OmniLedger: A Secure, Scale-Out, Decentralized Ledger

Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford
firstname.lastname@epfl.ch

École polytechnique fédérale de Lausanne, Switzerland

Abstract

Designing a secure and open Distributed Ledger (DL) system that performs on par with classic payment-service providers such as Visa is a challenging task. Current proposals either do not scale-out or do so only at the cost of security or decentralization. OmniLedger is a new scalable DL that provides secure, decentralized, horizontal scaling by splitting the state into multiple shards and using distributed randomness to assign validators securely. To maintain intra- and cross-shard consistency validators run a novel parallel consensus algorithm for the former and an Atomic Commit protocol for the latter. To mitigate storage cost and enable fast, secure bootstrapping, OmniLedger introduces compact state-blocks that summarize shard-states.

OmniLedger offers tunable performance based on the assumed strength of the adversaries, and scales linearly with the number of shards. Experiments show that it achieves Visa-level throughput of 6000 transactions per second (peaking at 50000) for 1800 validators, of which up to 12.5% (5%) are assumed to be malicious. Finally, OmniLedger significantly reduces bandwidth cost for out-of-date validators to update: for a one-month-old view, a validator downloads 40% of the amount of data compared to Bitcoin, whereas a new validator downloads only 7% while bootstrapping.

1. Introduction

The current increase of interest in decentralized systems has revealed the lack of users' trust in third-parties, such as banks for managing their assets [43], identity providers for managing their digital identities [2, 36], or certificate authorities [54] for certifying their trustworthiness. However, Bitcoin, the most prominent such systems, has been proven both inefficient [17] and insecure [3, 25, 30].

Existing solutions [1, 15, 19, 37, 46] change the consensus algorithm to PBFT [16] variants, which eliminates or mitigates the majority of attacks and can increase scalability [37]. However, as the number of validators increases the performance decreases due to the need to replicate all transactions and reach consensus across larger groups, *i.e.*, these systems do not scale out. A naive approach to scaling out, by

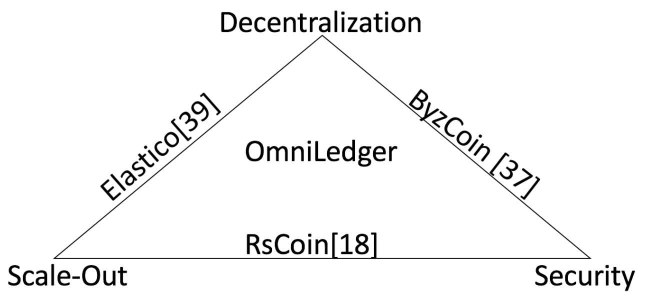

Figure 1. Trade-off triangle of current DL-systems

having one validator for each transaction, does not work in realistic environments where validators may be malicious.

To tackle these horizontal scaling issues, recent efforts [18, 39] split the state and bring sharding [61] to DLs, but do not yet offer a general solution that maintains strong security. RSCoin [18] relies on a central bank for randomness generation and auditing, making it inapplicable to open, decentralized trust environments. Elastico [39] offers horizontal scaling but with high failure probabilities: *e.g.*, 97% within an hour for 16 shards in its experimental configuration, as explained in Section 9. Furthermore, Elastico cannot atomically process transactions that touch more than one shard, a critical requirement in financial applications such as cryptocurrencies. In short, prior solutions [18, 37, 39] achieve only two out of the three desired properties decentralization, security, and scale-out, as illustrated in Figure 1.

This work introduces OmniLedger, the first general DL system that provides all three properties mentioned above and achieves performance comparable to Visa [7, 57]. OmniLedger addresses four key challenges.

First, current scalable systems [37, 39] suffer latencies of minutes, especially under faults. To address this challenge, OmniLedger increases the concurrecny of current approaches, using the observation that unlike Bitcoin's probabilistic consensus [43], PBFT ensures finality [58]: once consensus is reached on a block, validators subsequently accept only transactions that recognize this decision. Hence, a strict total ordering of blocks is not needed, as long as the partial ordering of transactions is consistent. Furthermore, OmniLedger increases robustness of ByzCoin without affecting performance significantly by adopting group communication patterns that enable the consensus leader to circumvent malicious group leaders.

Second, only Proof-of-Work (PoW) [4] systems currently provide a decentralized mechanism to assign validators to shards [39]. However, PoW is inefficient [20] and prone to recentralization (*e.g.*, only a couple of validators can control Bitcoin [33]), necessitating the pessimistic assumption that validators controlling a high fraction of the system's hashing power may be colluding. OmniLedger enables secure sharding based on bias-resistant decentralized randomness [26, 53], which is applicable to any Sybil-attack protection mechanism including PoW [37, 39, 43, 63], Proof-of-Stake (PoS) [5, 35], Proof-of-Personhood (PoP) [12], or permissioned blockchains [15, 18]. This flexibility enables OmniLedger to offer higher performance and scalability in deployments that can realistically assume weaker adversaries.

Third, current sharding approaches in DLs do not provide transaction atomicity [39], posing the risk of indefinitely locking funds if a transaction is accepted only partially. OmniLedger implements an Atomic Commit (AC) [60] algorithm to guarantee either that shards fully commit transactions, or that clients eventually hold proofs of rejection to reclaim locked funds. Given that shards use BFT consensus, we can treat them as honest, reliable processors and implement AC atop these secure shards.

Finally, while Bitcoin's blockchain is relatively small and slow-growing, more scalable systems [1, 37, 39] can drastically increase their DL's growth rate . Eventually, this would lead to recentralization effects as only validators with access to enough resources would be able to bootstrap and store the blockchain. To mitigate these problems, we introduce state blocks that checkpoint [16] the state of a shard enabling blockchain truncation. Thanks to the properties of BFT consensus, validators can drop most of the history, once they commit to a new state block (SB) which removes the problem of an ever-increasing DL. Thus validators can bootstrap and update much faster than in conventional DL-systems and without relying on trusted intermediaries.

To evaluate OmniLedger, we implemented a prototype in Go on commodity servers (12-core VMs on Deterlab). Our experimental results show that OmniLedger scales linearly in the number of validators yielding a throughput of 6,000 transactions per second with a 10-second consensus latency (for 1800 widely-distributed hosts, of which 12.5% are malicious). If we assume at most 5% of validators to be malicious, the system achieves throughput comparable to Visa [7, 57], with a latency of a few seconds. Furthermore, running multiple consensus instances enables OmniLedger's consensus algorithm to handle 20% more throughput than ByzCoin with a concurrent 35% decrease in the consensus latency. Finally, a Bitcoin validator with a month-long stale view of the state incurs 40% of the bandwidth, due to checkpointing with state blocks.

In summary, this paper makes the following key contributions:

- We introduce OmniLedger, the first DL system that provides secure sharding, decentralization, high performance, horizontal scaling and low storage costs, independently of the anti-Sybil-attack method.
- We provide an example of OmniLedger with auditing in an open cryptocurrency, where we introduce the first DL that enables trustless collaboration of a third-parties and an open cryptocurrency.
- We informally analyze the security of OmniLedger and provide guidelines on the trade-off between performance, decentralization and robustness.
- We implement OmniLedger and evaluate a prototype on real-world Bitcoin transactions.

Roadmap. Section 2, recalls existing systems serving as OmniLedger's starting point. Section 3, introduces OmniLedger's systems and threat models and introduces SimpleLedger, a strawman protocol trying to achieve OmniLedger's goals. Then it discusses SimpleLedger's shortcomings and provides a roadmap on how OmniLedger addresses them. Section 4 presents the general architecture of OmniLedger and introduces an example application on an open cryptocurrency. Then, Section 5 provides an informal security analysis, Section 6 describes the implementation of OmniLedger and Section 7 the experimental evaluation. Finally, Section 8 discusses limitations of the current design, Section 9 presents the related work, and Section 10 concludes this paper.

2. Background

This section outlines the most relevant prior work that OmniLedger builds on and how we extend it.

2.1 RandHound

RandHound [53] provides unbiasable decentralized randomness in a Byzantine setting. It assumes the existence of an externally accountable client that wants to get provable randomness from a big group of semi-trustworthy servers. To produce randomness, RandHound splits the group of servers into smaller ones and creates a publicly verifiable commit-then-reveal protocol [52] which leverages the pigeonhole principle to prove that the final random number includes the contribution of at least one honest participant thus perfectly randomizing RandHound's output.

OmniLedger combines RandHound with a random leader election process. This process uses Verifiable Random Functions (VRF [42]) in order to bound the bias probability, even without the existence of an accountable client.

2.2 Cothority

Multisignature schemes [51, 45] are a common approach for attesting files or statements by a joint group of independent parties and, usually, enable the parties to produce a single attesting signature. Such schemes, however, often do not scale well with the increasing number of parties. CoSi [54] is a protocol for large-scale collective signing. Aggregation techniques and communication trees [56] enable CoSi to efficiently produce compact Schnorr multi-signatures [51] and to scale to thousands of participants. The system assumes the existence of an authority (*e.g.* a CA) whose actions are observed and cross-checked and, if found to be valid, co-signed by a group of semi-trustworthy witnesses. Together authority and witnesses form a collective authority or *cothority*.

In OmniLedger we use this scheme for efficient auditing of untrusted shards managed by a third part (*e.g.*, a bank). The auditors form a cothority together with the third party as the leader and verify and attest the consistency of the shard's publicly exposed DL.

