PolyShard: Coded Sharding Achieves Linearly Scaling Efficiency and Security Simultaneously

Songze Li*, Mingchao Yu*, A. Salman Avestimehr*, Sreeram Kannan†, and Pramod Viswanath‡
*University of Southern California
†University of Washington
‡University of Illinois at Urbana-Champaign

Abstract

Today's blockchains do not scale in a meaningful sense. As more nodes join the system, the efficiency of the system (computation, communication, and storage) degrades, or at best stays constant. A leading idea for enabling blockchains to scale efficiency is the notion of sharding: different subsets of nodes handle different portions of the blockchain, thereby reducing the load for each individual node. However, existing sharding proposals achieve efficiency scaling by compromising on trust - corrupting the nodes in a given shard will lead to the permanent loss of the corresponding portion of data. We observe that sharding is similar to replication coding, which is known to be inefficient and fragile in the coding theory community. In this paper, we demonstrate a new protocol for coded storage and computation in blockchains. In particular, we propose `PolyShard`: "polynomially coded sharding" scheme that achieves information-theoretic upper bounds on the efficiency of the storage, system throughput, as well as on trust, thus enabling a truly scalable system. We provide simulation results that numerically demonstrate the performance improvement over state of the arts, and the scalability of the `PolyShard` system. Finally, we discuss potential enhancements, and highlight practical considerations in building such a system.

I. Introduction

Blockchain systems, which maintain a distributed trusted ledger and can do finite-state computations, can execute a wide range of programs in a trust-free setting. This has promised a host of new and exciting applications in various areas including digital cryptocurrency [1], industrial IoT [2], and healthcare management [3]. However, to enable these applications, we need blockchain systems that *scale well with the number of participating nodes* [4]. The scalability is with respect to three important performance metrics: the *throughput* - measured as the number of transactions verified in a distinct period of time, the *storage efficiency* - measured as the maximum size of the blockchain that can be handled by nodes with fixed storage sizes, and the *security* - measured as the number of malicious nodes the system can tolerate. By scalable, we mean that all these three metrics should *improve* as the number of nodes N increases; this is a remarkably high expectation – state of the art scaling solutions [5]–[9] expect performance metrics to not get worse (at best) as the number of nodes N increases.

Scalability has been one of the greatest challenges faced by current blockchain systems like Bitcoin [1] and Ethereum [10]. For example, Bitcoin presently restricts its block size to 1 MB, and processing rate to 7 transactions/sec [11], inherently restricting the throughput. The main reason that hinders these conventional technologies from scaling is that their key underpinning storage and computation methods involve *full replication*: Each network node stores the entire blockchain and replicates all the computations (verifications).

This feature gives a full replication system high security (of tolerating 49% adversarial nodes). But the storage efficiency and throughput of the system is compromised: They stay constant regardless of the network size. In practice, the computational burden even increases with the number of nodes (e.g., mining puzzles get harder as time progresses and more users participate), causing the throughput to drop.

However, such a trade-off is far from optimal from information theoretical point of view. Given the $N\times$ computation and $N\times$ storage resources across all the N network nodes, the following information-theoretic upper bounds hold:

$$\text{security} \leq \Theta(N); \quad \text{throughput} \leq \Theta(N); \quad \text{storage efficiency} \leq \Theta(N).$$

It is intuitive that these bounds can be *simultaneously* achieved by a centralized system, allowing all the 3 metrics to scale. If a carefully designed decentralized system can provide the same scalability, this would represent a significant progress over full replication and signal true scalability of blockchains – as more nodes participate in the system, all-round performance improves. Such are the goals of this paper.

