. In *Proceedings of the Twelfth European Conference on Computer Systems* (2017), ACM, pp. 222–237.

[16] Ben-Or, M., Kelmer, B., and Rabin, T. Asynchronous secure computations with optimal resilience. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing* (1994), ACM, pp. 183–192.

[17] Bentov, I., Pass, R., and Shi, E. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive 2016* (2016), 919.

[18] Bessani, A. N., Alchieri, E. P., Correia, M., and Fraga, J. S. Depspace: a byzantine fault-tolerant coordination service. In *ACM SIGOPS Operating Systems Review* (2008), vol. 42, ACM, pp. 163–176.

[19] Bessani, A. N., Santos, M., Felix, J., Neves, N. F., and Correia, M. On the efficiency of durable state machine replication.

[20] Boldyreva, A. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography* (2003), Springer, pp. 31–46.

[21] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In *Annual International Cryptology Conference* (2018), Springer, pp. 757–788.

[22] Boneh, D., Bünz, B., and Fisch, B. A survey of two verifiable delay functions. *IACR Cryptology ePrint Archive 2018* (2018), 712.

[23] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security* (2001), Springer, pp. 514–532.

[24] Borge, M., Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., and Ford, B. Proof-of-personhood: Redemocratizing permissionless cryptocurrencies. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2017), IEEE, pp. 23–26.

[25] Bracha, G. An o (log n) expected rounds randomized byzantine generals protocol. *Journal of the ACM (JACM) 34*, 4 (1987), 910–920.

[26] Buchman, E. *Tendermint: Byzantine fault tolerance in the age of blockchains.* PhD thesis, 2016.

[27] Bünz, B., Goldfeder, S., and Bonneau, J. Proofs-of-delay and randomness beacons in ethereum. *IEEE SECURITY and PRIVACY ON THE BLOCKCHAIN (IEEE S&B)* (2017).

[28] Buterin, V. Ethereum 2.0 spec–casper and sharding, 2018. *Available [online].[Accessed: 30-10-2018].*

[29] Cachin, C. Yet another visit to paxos. *IBM Research, Zurich, Switzerland, Tech. Rep. RZ3754* (2009).

[30] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference* (2001), Springer, pp. 524–541.

[31] Cachin, C., Kursawe, K., and Shoup, V. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology 18*, 3 (2005), 219–246.

[32] Cachin, C., and Poritz, J. A. Secure intrusion-tolerant replication on the internet. In *null* (2002), IEEE, p. 167.

[33] Cachin, C., and Tessaro, S. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)* (2005), IEEE, pp. 191–201.

[34] Cachin, C., and Vukolić, M. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873* (2017).

[35] Cascudo, I., and David, B. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security* (2017), Springer, pp. 537–556.

[36] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS) 20*, 4 (2002), 398–461.

[37] Castro, M., Liskov, B., et al. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.

[38] Catalano, D., Gennaro, R., Howgrave-Graham, N., and Nguyen, P. Q. Paillier's cryptosystem revisited. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM, pp. 206–214.

[39] Chandra, T. D., Griesemer, R., and Redstone, J. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing* (2007), ACM, pp. 398–407.

[40] Chandy, K. M., and Lamport, L. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS) 3*, 1 (1985), 63–75.

[41] Chaum, D., and Pedersen, T. P. Wallet databases with observers. In *Annual International Cryptology Conference* (1992), Springer, pp. 89–105.

[42] Chen, H., and Wang, Y. Sschain: A full sharding protocol for public blockchain without data migration overhead. *Pervasive and Mobile Computing* (2019), 101055.

[43] Chepurnoy, A., Larangeira, M., and Ojiganov, A. Rollerchain, a blockchain with safely pruneable full blocks. *arXiv preprint arXiv:1603.07926* (2016).

[44] Chun, B.-G., Maniatis, P., Shenker, S., and Kubiatowicz, J. Attested append-only memory: Making adversaries stick to their word. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 189–204.

[45] Churyumov, A. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball. org/Byteball. pdf* (2016).