2.3 ByzCoin

The first experimentally proven scalable BFT consensus protocol was introduced in ByzCoin [37], where the authors use collective signing [54] to build a scalable version of PBFT [16]. The biggest performance improvement comes from using the aggregation techniques and the communication patterns that CoSi introduced. Another contribution of ByzCoin is the way it bootstrapped BFT consensus groups from a PoW blockchain, thus managing to bring together the open-membership assumptions of Bitcoin with the closed-membership needs of PBFT. Later work [1, 46] provides formal proofs and incentive analysis of systems closely related to ByzCoin.

OmniLedger improves on ByzCoin by increasing the concurrency of block commitments and by using more conservative communication patterns that withstand faults without impacting performance heavily.

3. System Overview

This section presents system and threat models as well as the design goals of OmniLedger. Based on that we build a strawman protocol to identify the challenges that need to be addressed and specify a design roadmap towards OmniLedger.

3.1 System and Threat Model

We assume that the system evolves in *epochs*, coarse-grained units of time such as days or weeks. *Validators* are nodes who reach consensus on the validity of transactions and keep the ledger consistent. Moreover, we presume the existence of a (decentralized) *public-key infrastructure (PKI)* [37, 39], which allows validators to learn each others' public keys and subsequently verify messages sent to each other. We assume the group of validators is determined by a Sybil-attack resistant mechanism, such as PoW, PoS, PoP or simply a static permissioned group [15]. Validators can be anonymous, but they establish their identity depending on the used anti-Sybil-attack method. During an epoch the group of validators is randomly split into subgroups, so-called *shards*, each of which validates a designated portion of the system's state.

If OmniLedger's membership DL does not rely on PoW, the network model is partially synchronous, while for PoW systems, we assume that at least 51% of honest miners are in synchrony with the acceptable delay Δ defined by the expected block mining time as defined by Pass et al. [47]. This differentiation in the network assumptions is necessary for the security of PoW, but, as we do not rely on synchrony within one epoch, does not affect the rest of the paper.

We assume that adversaries are bound computationally and that the usual cryptographic hardness assumptions hold, *i.e.*, adversaries cannot break security of cryptographic hash functions, are subject to the computational Diffie-Hellman assumption, etc. Finally, we assume, similar to prior work [19, 37, 39], that an adversary is able to control at most 25% of the validators. This assumption is strong, as an adversary would have to control 25% of the total currency (PoS) or 25% of accountable validators (permissioned, PoP). In Section 7, we show that decreasing the power of the adversary to $(25 - \epsilon)\%$ for $\epsilon > 0$, can lead to an increase of OmniLedger's performance.

3.2 Design Goals

This section summarizes the design goals of OmniLedger categorized into objectives for performance and security.

Performance Goals:

- High throughput and low latency: The system provides higher throughput and lower latency than other scalable, state-of-the-art DL systems (*i.e.*, ByzCoin).
- Scale-out: If the number of validators increases, the expected throughput increases.
- Low storage: Validators do not need to store the full historical data but only the last state to validate transactions.

Security Goals:

- Full decentralization: There are no trusted third parties or single points of failures.
- Secure cross-shard transactions: Transactions that touch multiple shards commit atomically, or eventually abort, thus preventing funds from being locked forever.
- Secure sharding: Each shard correctly processes transactions assigned to it.
- Censorship resistance: Individual malicious shard leaders cannot censor transactions.

3.3 OmniLedger Design Roadmap

This section introduces SimpleLedger, a strawman DL-system, on the basis of which we specify our design roadmap towards OmniLedger.

The PKI mentioned in Section 3.1 is specified in SimpleLedger (and also later in OmniLedger) as an epoch-

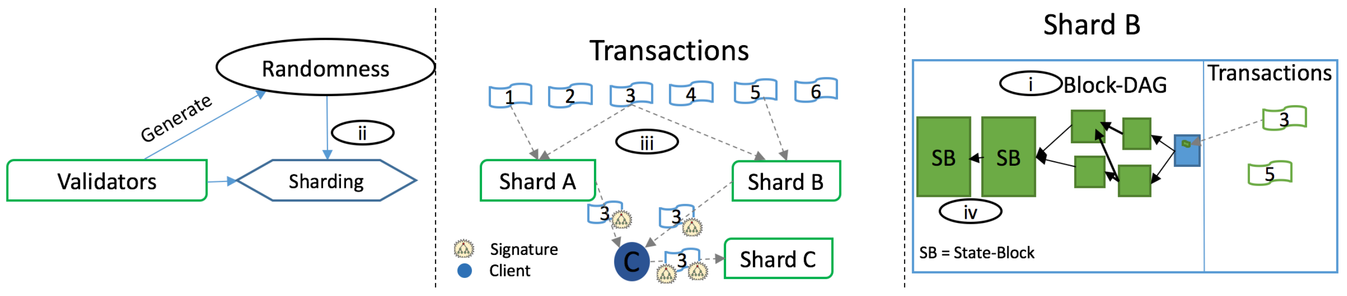

Figure 2. Architectural overview of OmniLedger

level DL, where membership is either determined through a CA [15] or through validators competing for election [37, 39] using their anti-Sybil-attack membership tokens (PoW, PoS, PoP) as proofs for having established their identities. All elected members can participate in the next epoch and gain rewards. Validators learn the epoch's public keys via gossiping mechanisms similar to Bitcoin's approach and use these keys to verify signatures. For the rest of the paper we assume that validators have established their identities and focus only on one epoch.

For the per epoch run of SimpleLedger, we start with the secure validator assignment to shards. Permitting the validators to choose the shards they want to participate is not secure, as the adversary could focus all his identities in one shard and break it. As a result, we need to use some source of randomness to ensure that the validators of one shard will be a sample of the overall system and w.h.p. will have about the same fraction of malicious nodes as the overall system. PoW is currently the only proven Sybil-attack resistant method that can provide such decentralized randomness. However, we want SimpleLedger to be applicable to any DL. Hence, SimpleLedger uses a centralized randomness beacon (*e.g.*, the NIST beacon [44]) to split the validators into subgroups.

After assigning validators to shards, SimpleLedger uses ByzCoin for inner-shard consensus. To ensure secure operations, ByzCoin is run in groups of 600, see Figure 6. Finally, SimpleLedger assumes that transactions are validated within one shard and that the DL of each shard holds all the historical data, for new validators to bootstrap securely.

Challenges. Although SimpleLedger is a first step towards our goals it still exhibits various restrictions. Namely, (1) ByzCoin's algorithm enforces a global total ordering of transactions leading to non-real-time latencies, especially under faults, (2) the randomness source is centralized, (3) there is no secure cross-shard transaction mechanism and (4) validators face high storage overheads. To address these restrictions, we transform SimpleLedger into OmniLedger in four steps (Figure 2):

1. To increase throughput and decrease latency, OmniLedger removes total ordering between blocks and introduces parallel consensus. (Section 4.2, ⓘ in Figure 2).
2. To enable sharding in any kind of DL, OmniLedger extends RandHound [53] and creates decentralized, bias-resistant randomness, without a trustworthy client (Section 4.3, ⓘⓘ in Figure 2).
3. To enable secure cross-shard transactions OmniLedger uses AC that provides clients with proofs of rejection, and enables them to abort partially committed transactions (Section 4.4, ⓘⓘⓘ in Figure 2).
4. To decrease the data stored on the ledger, OmniLedger introduces state blocks that summarize the current state and enable history-pruning. (Section 4.5, ⓘⓥ in Figure 2).

The first and fourth technique can increase performance of current DLs [1, 15, 19, 37, 43, 46, 62] that do not implement sharding.

4. OmniLedger's Design

This section presents OmniLedger, a framework that provides security, scalability, and performance for decentralized systems that maintain a distributed ledger (DL) and run in a Byzantine environment. For clarity of exposition, we start from SimpleLedger, as introduced in Section 3, and address its shortcomings step by step until we arrive at OmniLedger.

4.1 RoadMap

Section 4.2 shows how to increase performance of ByzCoin and PBFT by identifying the dependencies between conflicting transactions, removing total ordering of blocks and introducing a block-DAG to achieve maximum concurrency. In the next part, Section 4.3, we introduce bias-resistant, decentralized randomness to securely assign validators to shards, followed by the specification of an Atomic Commit protocol to preserve transaction atomicity in Section 4.4. Afterwards, Section 4.5 discusses how to deal with the problem of an ever-growing DL by introducing intermediate state blocks that enable efficient verification and fast bootstrapping and decrease storage needs. Finally, Section 4.6 describes an anti-censorship mechanism, the last component of OmniLedger, and Section 4.7 extends the default permissionless setting of OmniLedger to enable a trustless coexistence of open and private shards, the latter of which are managed by third-parties (*e.g.*, banks).

4.2 Speeding Up Consensus

This section provides a new *parallel consensus* algorithm that can be used on its own in a cryptocurrency and as experiments show (see Section 7), it concurrently provides higher throughput and lower latency than ByzCoin.

4.2.1 Challenges

A number of DL protocols [24, 37, 39, 43, 62], including SimpleLedger, have a common architecture, where a leader proposes a new block that respects a strict total ordering of blocks, that form a blockchain. This block is a batch of operations on the objects of the state, that are the Unspent Transaction Outputs (UTXOs). In this work we intuit that we can achieve better parallelization if the validators decide on the ordering of operations per UTXO-object, thus enabling them to commit transactions in parallel. However, in order to run parallel consensus instances, we need to be sure that conflicting transactions are not committed concurrently. Hence, in order to keep the state consistent, independent of the ordering upon which blocks were committed, we need to identify the dependencies between transactions.