To close this gap, the leading solution being discussed in the blockchain literature is via *sharding* [7]–[9]. The key idea is to break up (or "shard") the blockchain into fragments which are then replicated. This way, both storage and computation requirements are reduced by a factor equal to the number of shards K. However, in order to allow storage and throughput to scale linearly with N, the number of nodes $q = \frac{N}{K}$ participating in a shard has to be held constant. Consequently, an attacker only needs to control as less as $q/2$ nodes to compromise a shard, yielding a security level of $q/2$, which approaches zero with increasing N. Although various efforts have been made to alleviate this security issue (e.g., by periodically shuffling the

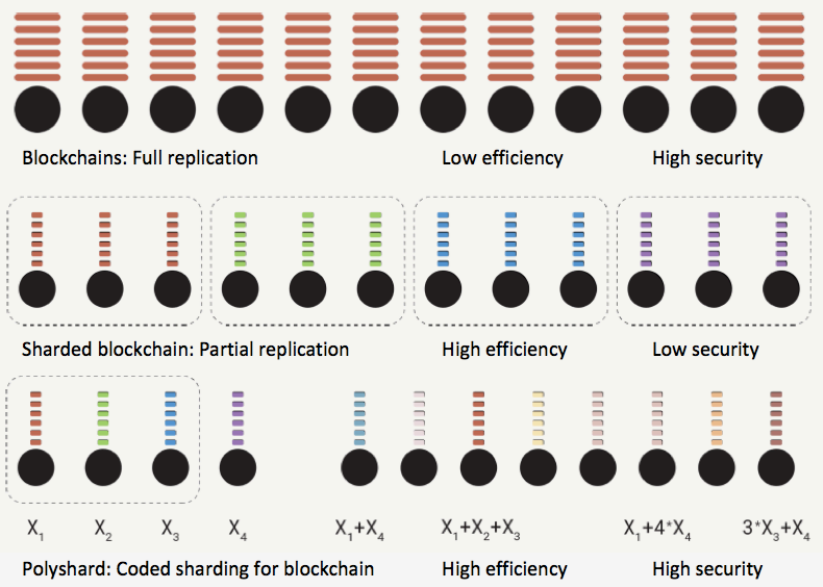


Fig. 1: Blockchain, conventional sharding and `Polyshard`.

nodes [8]), they are susceptible to powerful adversaries (e.g., who can corrupt the nodes *after* the shuffling), yet none achieves security scalability.

In summary, both full replication and sharding based blockchain systems make trade-offs between the scalability of throughput, storage efficiency, and security. A widely open fundamental question is thus:

> Is there a blockchain design that *simultaneously* scales storage efficiency, security, and throughput?

In this paper, we answer this question affirmatively by introducing the concept of *coded sharding*. In particular, we propose `PolyShard` (polynomially coded sharding), a scheme that simultaneously scales throughput, storage efficiency, and security by $\Theta(N)$. We show mathematically that `Polyshard` achieves all three information-theoretic upper bounds and enables a truly scalable blockchain system. (Table I)

TABLE I: Performance comparison of the proposed `PolyShard` verification scheme with other benchmarks and the information-theoretic limits.

	Storage efficiency	Security	Throughput
Full replication	$O(1)$	$\Theta(N)$	$O(1)$
Sharding	$\Theta(N)$	$O(1)$	$\Theta(N)$
Information-theoretic limit	$\Theta(N)$	$\Theta(N)$	$\Theta(N)$
`PolyShard` (this paper)	$\Theta(N)$	$\Theta(N)$	$\Theta(N)$

`PolyShard` is inspired by recent developments in *coded computing* [12]–[20], in particular Lagrange Coded Computing [20], which provides a transformative framework for injecting computation redundancy in unorthodox coded forms in order to deal with failures and errors in distributed computing. The key idea behind `PolyShard` is that instead of storing and processing a single uncoded shard as in convention, each node stores and computes on a coded shard of the same size that is generated by linearly mixing uncoded shards, using the well-known Lagrange polynomial. This coding provides computation redundancy to simultaneously provide security against erroneous results from malicious nodes, which is enabled by noisy polynomial interpolation techniques (e.g., Reed-Solomon decoding). The illustration in Fig. 1 compares and contrasts `Polyshard`, sharding and regular storage approaches in blockchains. The dark circles correspond to nodes, above which the stored blockchain is shown. The different colors in the second row correspond to distinct shards, and `Polyshard` mixes the shards (colors) .

While coding is generally applicable in many distributed computing scenarios, the following two salient features make `PolyShard` particularly suitable for blockchain systems.