[46] Clement, A., Wong, E. L., Alvisi, L., Dahlin, M., and Marchetti, M. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI* (2009), vol. 9, pp. 153–168.

[47] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS) 31*, 3 (2013), 8.

[48] Correia, M., Neves, N. F., and Verissimo, P. Bft-to: Intrusion tolerance with less replicas. *The Computer Journal 56*, 6 (2012), 693–715.

[49] Corrigan-Gibbs, H., Mu, W., Boneh, D., and Ford, B. Ensuring high-quality randomness in cryptographic key generation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 685–696.

[50] Costan, V., and Devadas, S. Intel sgx explained. *IACR Cryptology ePrint Archive 2016*, 086 (2016), 1–118.

[51] Crain, T., Gramoli, V., Larrea, M., and Raynal, M. Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)* (2018), IEEE, pp. 1–8.

[52] Cristian, F., Aghili, H., Strong, R., and Dolev, D. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation 118*, 1 (1995), 158.

[53] Crosby, S. A., and Wallach, D. S. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium* (2009), pp. 317–334.

[54] Daian, P., Pass, R., and Shi, E. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. Tech. rep., Technical Report. Cryptology ePrint Archive, Report 2016/919, 2017.

[55] Danezis, G., and Meiklejohn, S. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895* (2015).

[56] Dang, H., Dinh, A., Chang, E.-C., and Ooi, B. C. Chain of trust: Can trusted hardware help scaling blockchains? *arXiv preprint arXiv:1804.00399* (2018).

[57] David, B., Gaži, P., Kiayias, A., and Russell, A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2018), Springer, pp. 66–98.

[58] Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), ACM, pp. 1085–1100.

[59] Dolev, D., and Strong, H. R. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing 12*, 4 (1983), 656–666.

[60] Douceur, J. R. The sybil attack. In *International workshop on peer-to-peer systems* (2002), Springer, pp. 251–260.

[61] Duan, S., Reiter, M. K., and Zhang, H. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 2028–2041.

[62] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference* (1992), Springer, pp. 139–147.

[63] Eberhardt, J., and Tai, S. On or off the blockchain? insights on off-chaining computation and data. In *European Conference on Service-Oriented and Cloud Computing* (2017), Springer, pp. 3–15.

[64] Ekberg, J.-E., Kostiainen, K., and Asokan, N. The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy 12*, 4 (2014), 29–37.

[65] Eyal, I., and Sirer, E. G. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM 61*, 7 (2018), 95–102.

[66] Feldman, A. J., Zeller, W. P., Freedman, M. J., and Felten, E. W. Sporc: Group collaboration using untrusted cloud resources. In *OSDI* (2010), vol. 10, pp. 337–350.

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

[68] Fischer, M. J., Lynch, N. A., and Paterson, M. S. Impossibility of distributed consensus with one fault process. Tech. rep., YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, 1982.

[69] Forestier, S., and Vodenicarevic, D. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. *arXiv preprint arXiv:1803.09029* (2018).

[70] Ganesh, A. J., Kermarrec, A.-M., and Massoulié, L. Peer-to-peer membership management for gossip-based protocols. *IEEE transactions on computers 52*, 2 (2003), 139–149.

[71] Gencer, A. E., van Renesse, R., and Sirer, E. G. Service-oriented sharding with aspen. *arXiv preprint arXiv:1611.06816* (2016).

[72] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 51–68.

[73] Glendenning, L., Beschastnikh, I., Krishnamurthy, A., and Anderson, T. Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 15–28.

[74] Gray, J. N. Notes on data base operating systems. In *Operating Systems*. Springer, 1978, pp. 393–481.

[75] Grigg, I. Eos - an introduction. https://eos.io/documents/EOS_An_Introduction.pdf.

[76] Guerraoui, R., Knežević, N., Quéma, V., and Vukolić, M. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems* (2010), ACM, pp. 363–376.

[77] Gupta, S., and Sadoghi, M. Easycommit: A non-blocking two-phase commit protocol. In *Proceedings of the 21st international conference on extending database technology, Open Proceedings, EDBT* (2018).