Identifying the dependencies is not enough, as running consensus is costly and consequently does not scale well if run on a per transaction basis [37]. As a result, the system needs to rely on batching non-conflicting sets of transactions, *i.e.*, creating blocks. However, two problems arise with the current blockchain [43] data-structure. First, every block is dependent on its parent, thus creating a linear chain which means that the shard cannot run consensus rounds in parallel. Second, it is almost certain that most transactions within one block will depend on different transactions that were committed in multiple different blocks, creating a multitude of backpointers to capture all dependencies.

4.2.2 Identifying the Dependencies

To address the challenge of safely running parallel consensus instances we first identify the potential dependencies between transactions. Bitcoin transactions have two kinds of operations: inputs and outputs. At the input fields, the clients spend their UTXOs. At the output fields, the clients create UTXOs and credit them some value. A conflict between transactions exists only if they have overlapping sets of inputs and/or outputs, hence the validators can safely commit in parallel any other pair of transactions without imposing order. We now look into the specific dependencies between two transactions that update overlapping parts of the state. We call the two transactions A and B, where A should be committed before B.

- **Input-after-input**: This dependency is an unavoidable data dependency, as B needs to wait until A commits. By breaking consistency and forcing the concurrent commitment of two input-conflicting transactions, attackers mount double-spending attacks [43] in slowly consistent protocols [24, 43, 62].
- **Input-after-output**: This dependency emerges when A creates a UTXO as an output and B concurrently spends that specific UTXO. Consequently, if B is validated before A, the validation fails. Hence, B should be in a block that causally depends on the block that validated A.
- **Output-after-output**: When it comes to concurrently committing transactions that try to credit the same address (account), there is no real dependency due to the commutativity of addition.[1] This is also why transactions A and B create different UTXOs for the same output.

4.2.3 Block-DAG (Directed Acyclic Graph)

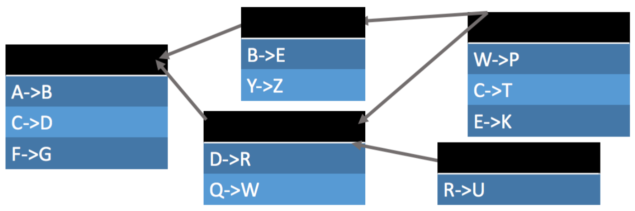

Figure 3. Block-DAG in OmniLedger ($A- > B$ denotes that funds are transfered from account A to account B)

To address the challenge of putting transactions into batches while preserving concurrency, we move to the block-DAG data-structure (see Figure 3), where every block has multiple backpointers. Unlike in previous research [38], we enforce pending blocks to include non-conflicting transactions and should point to all other blocks upon which the pending blocks depend.

Finally, as for the challenge of needing a big amount of backpointers per block, we relax the requirement that a block (we call it block A) must point back to all the committed blocks that A's transactions depend on. Instead, A refers to the minimal set of committed blocks that are independent of each other (because they were created concurrently) but that A depends on.

4.3 Secure Shard Creation

A crucial aspect of the secure operation of OmniLedger is the proper assignment of validators to shards. The strawman approach, introduced in Section 3, uses a centralized randomness beacon, thus bearing the risk that if an adversary manages to subvert the beacon he can also undermine OmniLedger's security. To mitigate this problem, OmniLedger uses a verifiable randomness-generation protocol [11, 32, 53] that does not rely on a trusted third party. To keep OmniLedger's randomized validator assignment independent from other blockchain-based systems, such as Bitcoin [11], we make use of the RandHound protocol [53]. Next, we describe the challenges of integrating such a beacon into OmniLedger and discuss our solution.

[1] Transactions add a defined amount of funds, which is commutative.

4.3.1 Challenges

RandHound requires a leader who manages protocol runs. Obviously, this leader might fail accidentally or deliberately at his task of producing collective randomness for the sharding process. To mitigate this problem we specify a mechanism to elect leaders who take over responsibilities from their failing predecessors. Making this election mechanism deterministic, however, might enable a Byzantine adversary to enforce the sequential failing of $n/3$ out of n protocol runs, hence enabling him to either significantly delay or in the worst case even bias the sharding. Therefore, the election of leaders should be unpredictable and unbiasable, which results in a chicken-and-egg problem, as the purpose of a RandHound protocol run is to generate randomness.

4.3.2 Randomized Leader-Election Protocol

To address the challenge of delaying or biasing the randomized sharding process, we use two countermeasures. First, OmniLedger enforces the rule that if a leader fails to finish a RandHound run, he loses his right to participate in the epoch and the associated reward. Second, OmniLedger uses a lottery mechanism for leader election; it is based on verifiable random functions [42], reducing an adversary's chance to be the leader in n consecutive runs to $(1/3)^n$.

There are multiple alternative ways to get a VRF [22, 23]. In OmniLedger we use the construction of Franklin and Zhang [26]: Let $(k_i, K_i) = (k_i, G^{k_i})$ denote the private-public key pair of node i where G is the generator of the cryptographic group. We assume that K_i is known to other OmniLedger participants, that a protocol instance is identified by a unique value u (*e.g.*, the hash of the membership DL signed by the previous epoch's members), and that views v are consecutively numbered starting from 0. Then, for each view, participants compute a view base $H = \mathsf{H}(u, v)$ for a hash function H such that $H \neq G$ is a generator of the respective cryptographic group. Then, each participant i computes his lottery ticket H^{k_i} together with a discrete-logarithm equality proof [26] that is verifiable against the public key K_i.

Participants then start gossiping with each other to find the lottery ticket with the lowest value. After waiting a sufficient amount of time for this gossip to propagate the lowest-available solution to all good participants, each participant "locks in" the lowest valid ticket he has seen and takes the holder of that solution to be the leader, subsequently ignoring messages from any other node purporting to be the leader for that view. To prevent replay attacks, all messages coming from the elected leader in this view need to be tagged by a unique identifier derived from the values u, v, and i.

Each participant, including malicious ones, can produce only one single valid lottery ticket per round. This ticket is unpredictable to other non-colluding nodes, as they do not know the solution-holder's private key. However, a malicious holder of the lowest solution in a view can still cause mischief by either (a) withholding their solution entirely, or (b) withholding their solution until close to the time the honest participants will lock in their choice of leader.

Nevertheless, both failure cases can only occur if a malicious node draws the winning lottery solution in a given view, thus resulting in a success probability that is upper-bounded by $(1/3)^n$ for n successive malicious views. Furthermore, the aforementioned attacks are not-stronger equivalents to the malicious leader executing the lottery protocol correctly but then doing nothing as leader of that view. In this case, the honest participants will detect the DoS and change the view. Hence, in the face of any such attacks, a good (or at least live) leader will be picked in a small, constant-expected number of rounds.

Shard Size Security. One of the goals of OmniLedger is to provide tunable performance based on the strength of the adversary. Let m be the percentage of validators the adversary controls drawn from an infinite pool of validators and N be the size of one shard. Table 4.3.2 provides guidelines on how OmniLedger should be configured based on the adversarial strength to achieve a failure probability of 10^{-6} with a more detailed analysis being presented later in Section 5.1.

m	1%	5%	12.5%	25%
N	4	25	70	600

Table 1. Adversarial power m dependent shard sizes N to achieve a failure probability of at most 10^{-6}

4.4 Coordinating Cross-Shard Transactions

SimpleLedger assumes that transactions can be fully validated inside a single shard; We start by proving that cross-shard transactions for sharded DLs are expected to be the norm, given that previous work [39] ignored this issue. Afterwards, we address the challenge of guaranteeing transaction atomicity by introducing the Atomic Commit protocol that is used in OmniLedger and reasoning about the protocol's validity and security.

4.4.1 Probability of Cross-Shard Transactions

This section explores the limitations cross-shard transactions pose to the performance of the system. When splitting the state into disjoint parts, the common practice [18, 39] is to assign UTXOs to shards, based on the first bits of their hash. For example, one shard manages all UTXOs whose first bit is 0, and the second shard all UTXOs whose first bit is 1. Then each shard is managed by a group of validators who keep the state consistent and commit updates.

For an intrashard transaction we want all the inputs and outputs of the transaction to be assigned at the shame shard. The probability of assigning a UTXO in a shard is uniformly random from the randomness guarantees of cryptographic hash functions. Let m be the total number of shards, n the sum of input and output UTXOs and k the number of shards that need to participate in the cross-shard validation of the transaction. The probability can be calculated as:

$$P(n,k,m) = \begin{cases} 1, n=1, k=1 \\ 0, n=1, k\neq 1 \\ \frac{m-k}{m}P(n-1,k-1,m)+ \\ \frac{k}{m}P(n-1,k,m), n\neq 1, k>0 \end{cases} \quad (1)$$

For a typical transaction with two inputs and one output and a three-shard setup, the probability of a transaction being intra-shard is $P(3,1,3) = 3.7\%$, rendering the assumption that transactions touch only one shard [39] problematic.

4.4.2 Secure Validation of Cross-Shard Transactions

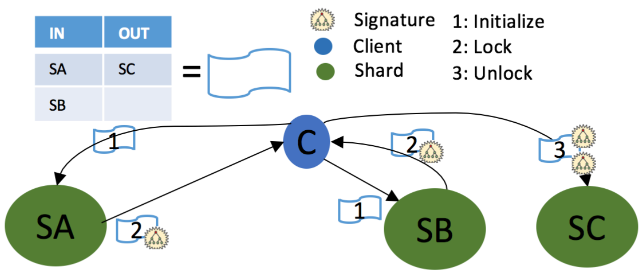

Figure 4. Atomic Commit protocol in OmniLedger. Arrows indicate flow of information, not direct messages

An important issue in order to enable every user to transact with anyone securely and instantly is preserving atomicity of cross-shard transactions that touch more than one shard. Otherwise money can be locked forever in one shard because another input of the transaction is invalid.