- *Oblivious*: The coding strategy applied to generate coded shards is oblivious of the verification function. That means, the same coded data can be simultaneously used for multiple verification items (examples: digital signature verification and balance verification in a payment system);
- *Incremental*: `PolyShard` allows each node to grow its local coded shard by coding over the newest verified blocks, without needing to access the previous ones. This helps to maintain a constant coding overhead as the chain grows.

As a proof of concept, we simulate a payment blockchain system, which keeps records of all the balance transfers between clients, and verifies new blocks by comparing them with the sum of the previously verified blocks. We run experiments on this system for various combinations of network size and chain length, and measure/evaluate the throughput, storage, and security achieved by the full replication, uncoded sharding, and the `PolyShard` schemes. As we can see from the measurements plotted in Fig. 2, `PolyShard` indeed achieves the throughput scaling with network size as the uncoded sharding scheme, improving significantly over the full replication scheme. These experiments provide an empirical verification of the theoretical guarantees of `PolyShard` in simultaneously scaling storage efficiency, security, and throughput.

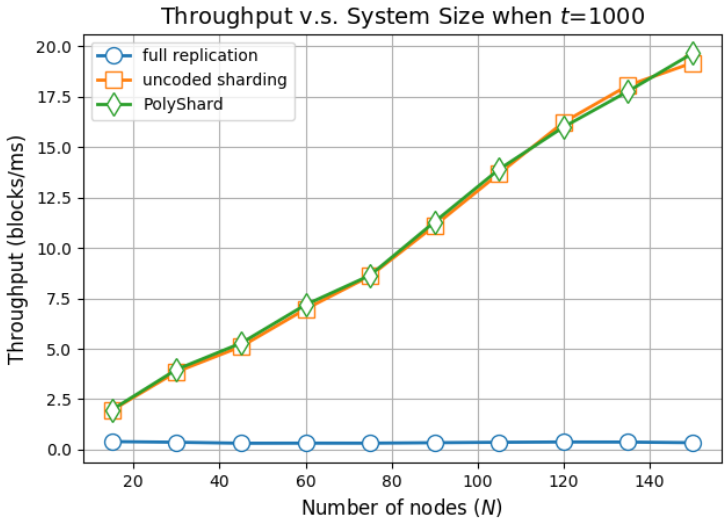


Fig. 2: Measured throughput of verification schemes; here number of epochs $t = 1000$.

In the following, we list the main contributions of this paper.

- Drawing upon principles from information and coding theory, we propose a radically different scalable design methodology called *coded sharding*.
- We propose an instantiation of the general coded sharding principle called `PolyShard`, which stands for Lagrange polynomial based coded sharding.
- We prove that `PolyShard` provides $\Theta(N)$ scalability to both efficiency (throughput and storage) and security;
- We corroborate our theoretic findings with simulation and evaluation of a payment blockchain system. We implemented `PolyShard` in this system, and numerically demonstrate its efficiency and security scalibility.

Related works. Sharding is the state of the art technique to scale the computation, communication, and storage of blockchain systems [7]–[9], [21]–[32]. The key idea is to partition the entire blockchain into K shards, each of which is managed by a disjoint subset of nodes (miners). This reduces the required amount of computation by a factor of K. As an example, a sharding protocol `ELASTICO` was proposed in [7], in which the the incoming transactions are partitioned into shards, and each shard is verified by a disjoint committee of nodes in parallel. A decentralized sharding protocol `OmniLedger` [8] improved upon `ELASTICO` in multiple avenues, including new methods to assign nodes into shards with a higher security guarantee, proposed an atomic protocol for cross-shard transactions, and further optimized the communication and storage designs. For all current sharding systems, in order to scale out with the number of nodes N, the number of shards K needs to grow with N, i.e., $K = O(N)$. This results in a constant number of nodes in each shards, which makes the system more susceptible to adversaries as the network expands. Current systems achieve security through system solutions such as shard rotation [7], [8], [33]. For example, `OmniLedger` leverages `RandHound` [34], a bias-resistant distributed randomness generation protocol, to randomly sampling the subset of users assigned to a shard, and randomly updating the these subsets regularly over time. Such methods can provide near-optimal security when the adversarial nodes are fixed prior to the random assignment, but are susceptible when the adversary is dynamic and can corrupt nodes *after* they have been assigned to the shards.