[78] Haerder, T., and Reuter, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR) 15*, 4 (1983), 287–317.

[79] Hanke, T., Movahedi, M., and Williams, D. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548* (2018).

[80] Hearn, M. Corda: A distributed ledger. *Corda Technical White Paper* (2016).

[81] Hope-Bailie, A., and Thomas, S. Interledger: Creating a standard for payments. In *Proceedings of the 25th International Conference Companion on World Wide Web* (2016), International World Wide Web Conferences Steering Committee, pp. 281–282.

[82] Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S. V., Schröder-Preikschat, W., and Stengel, K. Cheapbft: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), ACM, pp. 295–308.

[83] Kate, A., Huang, Y., and Goldberg, I. Distributed key generation in the wild. *IACR Cryptology ePrint Archive 2012* (2012), 377.

[84] Kiayias, A., and Russell, A. Ouroboros-bft: A simple byzantine fault tolerant consensus protocol. *IACR Cryptology ePrint Archive 2018* (2018), 1049.

[85] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference* (2017), Springer, pp. 357–388.

[86] King, S., and Nadal, S. Peercoin–secure & sustainable cryptocoin. *Aug-2012 [Online]. Available: https://peercoin. net/whitepaper ()*.

[87] King, S., and Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August 19* (2012).

[88] King, V., and Saia, J. Breaking the o (n 2) bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM) 58*, 4 (2011), 18.

[89] King, V., Saia, J., Sanwalani, V., and Vee, E. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm* (2006), Society for Industrial and Applied Mathematics, pp. 990–999.

[90] Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 279–296.

[91] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 583–598.

[92] Krug, J., and Peterson, J. Sidecoin: a snapshot mechanism for bootstrapping a blockchain. *arXiv preprint arXiv:1501.01039* (2015).

[93] Kwon, J. Tendermint: Consensus without mining. *Draft v. 0.6, fall* (2014).

[94] Lenstra, A. K., and Wesolowski, B. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive 2015* (2015), 366.

[95] Li, J., Krohn, M. N., Mazières, D., and Shasha, D. E. Secure untrusted data repository (sundr). In *OSDI* (2004), vol. 4, pp. 9–9.

[96] Li, S., Yu, M., Avestimehr, S., Kannan, S., and Viswanath, P. Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously. *arXiv preprint arXiv:1809.10361* (2018).

[97] Liu, J., Li, W., Karame, G. O., and Asokan, N. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers 68*, 1 (2018), 139–151.

[98] Liu, S., Viotti, P., Cachin, C., Quéma, V., and Vukolić, M. {XFT}: Practical fault tolerance beyond crashes. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 485–500.

[99] Liu, Y., Wang, Y., and Jin, Y. Research on the improvement of mongodb auto-sharding in cloud environment. In *2012 7th International Conference on Computer Science & Education (ICCSE)* (2012), IEEE, pp. 851–854.

[100] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 17–30.

[101] Malkhi, D., and Reiter, M. Byzantine quorum systems. In *STOC* (1997), vol. 97, Citeseer, pp. 569–578.

[102] Malkhi, D., and Reiter, M. Byzantine quorum systems. *Distributed computing 11*, 4 (1998), 203–213.

[103] Maric, O., Sprenger, C., and Basin, D. Consensus refined. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2015), IEEE, pp. 391–402.

[104] Maymounkov, P., and Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 53–65.

[105] Micali, S., Rabin, M., and Vadhan, S. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)* (1999), IEEE, pp. 120–130.

[106] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 31–42.

[107] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system.

[108] Newsome, J., Shi, E., Song, D., and Perrig, A. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of the 3rd international symposium on Information processing in sensor networks* (2004), ACM, pp. 259–268.

[109] Oki, B. M., and Liskov, B. H. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing* (1988), ACM, pp. 8–17.

[110] Pass, R., and Shi, E. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC 2017)* (2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[111] Patterson, R. Alternatives for proof-of-work, part 2: Proof of activity, proof of burn, proof of capacity, and byzantines generals, bytecoin, 2015.