We design an Atomic Commit protocol that assumes that shards run BFT-consensus inside. This protocol works only if the shards participating are not compromised (*i.e.*, less than 1/3 of the validators in each shard are malicious). The protocol works as shown in Figure 4:

Initialize: The client creates a transaction whose inputs touch the state of some *Input Shards* (IS) and whose outputs touch the state of some *Output Shards* (OS). The transaction is gossiped and eventually reaches the ISs.

Lock: Each IS leader validates the transaction inside the shard deciding that the inputs can be spent and returns a proof-of-acceptance. If the transaction is not accepted, the leader creates analogously a proof-of-rejection (a special bit indicates acceptance or rejection). The client can follow the DLs of each IS to verify the proofs; once all ISs have decided, the client holds enough proofs to show that the transaction is committed or to reclaim any locked funds.

Unlock: This process occurs at the OSs for accept and at the ISs for abort. After the Lock phase, the client uses the proofs to create an unlock transaction, that has a special format as the inputs are spendable by verifying the proofs. The transaction reaches the ISs and OSs. If the transaction is accepted, the OSs' leaders add the transaction in the next block, to update the state and enable the expenditure of the new funds. If the transaction is aborted, then the ISs' leaders follow the same procedure to unlock the UTXOs.

Discussion. Assuming that shards cannot fail, the above algorithm guarantees that if all shards propose to accept the transaction, then the transaction is eventually committed, as each shard will eventually have an honest leader who reaches consensus with all honest validators. A malicious client can create a valid proof-of-acceptance (for OSes) if and only if he receives proofs-of-acceptance from the Lock phases of all the ISs. Similarly the malicious client can create a valid proof-of-rejection (for ISes) if and only if he receives a proof-of-rejection from the Lock phase of one of the ISs. Given that the ISs are honest, the client cannot hold both proofs concurrently.

Clients experience similar latency as intrashard transactions, as they wait only for the Lock phase to finish. However, they pay transaction fees proportionally to the number of shards the transaction touched and pay for rejected transactions as well. The difference between classic AC and this protocol, is that shards cannot crash and remain silent indefinitely. Moreover, clients are responsible for proceeding with the Unlock phase, as it is their funds at stake. A client who crashes permanently only harms himself and is isomorphic to a client who crashes and loses his private key, leading to his inability to spend the corresponding to the key UTXOs.

If OmniLedger's techniques are extended to implement scalable state-machine replication, then protection from malicious clients that do not unlock is needed. In this case, one shard can be the coordinator of the transaction and is the one that initiates the Unlock. A further refinement would be to run (meta-)consensus on top of the participating ISs and OSs, and only afterwards, to send the proofs to the clients.

4.5 Pruning the Ledger With State-Blocks

DL protocolss [24, 37, 39, 43, 62], including SimpleLedger, follow the same pattern of storing all historical data from the first day of operation. Bitcoin's blockchain is more than 120 GB [9] and increases with an average of 1 MB every 10 minutes. Given the high throughput that next generation systems provide [37, 39], the expected growth rate can explode. On the other hand, the state size of Bitcoin is around 1.7 GB [50] and increases around 500 MB per year.

To resolve the challenge of storing the full DL, OmniLedger introduces state blocks (SB). SBs play a similar role to stable checkpoints of PBFT [16]. An important difference between SBs and checkpoints is how those are stored. SBs should not be forgotten, as this defeats the purpose of having a DL with an authenticated immutable history. Instead validators create a conceptually higher-level DL (storing these SBs), that provides skips from an epoch's SB to another. This SB-DL holds the last SB in full and all previous SBs' headers. This is necessary as clients that want to prove the validity of a past transaction need to have a reference point.

For example, a client who holds a Proof-of-Existence (PoE) [59] cannot use that proof as it consists of showing that a transaction was included in a normal block (bearing a timestamp). Even if the transaction is summarized in the latest SB, this SB will have a later timestamp, meaning that the client cannot prove the time the transaction was first created.

At the end of the epoch, the leader starts the creation of an SB by reconstructing the shard's state that consists of all UTXOs and creates an ordered Merkle Tree [41]. Then the shard's validators run consensus on the SB's header (which includes the Merkle Tree Root), while no regular blocks are pending. Once they reach consensus that the SB is correct, the new block becomes the *epoch genesis block*.

Meanwhile, validators store the headers of all the SBs that form a SB-DL (see Figure 2) and drop normal blocks. This way OmniLedger delegates the responsibility of proving past transactions to the client who holds the proofs. These proofs consist of all the normal block headers, starting from the block that validated their transaction until the subsequent SB. In this system the PoE client can skip backwards the SB-DL blockchain, find the SB he has a reference to and then use his PoE. On the contrary, proving that a transaction is still valid is more efficient with SBs, as any client can use the SB as proof, instead of the normal block; and a verifier will need to only check the last part of the DL, instead of checking all the DL from the validation of the transaction.

Finally, bootstrapping new validators or bringing crashed validators up-to-date becomes more efficient, as they start from the last valid SB and replay only the last part of the DL, instead of replaying the full DL from the first block or from the moment they crashed. For example, if Bitcoin is deployed on OmniLedger the bandwidth bootstrap cost would be two orders of magnitude less (1.7 GB instead of 120 GB). This becomes more important when sharding is used and, due to the random shard assignment process, validators switch shards periodically and need to update.

4.6 Censorship Resistance

A final issue that OmniLedger addresses is when a malicious shard leader censors transaction. This attack can be undetectable from the rest of the shard's validators. A leader who does not propose a transaction is acceptable as far as the state is concerned, but this attack can compromise the fairness of the system or be used as a coercion tool. For this reason the rest of the validators can request transactions to be committed, because they think the transactions are censored. They can either collect those transactions via the normal gossiping process or receive a request directly from a client.

The workflow (Figure 5), starts (step 1) with each validator proposing a few hashed transactions for anti-censorship, which initiates a consensus round. Once the round ends, there is a list (step 2) of transactions that are eligible for

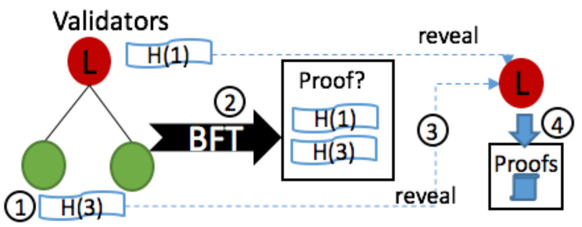

Figure 5. Anti-censorship mechanism OmniLedger

anti-censorship, which is a subset of the proposed. As the transactions are hashed, no other validator knows which ones are proposed before the end of the consensus. Each validators reveals (step 3) his chosen transactions, the rest of the validators check that the transactions are valid and expect the leader to propose them within a fixed (*e.g.* 10) amount of blocks. The leader is then obliged to include (step 4) the transactions that are consistent with the state, otherwise the honest validators will cause a view-change [16].

4.7 OmniLedger on a Hybrid Cryptocurrency

In the specific case of an open cryptocurrency, OmniLedger enables hybrid systems, where third-parties (*e.g.* banks using RsCoin [18]) establish untrusted (private) shards. OmniLedger enables users to move coins to and from private shards, as they please.

private shards are validated by assigned validators whom OmniLedger considers untrustworthy. However, these shards provide features not available in the generic permissionless DL, such as transaction privacy where the private shard's DL is not transparent. The basic requirement is for the ledger's governance and the third-party to agree on the existence of mutually trusted auditors.

Auditors form a cothority [54] with the third-party as the leader. The cothority or collective authority is a group of semi-trustworthy, accountable witnesses that attest on the validity of the records the leader proposes, or in our case independently check the consistency of each exposed SB at the end of the epoch. The difference between auditors and validators (other than accountability), is that the former do not store history but only replay the exposed DL and need not run consensus, as there is only one question to answer; is the SB valid? Auditors can run the scalable Collective Signinig [54] protocol atop which ByzCoin builts, or attest with individual signatures.

If we assume $\lfloor \frac{N}{3} \rfloor$ of malicious auditors, where N is the number of auditors, once $2 \cdot \lfloor \frac{N}{3} \rfloor + 1$ of auditors accept the SB the rest of the system is certain of the SB's validity. The fundamental goal of this system is to protect normal (open) shards from compromised private shards so that they can keep functioning and providing the same security guarantees as before. The full protection of a client using a private shard is not considered as a client chooses to trust a specific private shard, in order to get an additional feature.

Auditors also detect and mitigate attacks where a compromised shard inflates the currency (Sections 4.7.1 and 4.7.2) or tries to DoS clients (Section 4.7.2).

4.7.1 Shard Poisoning Mitigation

A *shard poisoning* attack occurs when a private shard accepts an invalid cross-shard transaction, whose inputs are within the private shard's state, creating money out of thin air. To mitigate this, we pose a limit in the amount of money that can move out of the private shard. When a third-party establishes a private shard, it posts collateral that serves as a safety net in case of misbehavior.

Afterwards the auditors set a quota of funds that can be transfered outside the shard between two epochs: this equals the amount of collateral posted and is confiscated after detection of an attack. This way the state of the private shard can be recovered, upon detection, without affecting the correctness of the rest of the state. We leave the recovery process of faulty private shards to the discretion of the shard's manager, who can decide which transactions should be deleted in order to get back to a correct state. The quota level is renewed after the auditors accept each state-block, or increased upon request and more collateral posting.