II. Problem Formulation: Block Verification

We consider a blockchain system that consists of K independent shards, with each client associated to only one shard. For clarity of presentation we focus on transactions that are verifiable intra-shard; cross-shard verifications are an added complexity complementary to the contributions of this paper; for instance, the atomic payment and locking mechanisms of [8] can be naturally incorporated with the ideas in this paper. We define the computation and networking models, and then performance metric below.

A. Computation model

Each shard k, $k \in [1, K]$ maintains its own sub-chain. We use t to denote discrete time in rounds and $Y_k(t) \in \mathbb{U}$ denotes the verified block in round t at shard k, where $\mathbb{U}$ is a vector space over a field $\mathbb{F}$. Then the sub-chain at shard k till time $t-1$ is denoted as $Y_k^{t-1} = (Y_k(1), Y_k(2), \ldots, Y_k(t-1))$. We wish to validate a new block $X_k(t) \in \mathbb{U}$ proposed in shard k in round t with respect to the past of the sub-chain Y_k^{t-1} to ensure that it does not contain any double-spends or other irregularities. We abstract out the mechanism which generates the proposal and instead focus on validating the proposed transaction. We note that the proposal can be generated by a small number of nodes, while security depends on the number of nodes that validates the transactions. Note that since transactions are intra-shard, the sub-chain is sufficient to verify whether $X_k(t)$ is valid. To verify $X_k(t)$, shard k computes a verification function $f^t : \mathbb{U}^t \to \mathbb{V}$, over $X_k(t)$ and the sub-chain Y_k^{t-1}, for some vector space $\mathbb{V}$ over $\mathbb{F}$.

Having obtained $h_k^t = f^t(X_k(t), Y_k^{t-1})$, shard k computes an indicator variable z_k^t, such that $z_k^t = \mathbb{1}(h_k^t \in \mathcal{W})$, where $\mathcal{W} \subseteq \mathbb{V}$ denotes the subset of the values of the verification function for which the block is valid. A simple example is when $\mathbb{V} = \{0, 1\}$, the output is binary and denotes the validity, i.e., $\mathcal{W} = \{1\}$. Finally, the verified block $Y_k(t)$ is computed as $Y_k(t) = z_k^t X_k(t)$, i.e., if the block is not valid, it is treated as a null block which is represented as 0 in the field.

Without loss of generality, we model the verification function f^t as a multivariate polynomial of degree d, motivated by the following result [35]: Any Boolean function $\{0, 1\}^n \to \{0, 1\}$ can be represented by a polynomial of degree $\leq n$ with at most 2^{n-1} terms. Due to this result, common verification functions, such as balance check and digital signature verification, can all be transformed into polynomials; details are available in Appendix A.

Example 1 (Balance verification)**.** We consider a simple payment blockchain system that keeps records of the balance transfers between clients. We assume that there are M clients in each shard, for some constant M that does not scale with t. In this scenario, a block submitted to shard k at time t, $X_k(t)$, consists of multiple transactions, and is represented by a pair of real vectors $X_k(t) = (X_k^{\text{send}}(t), X_k^{\text{receive}}(t))$, for some $X_k^{\text{send}}(t), X_k^{\text{receive}}(t) \in \mathbb{R}^M$. A transaction that reads "Client p sends s dollars to client q." will deduct s from $X_k^{\text{send}}(t)[p]$ and add s to $X_k^{\text{receive}}(t)[q]$. Clients that do no send/receive money will have their entries in the send/receive vectors set to zeros. To verify $X_k(t)$, we need to check that all the senders in $X_k(t)$ have accumulated enough unspent funds from previous transactions. This naturally leads to the following verification function.

$$f^t(X_k(t), Y_k^{t-1}) = X_k^{\text{send}}(t) + \sum_{i=1}^{t-1} (Y_k^{\text{send}}(i) + Y_k^{\text{receive}}(i)). \tag{1}$$

Here the verification polynomial f^t has constant degree $d = 1$, with $\mathbb{F} = \mathbb{R}$, $\mathbb{U} = \mathbb{R}^M$, and $\mathbb{V} = \mathbb{R}^M$. We claim the block $X_k(t)$ valid if none of the entries of the above verification function is negative, and we set $Y_k(t) = X_k(t)$ and append it the sub-chain of shard k. Otherwise, we append a dummy all-zero block.