[112] Pease, M., Shostak, R., and Lamport, L. Reaching agreement in the presence of faults. *Journal of the ACM (JACM) 27*, 2 (1980), 228–234.

[113] Pietrzak, K. Z. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference* (2019), vol. 124.

[114] Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[115] Popov, S. The tangle. *cit. on* (2016), 131.

[116] Rosenfeld, M. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009* (2014).

[117] Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. R. Hydrand: Practical continuous distributed randomness. *IACR Cryptology ePrint Archive 2018* (2018), 319.

[118] Schoenmakers, B. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference* (1999), Springer, pp. 148–164.

[119] Sen, S., and Freedman, M. J. Commensal cuckoo: Secure group partitioning for large-scale services. *ACM SIGOPS Operating Systems Review 46*, 1 (2012), 33–39.

[120] Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[121] Skeen, D. Nonblocking commit protocols. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data* (1981), ACM, pp. 133–142.

[122] Skeen, D., and Stonebraker, M. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, 3 (1983), 219–228.

[123] Sousa, J., and Bessani, A. From byzantine consensus to bft state machine replication: A latency-optimal transformation. In *Dependable Computing Conference (EDCC), 2012 Ninth European* (2012), IEEE, pp. 37–48.

[124] Stadler, M. Publicly verifiable secret sharing. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1996), Springer, pp. 190–199.

[125] Stathakopoulous, C., and Cachin, C. Threshold signatures for blockchain systems. *Swiss Federal Institute of Technology* (2017).

[126] Swanson, T. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. *Report, available online, Apr* (2015).

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

[128] Syta, E., Tamas, I., Visher, D., Wolinsky, D. I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., and Ford, B. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on* (2016), Ieee, pp. 526–545.

[129] Technology, S. https://symbiont.io/technology/.

[130] Thomas, S., and Schwartz, E. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf* (2015).

[131] Van Renesse, R., Schiper, N., and Schneider, F. B. Vive la différence:

Paxos vs. viewstamped replication vs. zab. *IEEE Transactions on Dependable and Secure Computing 12*, 4 (2015), 472–484.

[132] Veronese, G. S., Correia, M., Bessani, A. N., and Lung, L. C. Ebawa: Efficient byzantine agreement for wide-area networks. In *2010 IEEE 12th International Symposium on High Assurance Systems Engineering* (2010), IEEE, pp. 10–19.

[133] Veronese, G. S., Correia, M., Bessani, A. N., Lung, L. C., and Verissimo, P. Efficient byzantine fault-tolerance. *IEEE Transactions on Computers 62*, 1 (2011), 16–30.

[134] Vogels, W. Eventually consistent. *Communications of the ACM 52*, 1 (2009), 40–44.

[135] Wang, G., Shi, Z., Nixon, M., and Han, S. Chainsplitter: Towards blockchain-based industrial iot architecture for supporting hierarchical storage. In *2019 IEEE International Conference on Blockchain* (2019), IEEE.

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

[137] Wang, J., and Wang, H. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)* (2019), pp. 95–112.

[138] Wang, X., Zha, X., Ni, W., Liu, R. P., Guo, Y. J., Niu, X., and Zheng, K. Survey on blockchain for internet of things. *Computer Communications* (2019).

[139] Wood, G., et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper 151* (2014), 1–32.

[140] Yin, M., Malkhi, D., Reiterand, M., Gueta, G. G., and Abraham, I. Hotstuff: Bft consensus with linearity and responsiveness. In *38th ACM symposium on Principles of Distributed Computing (PODC'19)* (2019).

[141] Zamani, M., Movahedi, M., and Raykova, M. Rapidchain: scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 931–948.

[142] Zheng, Q., and Xu, S. Secure and efficient proof of storage with deduplication. In *Proceedings of the second ACM conference on Data and Application Security and Privacy* (2012), ACM, pp. 1–12.

[143] Zhou, L., Schneider, F. B., and Van Renesse, R. Apss: Proactive secret sharing in asynchronous systems. *ACM transactions on information and system security (TISSEC) 8*, 3 (2005), 259–286.