4.7.2 Cross-Shard Atomic Commit

Another challenge for private shards is that the protocol of Section 4.4 cannot work out-of-the-box. The properties we want from such an algorithm that can be violated are:

1. **Agreement:** If an honest shard decides on action A, then every honest shard decides on action A.
2. **Termination:** Every honest shard eventually decides.

Agreement can be violated in the previous algorithm, as a dishonest shard can collude with the client and provide both a proof-of-acceptance and a proof-of-rejection. The client can then cause the ISs and OSs to diverge. Termination can be violated by a DoS from a private shard that never creates the required proofs. To resolve these challenges, we resort to the auditors to provide a trustworthy intermediary between OmniLedger and private shards. The auditors job is to (a) notarize all cross-shard transactions so that they can check the state-block later and (b) make sure that the private shard does not exceed its quota level.

During a cross-shard transaction, auditors verify the proofs provided by the private shards and sign-off[2], before the client and any normal OSs or ISs can feel secure that the transaction is committed (or aborted), preserving agreement. Preserving termination is similar to the censorship resistance algorithm (see Section 4.6) run among auditors and requesting proofs from validators.

[2] 2/3 of auditors needs to sign, in order to prevent an equivocation attack

5. Security Arguments

This section analysis the security of OmniLedger informally.

5.1 Randomness Creation

RandHound assumes an honest leader, however in OmniLedger we cannot guarantee his existence, meaning that a dishonest leader can choose to release the output of RandHound or fail and cause a restart. We economically disincentivize such behavior and bound the bias by the randomized leader election process described in Section 4.3.

The leader election process is unpredictable as the adversary is bound by the usual cryptographic hardness assumptions (see Section 3.1) and, in particular, is unaware of (a) the private keys of the benevolent validators and (b) the input of the hash function. The membership DL is unpredictable at the moment of private key selection and private keys are bound to the identity and cannot change without destroying the identity. As a result, the adversary has $m = 1/3$ probability per round to control the elected leader. Thus, the probability that an adversary leads n consecutive views is upper-bounded by $P\left[X \geq n\right] = \frac{1}{3^n} < 10^{-\lambda}$. If we want $\lambda = 6$, then the adversary will at most control 12 consecutive views. This is an upper bound as we do not include in our calculation the exclusion of the previous leader from the consecutive leader election process.

Shard Size Security. The security of OmniLedger's validator assignment mechanism is modeled as a random sampling problem with two possible outcomes (benevolent, malicious). Assuming an infinite pool of potential validators we can use the binomial distribution (Equation 2). We can assume random sampling due to RandHound's unpredictability property that guarantees that each selection is completely random, leading to the adversarial power to be at most $m = 0.25$, from the assumption of Section 3.

$$P\left[X \leq \lfloor\frac{n}{3}\rfloor\right] = \sum_{k=0}^{n} \binom{n}{k} m^k \left(1-m\right)^{n-k} \quad (2)$$

In this setting, we are interested in the probability that one shard picks less than $c = \lfloor\frac{n}{3}\rfloor$ Byzantine nodes as consensus group members, hence guarantees safety. To calculate the failure rate of one shard, we use the cumulative distributions over the shard size n, where X is the random variable that represents the number of times we pick a Byzantine node. Figure 6 illustrates the proposed shard size based on the power of the adversary.

5.2 Epoch Security

Although the security of a shard can be modeled as a random selection process the failure probability calculated in the previous section is not equal to the failure probability of an epoch. Instead, the total failure rate of a correctly bootstrapped epoch is the union bound of the failure rates of

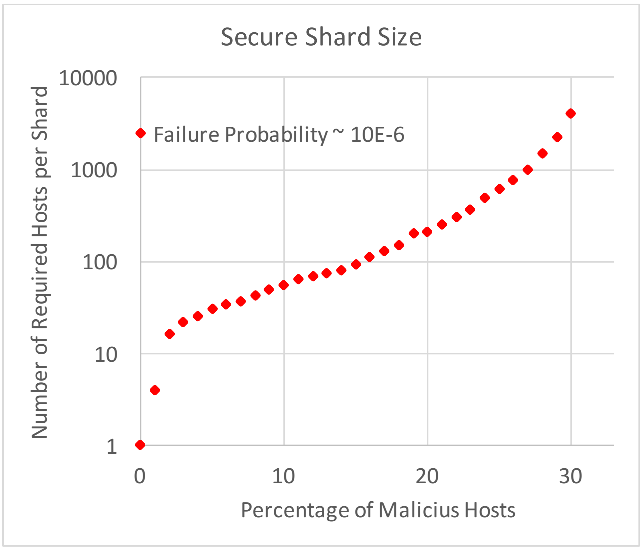

Figure 6. Secure Shard Size

the individual shards. Furthermore, correct bootstrap of an epoch is eventually achieved giving the adversary a bias advantage every time he controls the leader of RandHound.

We can calculate an upper bound of the total failure probability by allowing the adversary to run RandHound multiple times and choose the random number he prefers. This is stronger than what RandHound permits, as the adversary cannot choose an older random number.

An upper bound for the epoch failure event X_E can be calculated as

$$P\left[X_E\right] \leq \sum_{k=0}^{l} \frac{1}{3^k} \cdot n \cdot P\left[X_S\right] \tag{3}$$

where l is the number of consecutive views the adversary controls, n is the number of shards and $P\left[X_S\right]$ is the failure probability of one shard as calculated in Section 5.1. For $l \to \infty$, we get $P\left[X_E\right] \leq \frac{3}{2} \cdot n \cdot P\left[X_S\right]$. If we analyze the experiment of Figure 15, the failure probability for a 12.5%-adversary and 16 shards is $5 \cdot 10^{-5}$ or one failure in 358 years if one epoch corresponds to one week.

5.3 Censorship Resistance

This section analyzes the censorship resistance guarantees of the algorithm described in Section 4.6.

Because transactions are hashed randomly, the Byzantine leader does not know the transactions that are proposed by honest validators. As a result, honest validators will eventually propose transactions with sufficient fees whose delay is abnormally high. The Byzantine leader cannot block all honest proposals as this will cause a view-change. This means that the transaction is going to be proposed for proof. Hence, the failed leader is obliged to commit or the honest validators will switch to another leader.

5.4 Shard Poisoning

Any currency inflation is contained within the failed shard, due to the rate-limit enforced for cross-shard transactions. Since the maximum amount of cross-shard funds per epoch is equal to the sum of the withheld collateral and considering that auditors will detect misbehavior within the epoch, we are sure that a compromised private shard will not affect the rest of the state.

5.5 Atomic Commit

The validity and agreement of the Atomic Commit algorithm follows, as already outlined in Section 4.4, from the fact that normal shards behave as honest processors and private shards have an auditor that enforces compliance. Termination follows from the liveness guarantees of the underlying consensus algorithm.

6. Implementation Details

We implemented OmniLedger in Go [31] and will make it available on GitHub. We extended RandHound's code that is available on GitHub and implemented the randomized leader election by using the VRF construction of Franklin and Zhang [26]. To implement OmniLedger's intra-shard consensus, we modified ByzCoin's code that is available on GitHub by having multiple rounds concurrently running on blocks that include independent transactions.

Furthermore, even if ByzCoin is scalable, it suffers availability risks due to its tree structure. In order to mitigate this, we modify the communication pattern to Groups, as shown in Figure 7, so that there is a leader and $\sqrt{N}$ groups of size $\sqrt{N}$. These groups are created randomly using the output of RandHound. The protocol uses the same communication pattern, but the shard's leader randomly chooses a participant of each group to be the group's leader. If this group's leader does not respond in a predefined timeout, the shard's leader contacts another group member. This way the expected number of retries is bound to a maximum of $\frac{\sqrt{N}}{2}$. As you can see in Section 7, there is a performance penalty for this approach; nevertheless, the system remains highly performant and withstands faults in a clean way.

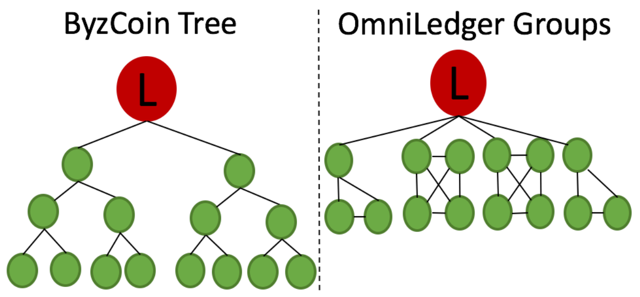

Figure 7. OmniLedger's Group communication pattern vs ByzCoin's tree

Another performance improvement came from pipelining the internal Collective Signing [54] rounds of ByzCoin so that there is one signed block per communication round instead of every four. We implemented the group splitting process; the AC consensus protocol that runs on top of the ByzCoin shards; a client that is dispatching and verifying the proofs and an auditor group that runs AC with private shards and Collective Signing for notarization.

Pipelining ByzCoin Collective Signing [54] is done in four different phases per round, namely Announce phase, Response phase, Challenge phase and Commit phase. In ByzCoin the Announce and Commit phases of the CoSi protocol are not used since the proposed messages (the block in the prepare round and the proof-of-acceptance in the commit round) can be sent to the signers in the Challenge phase. This allows for the intersection of the two rounds so that the announce/commit phases of the ByzCoins's commit round are piggybacked on the challenge and response messages of the ByzCoins's prepare round. This pipeline allows for a reduction of latency (one round-trip less).

Looking into the normal execution of ByzCoin this pipeline can be extended so that an optimal throughput of one signed block per round-trip is produced. A sample execution can be seen in Table 2.

		t_i	t_{i+1}	t_{i+2}	t_{i+3}	t_{i+4}
B_k	prepare	An/Co	Ch/Re			
	commit		An/Co	Ch/Re		
B_{k+1}	prepare		An/Co	Ch/Re		
	commit			An/Co	Ch/Re	
B_{k+2}	prepare			An/Co	Ch/Re	
	commit				An/Co	Ch/Re

Table 2. OmniLedger pipelining for maximum transaction-throughput; B_k denotes the block signed in round k, An/Co the Announce-Commit and Ch/Re the Challenge-Response round-trips of CoSi

7. Evaluation

We ran all our experiments on DeterLab [21] by using 60 physical machines, each having an Intel E5-2420 v2 CPU and 24 GB RAM. We arranged the servers in a star topology and in order to simulate a realistic wide area network we imposed a latency of 100ms per link. Furthermore we limited the bandwidth of each host to be 20Mbps. The blocks include actual Bitcoin transactions parsed from the first $10,000$ blocks of Bitcoin's blockchain.

7.1 Scaling Intra-shard Consensus

In this experiment, we measure the performance benefits of the parallel consensus when extending ByzCoin's consensus algorithm. In order to have a fair comparison, each data-series corresponds to the total size of data concurrently in the network, meaning that if the concurrency level is 2 then

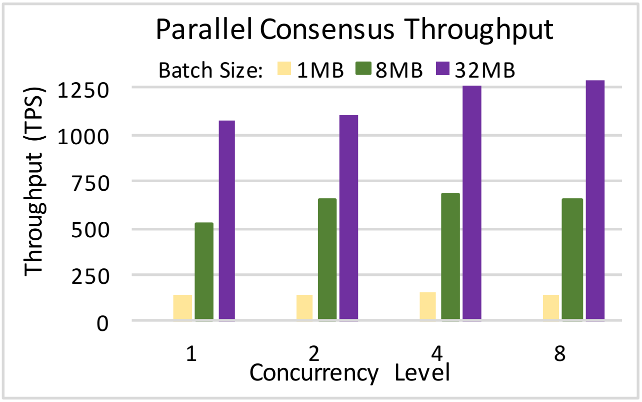

Figure 8. Throughput achieved with parallel consensus

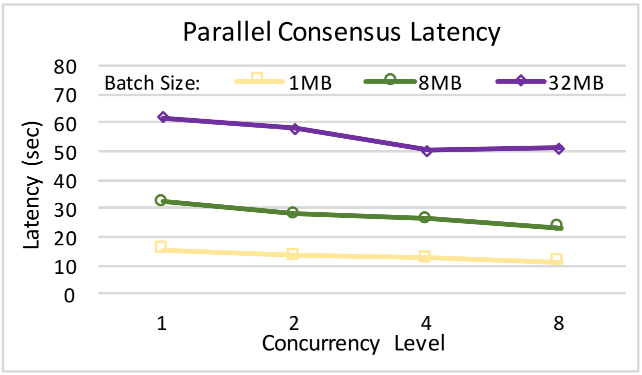

Figure 9. Latency achieved with parallel consensus

there are 2 blocks of 4 MB concurrently, adding to a total of 8 MB.

As depicted in Figures 8 and 9, we can see that there is a 20% performance increase when moving from one big block to 4 smaller concurrently running blocks, with an additional 35% decrease in the per block consensus latency. This can be attributed to the higher resource utilization of the system, when blocks are arriving more frequently for validation. When the concurrency further increases, we can see a slight drop in performance, meaning that the overhead of the parallel consensus outweighs the parallelization benefits.

Figure 10 illustrates the scalability of ByzCoin's [37] tree and fall-back flat topology, versus OmniLedger's more fault-tolerant (Groups) communication pattern and its performance when failures occur. As expected the tree topology scales better, but only after the consensus is run among almost a thousand nodes, which assumes a rather strong adversary (see Figure 6). This is due to OmniLedger's communication pattern which can be seen as a shallow tree where the round-trip from root to leaves is faster than in the tree of ByzCoin.

Hence, whereas ByzCoin has a fixed branching factor and an increasing depth, OmniLedger has a fixed depth and an increasing branching factor. The effect of these two choices

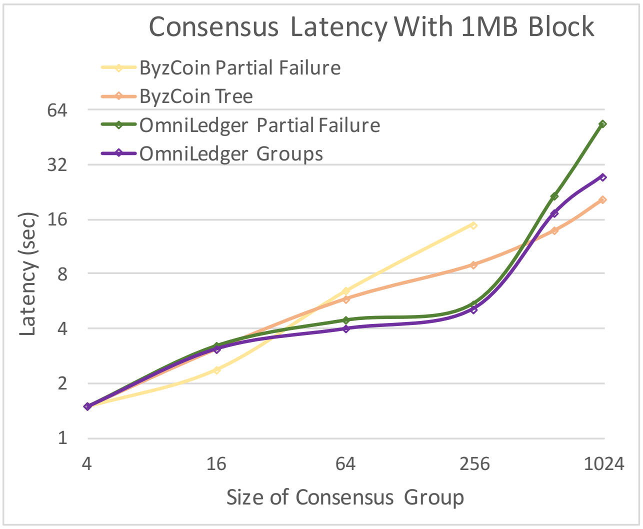

Figure 10. Scalability for different communication patterns

leads to better latencies for a few hundred nodes for fixed depth. However, the importance of the group topology is that it is more fault tolerant and even when failures occur the performance is not seriously affected. This is not true for ByzCoin that switches to a flat topology in case of failure that does not scale after a few hundred nodes, due to the huge branching factor. This experiment was run with 1 MB blocks.

7.2 Epoch-boot Cost

In this experiment, we measure the time to bootstrap a new epoch. The process starts with collectively signing the new membership configuration that is used for the VRF based leader election. Once the leader is elected, he runs RandHound with a group-size of 16 hosts, (which is secure enough for a 25% adversary [53]). When RandHound finishes, each validator verifies the random number and connects to his peers. In this experiment we assume that validators already know the state of the shard they will be validating. It is important to mention that this process is not on the critical path, but occurs concurrently with the previous epoch. Once the new groups have been setup, the new shard-leaders force a view-change on them, like in ByzCoin [37].

As we can see in Figure 11, the cost of bootstrapping is mainly due to the RandHound run that takes up more than 70% of the total time. However, if we combine that with the proof of Section 5.1 we can see that even in the case with 1800 hosts an honest leader will be elected after at most 12 RandHound runs, which will take a bit more than 3 hours. Given an epoch duration of one day, this worst case overhead is acceptable. Another approach would be to run RandHound among only the new members (144 for one day epoch), this would be faster but it is susceptible to PoW burst attacks, that could allow the adversary to bias the shard assignment. In both cases the randomness splits only the new

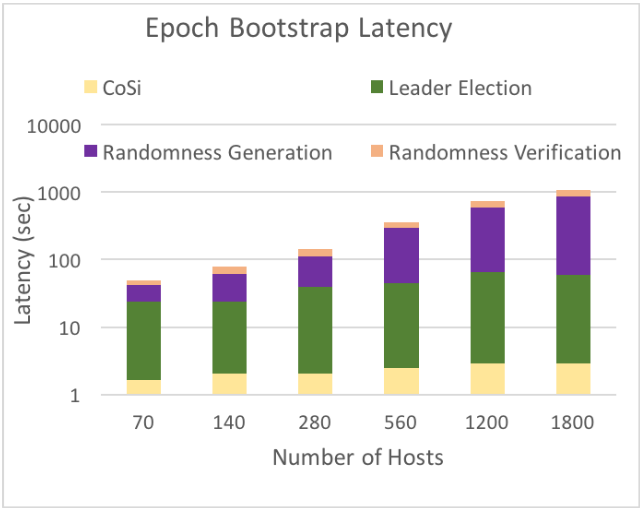

Figure 11. Epoch bootstrap latency

members, whereas the old ones who retained membership keep validating the same shard, in order to lower the epoch bootstrap costs.

7.3 Client-Side Latency for Cross-Shard Transactions

In this experiment a client submits transactions to private shards. We evaluate this version, as it is similar to validators submitting transactions to normal shards with the additional overhead of auditing. As shown in Figure 12, the client perceived latency is almost double than the consensus latency and slightly increases when multiple shards validate a transaction. This occurs because, when a transaction arrives at one shard, there are already on-the-fly blocks and, as a result, the inclusion of the transaction in a block is delayed.

Furthermore, we can see that the latency that auditing imposes for private shards can be almost equal to the consensus latency if the size of the auditor's group is equal to the size of the validators group. The total end-to-end latency would be higher if the clients also waited for the Output Shards to run consensus, but this is not needed. If they want to directly spend the new funds they can batch together the proof-of-acceptance and the expenditure transaction in order to respect the input-after-output constraint.

7.4 Multi-Shard Performance

In this experiment, we look into the max throughput the system can achieve with either a fixed number of hosts and a varying adversary or a fixed adversary and increasing number of hosts. The *Observed* data-series is the measured throughput, where each shard replays the same transactions internally. The *Normalized* data series is derived from assuming a canonical workload where all transactions have one input and two outputs (split) or one output and two inputs (merge), as we can construct arbitrary transactions from these primitives. This data series is calculated on the Observed data-series using the equation of Section 4.4.

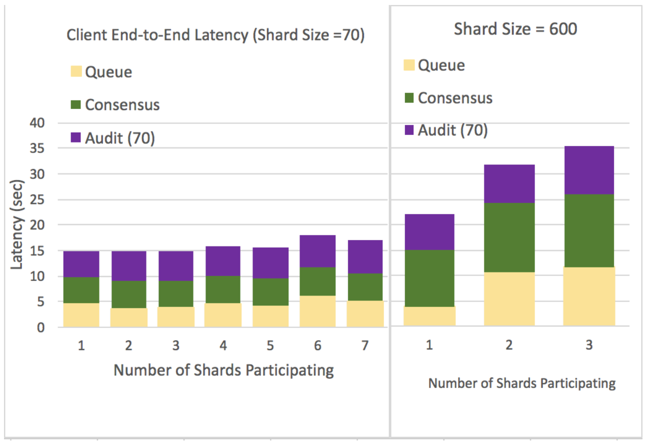

Figure 12. Client perceived, end-to-end latency

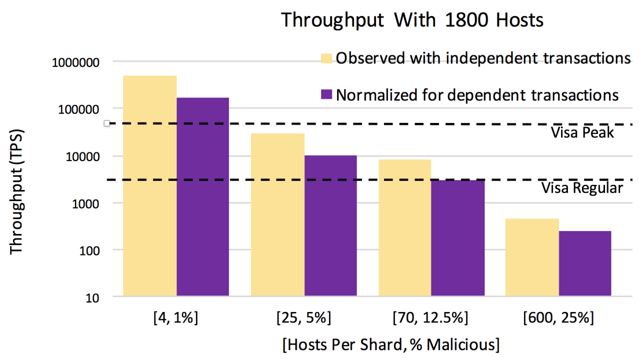

Figure 13. Throughput with varying adversary

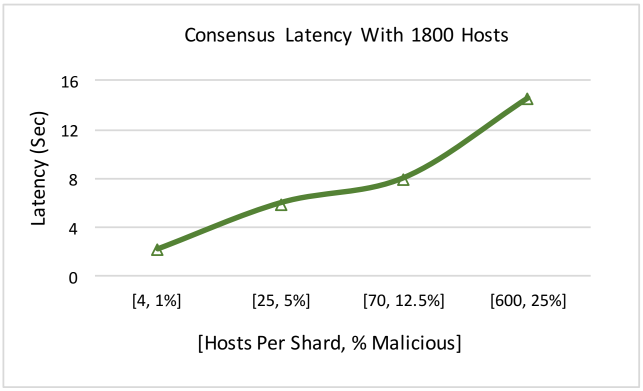

Figure 14. Latency with varying adversary

In Figures 13 and 14, we see that the performance depends on the assumed adversary, because with the same resources a 5% adversary instead of a 25% leads to throughput increase of two orders of magnitude. This assumption is too optimistic in a PoW system but might be acceptable for a PoS, while even an 1% adversary can be acceptable for centrally banked permissioned systems [18].

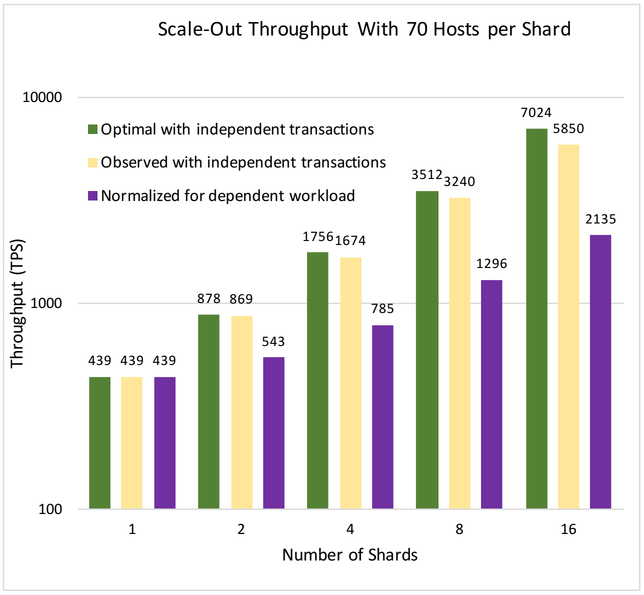

Figure 15. Scale-Out throughput with a fixed adversary

In Figure 15, we can see the performance of OmniLedger with a fixed adversary at 12.5% and increase number of shards from 1 to 16. As we can see, the Observed throughput increases almost linearly with the number of shards, whereas the Normalized starts to increase slower, because increasing the number of shards increases the probability of cross-shard transactions. Once the majority of transactions touches all the three shards they can, we see that the Normalized throughput increases almost linearly as well.

Finally, the *Optimal* data series is derived by multiplying the performance of one shard with the number of shards. The difference between Observed and Optimal increases as the number of hosts increase, due to the oversubscription of the physical servers. Finally, if we extrapolate the scalability, we can see that with 80 shards adding to a total of 5600 validators, less than the number of Bitcoin full nodes [8], the system could surpass the peek throughput of Visa [7, 57], even under this stronger adversary.

7.5 Cost of Bootstraping with State Blocks

In this experiment, we investigate the bootstrap improvement that SBs offer. A validator of OmniLedger has to crawl the membership blockchain first and then only get the latest SB, instead of replaying the full blockchain. We reconstructed Bitcoin's blockchain [9] and created a parallel OmniLedger blockchain that has SBs every week. We recovered the state size from [50]. Figure 16 depicts the bandwidth overhead of a validator that did not follow the state for 1 to 100 days. As we can, see the SB approach is better if the validator is stale for 19 days or 2736 Bitcoin blocks.

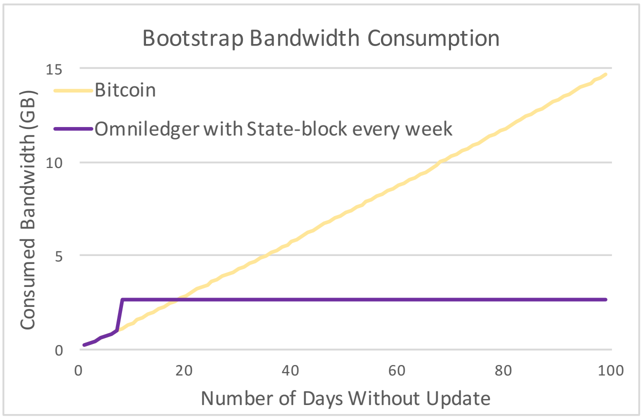

Figure 16. Bootstrap bandwidth consumption

The benefit might not feel substantial for Bitcoin but, in ByzCoin or OmniLedger, 2736 blocks are created in less than 8 hours, meaning that for one-day-long epoch, the SB approach is significantly better. This can provide further benefits when the peak throughput is needed and 16 MB blocks are deployed. In this case, the bandwidth benefit of SBs is expected to be close to two orders of magnitude.

8. Limitations and Future Work

OmniLedger is a proof-of-concept with several limitations. First, even if the epoch bootstrap does not interfere with the normal operation, its cost (in the order of minutes) is significant. We leave to future work the use of advanced cryptography, such as BLS [10] for performance improvements.

Furthermore, the actual throughput is dependent on the workload. If all transactions touch all the shards before committing, then the system is better-off with only one shard. We leave to future work the exploration of alternative ways of sharding, using locality measures instead of randomly. Performance wise the latency cannot be considered real-time, especially when the system is under heavy load.

Finally, even if OmniLedger is able to protect the rest of the system from a compromised private shard, the recovery process is left on the administrative level, which means that clients of this specific shard might lose part of their assets.

9. Related Work

There is a growing effort in scaling blockchain protocols. We provide a comparison in Table 9 of the most prominent systems introduced the last year. We already talked about ByzCoin [37] in Section 2. Although it is more scalable than PBFT, it does not scale-out, meaning that it can provide sufficient performance for a few hundreds nodes, but after 1000 participating hosts the consensus latency is as high as two minutes. Furthermore, if malicious validators are assigned at a high level on the communication tree, they can stop forwarding messages causing a Tree-DoS. Then, the leader would switch to a flat-topology reducing performance significantly.

Elastico [39] is the first system to experimentally prove that sharding is a viable approach for open, blockchain systems wherein participating processors have no pre-established identities. In Elastico, every round is bootstrapped from PoW similar to ByzCoin, then the least significant bits of the PoW distribute the miners to be authoritative on different shards. Once the setup has finished, every shard runs classic PBFT to reach consensus and a leader shard verifies all the signatures and creates a global block.

One of Elastico's weaknesses appears in its experimental configuration, where every shard has only 100 authoritative validators, a choice likely motivated by PBFT's lack of scalability. Shards this small yield a high failure probability of 2.76%[3] per shard per block under the 25% Byzantine adversary, which cannot safely be relaxed in a PoW system [29]. For 10-minute epoch and 16 shards, the failure probability is 97% over the course of one hour $(1-(1-0.0276*16)^6)$. A second weakness of Elastico is that transaction atomicity is not preserved on transactions that touch more than one shard, creating the potential of blocking funds forever. A final weakness is that the validators constantly switch shards, leading to the need for them to store the global state, which can hinder the total performance of the system. We did not compare experimentally with Elastico (whose performance is dominated by PBFT), as the biggest part of performance benefit for OmniLedger will come from ByzCoin [37] which is experimentally proven to be more performant than PBFT, even under failures. Furthermore, we were unable to find an open source implementation of Elastico.

In the permissioned setting, sharding is proposed as a scalable approach to transparent centrally banked cryptocurrency. RSCoin [18] uses a random numbers generated by a central bank, to split the validators. Each shard coordinates with the client to validate his transactions. RSCoin does not run BFT, but a simplified two-phase commit protocol, assuming that if the majority of validators are honest, safety is preserved. However, the system implicitly trusts the clients and does not provide pro-active security from a malicious client who can sent two conflicting transaction to two disjoint halves of validators and achieve two majorities, as long as he colludes with one only validator. Furthermore, RSCoin relies on the trusted third party for randomness generation and auditing, making the system problematic to use in a trustless setting.

Bitcoin-NG is a notable approach to scaling Bitcoin without changing the consensus algorithm. Its key insight, which ByzCoin also used, is that the PoW process does not have to be the same as the transaction validation process. This lead to the use of two separate timelines; one slow for PoW and one fast for transaction validation, significantly increasing the throughput of Bitcoin. However, Bitcoin-NG is still

[3] Cumulative binomial distribution ($P = 0.25$, $N = 100$, $X \geq 34$)

System	Scale-Out	Transaction Atomicity	State Blocks	Measured Scalability	Estimated Time to Fail	Measured Latency	Attacks
RsCoin [18]	In Permissioned	Partial	No	30	N/A	1 sec	Client
Elastico [39]	In PoW	No	No	1600	1 hour	800 sec	Shard Fail
ByzCoin [37]	No	N/A	No	1008	19 years	40 sec	Tree DoS
Bitcoin-NG [24]	No	N/A	No	1000	N/A	600 sec	Bitcoin & Leader
PBFT [13, 15]	No	N/A	No	16	N/A	1 sec	Censorship
Nakamoto [43, 14]	No	N/A	No	4000	N/A	600 sec	Bitcoin
OmniLedger	Yes	Yes	Yes	2400	358 years	20 sec	No

Table 3. Comparison of Scalable DLs

susceptible to the same attacks as Bitcoin [30, 3], with an additional Achilles heel on the leader of the epoch, that is not pro-actively checked.

Aspen [28] is a higher-level sharding protocol that builds on top of Bitcoin-NG [24] to further scale the protocol, by splitting the state into services. This approach is orthogonal to OmniLedger that is service agnostic and can be combined for further scalability benefits.

Other efforts to scale blockchains exist in the industry. Tendermint [13] is one such protocol that implements a protocol similar to PBFT for shard-level consensus. However, the benchmarking is done with a small number of validators and should be tested for hundreds instead of 16. Lightning network [48] is another industrial protocol that tries to scale Bitcoin by limiting the number of transactions exposed to a ledger and is also compatible with OmniLedger.

Buterin et al. [14] address the scaling problems of Ethereum with sharding, however the protocol is not evaluated and depends in randomness, without showing how to generate it. An interesting orthogonal system is Interledger [55] that is used to connect different cryptocurrencies. It is inspired by two-phase commit and relies on the existence of notaries to transfer funds between currencies.

Finally, in a different setting there are protocols that scale state-machine replication [40] and database replication [27] using, among other techniques, state sharding to scale-out [6] and split validators into groups [49]. However the assumptions are not compatible with DL systems and do not work with a Byzantine adversary among thousands of replicas, while preserving safety w.hp. OmniLedger could be used as a middleware for large scale replicated databases, however in order to provide Snapshot Isolation [34], it would need an additional global blockchain that sequences transactions before validation, similar to Hyperledger [15].

10. Conclusion

OmniLedger is the first system to securely scale-out distributed ledgers to Visa-level throughput coupled with seconds of latency, while preserving full decentralization and protecting against a Byzantine adversary. OmniLedger achieves this through a novel approach consisting of three steps. First, OmniLedger is designed with concurrency in mind, to identify and maximize the resource utilization while preserving safety. Second, OmniLedger preserves the ability to transact securely with any other user of the system. Finally, OmniLedger enables validators to securely and efficiently switch between shards, in order to verify any transaction.

References

[1] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus. *CoRR*, abs/1612.02916, 2016.

[2] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 181–194, Denver, CO, June 2016. USENIX Association.

[3] M. Apostolaki, A. Zohar, and L. Vanbever. Hijacking Bitcoin: Large-scale Network Attacks on Cryptocurrencies. *38th IEEE Symposium on Security and Privacy*, May 2017.

[4] A. Back. Hashcash – A Denial of Service Counter-Measure, Aug. 2002.

[5] I. Bentov, R. Pass, and E. Shi. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919, 2016. `http://eprint.iacr.org/2016/919`.

[6] C. E. Bezerra, F. Pedone, and R. Van Renesse. Scalable state-machine replication. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 331–342. IEEE, 2014.

[7] Bitcoin Wiki. Scalability, 2016.

[8] Bitnodes. Bitcoin Network Snapshot, April 2017. `https://bitnodes.21.co/nodes/`.

[9] Blockchain.info. Blockchain Size, Feb. 2017. `https://blockchain.info/charts/blocks-size`.

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

[11] J. Bonneau, J. Clark, and S. Goldfeder. On Bitcoin as a public randomness source. IACR eprint archive, Oct. 2015.

[12] M. Borge, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, and B. Ford. Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies. In *1st IEEE Security and Privacy On The Blockchain*, Apr. 2017.

[13] E. Buchman. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains, 2016.

[14] V. Buterin, J. Coleman, and M. Wampler-Doty. Notes on Scalable Blockchain Protocols (verson 0.3), 2015.

[15] C. Cachin. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[16] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Feb. 1999.

[17] K. Croman, C. Decke, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. S. an, and R. Wattenhofer. On Scaling Decentralized Blockchains (A Position Paper). In *3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[18] G. Danezis and S. Meiklejohn. Centrally Banked Cryptocurrencies. *23rd Annual Network & Distributed System Security Symposium (NDSS)*, Feb. 2016.

[19] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin Meets Strong Consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN), Singapore*, January 2016.

[20] S. Deetman. Bitcoin Could Consume as Much Electricity as Denmark by 2020, May 2016.

[21] DeterLab Network Security Testbed, September 2012.

[22] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[23] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, T. Ristenpart, and C. Tech. The pythia prf service. *IACR Cryptology ePrint Archive*, 2015:644, 2015.

[24] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA, Mar. 2016. USENIX Association.

[25] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[26] M. Franklin and H. Zhang. Unique ring signatures: A practical construction. In *International Conference on Financial Cryptography and Data Security*, pages 162–170. Springer, 2013.

[27] R. Garcia, R. Rodrigues, and N. Preguiça. Efficient middleware for byzantine fault tolerant database replication. In *Proceedings of the sixth conference on Computer systems*, pages 107–122. ACM, 2011.

[28] A. E. Gencer, R. van Renesse, and E. G. Sirer. Short Paper: Service-Oriented Sharding for Blockchains. *Financial Cryptography and Data Security 2017*, 2017.

[29] A. Gervais, G. Karame, S. Capkun, and V. Capkun. Is Bitcoin a decentralized currency? *IEEE security & privacy*, 12(3):54–60, 2014.

[30] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[31] The Go Programming Language, Sept. 2016.

[32] T. Hanke and D. Williams. Intoducing Random Beascons Using Threshold Relay Chains, Sept. 2016.

[33] E. G. S. Ittay Eyal. It's Time For a Hard Bitcoin Fork, June 2014.

[34] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems (TODS)*, 25(3):333–379, 2000.

[35] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. Cryptology ePrint Archive, Report 2016/889, 2016. `http://eprint.iacr.org/2016/889`.

[36] E. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford. Managing Identities Using Blockchains and CoSi. Technical report, 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016), 2016.

[37] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium*, 2016.

[38] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[39] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.

[40] P. J. Marandi, M. Primi, and F. Pedone. High performance state-machine replication. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 454–465. IEEE, 2011.

[41] R. Merkle. A certified digital signature. In *Advances in CryptologyCRYPTO 89 Proceedings*, pages 218–238. Springer, 1990.

[42] S. Micali, S. Vadhan, and M. Rabin. Verifiable Random Functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 120–130. IEEE Computer Society, 1999.

[43] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[44] NIST. NIST Randomness Beacon, 2017.

[45] K. Ohta and T. Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Jan. 1999.

[46] R. Pass and E. Shi. Hybrid Consensus: Efficient Consensus in the Permissionless Model. Cryptology ePrint Archive, Report 2016/917, 2016. `http://eprint.iacr.org/2016/917`.

[47] R. Pass, C. Tech, and L. Seeman. Analysis of the Blockchain Protocol in Asynchronous Networks. *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2017.

[48] J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments, Jan. 2016.

[49] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-scale Byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd Workshop on Hot Topics in System Dependability*, 2007.

[50] Satoshi.info. Unspent Transaction Output Set, Feb. 2017.

[51] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[52] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *IACR International Cryptology Conference (CRYPTO)*, pages 784–784, 1999.

[53] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable Bias-Resistant Distributed Randomness. In *38th IEEE Symposium on Security and Privacy*, May 2017.

[54] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *37th IEEE Symposium on Security and Privacy*, May 2016.

[55] S. Thomas and E. Schwartz. A protocol for interledger payments. 2015.

[56] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *14th International Conference on Network Protocols (ICNP)*, Nov. 2006.

[57] Visa. Visa Inc. at a Glance, 2017.

[58] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[59] Wikipedia. Proof of Existence, April 2017. `http://en.wikipedia.org/wiki/Proof_of_Existence`.

[60] Wikipedia. Atomic commit, Feb. 2017.

[61] Wikipedia. Shard (database architecture), Feb. 2017.

[62] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*, 2014.

[63] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse. REM: Resource-Efficient Mining for Blockchains. Cryptology ePrint Archive, Report 2017/179, 2017. `http://eprint.iacr.org/2017/179`.