B. Networking model

The above blockchain system is implemented distributedly over N network nodes. A subset $\mathcal{M} \subset \{1, \ldots, N\}$ of these nodes may be malicious/adversarial, and the malicious nodes compute and communicate arbitrarily erroneous results during the process of block verification. Only the honest nodes will follow the designed networking protocol described below.

At time t, each node i, $i = 1, 2, , \ldots, N$, locally stores some data, denoted by $Z_i^{t-1} = (Z_i(1), Z_i(2), \ldots, Z_i(t-1))$, where $Z_i(j) \in \mathbb{W}$ for some vector space $\mathbb{W}$ over $\mathbb{F}$. The locally stored data Z_i^{t-1} is generated from all shards of the blockchain using some function ϕ_i^{t-1}, i.e., $Z_i^{t-1} = \phi_i^{t-1}(Y_1^{t-1}, Y_2^{t-1}, \ldots, Y_K^{t-1})$.

Next, given the K incoming blocks $\{X_k(t)\}_{k=1}^K$, each node i computes an intermediate result g_i^t using some function ρ_i^t on the incoming blocks and its local storage, such that $g_i^t = \rho_i^t(X_1(t), X_2(t), \ldots, X_K(t), Z_i^{t-1})$, and then broadcasts the result g_i^t to all other nodes.

Having received all the broadcast messages, each node i decodes the verification results for all K shards $\hat{h}_{1i}^t, \hat{h}_{2i}^t, \ldots, \hat{h}_{Ki}^t$, using some function ψ_i^t, i.e., $(\hat{h}_{1i}^t, \hat{h}_{2i}^t, \ldots, \hat{h}_{Ki}^t) = \psi_i^t(g_1^t, g_2^t, \ldots, g_N^t)$. Using these decoded results, node i computes the indicator variables $(\hat{z}_{1i}, \hat{z}_{2i}, \ldots, \hat{z}_{Ki})$, and then the verified blocks $\hat{Y}_{ki}(t) = \hat{z}_{ki}^t X_k(t)$, for all $k = 1, 2, \ldots, K$.

Finally, each node i utilizes the verified blocks to update its local storage using some function χ_i^t, i.e., $Z_i^t = \chi_i^t(\hat{Y}_{1i}(t), \hat{Y}_{2i}(t), \ldots, \hat{Y}_{Ki}(t), Z_i^{t-1})$.

We say that a block verification scheme S, defined as a sequence of collections of the above functions, i.e., $S = (\{\rho_i^t, \psi_i^t, \phi_i^t\}_{i=1}^N)_{i=1}^\infty$, is b-secure, if for any subset $\mathcal{M} \subset \{1, \ldots, N\}$ of malicious nodes with $|\mathcal{M}| \leq b$, and each node $i \notin \mathcal{M}$, we have $(\hat{h}_{1i}^t, \hat{h}_{2i}^t, \ldots, \hat{h}_{Ki}^t) = (h_1^t, h_2^t, \ldots, h_K^t)$, for all $t = 1, 2, \ldots$. That is, a blockchain system using b-secure verification scheme can guarantee the correct verification results at the honest nodes in the presence of b malicious nodes. We define the computational complexity of a function f, denoted by $c(f)$, as the number of additions and multiplications performed in the domain of f to evaluate f. In this paper, we are interested in the following three performance metrics.

Storage efficiency. Denoted by γ_S, it is the ratio between the size of the entire block chain and the size of the data stored at each node, i.e.,

$$\gamma_S \triangleq \frac{K \log |\mathbb{U}|}{\log |\mathbb{W}|}. \tag{2}$$

The above definition also applies to a probabilistic formulation where the blockchain elements $Y_k(j)$s and the storage elements $Z_i(j)$s are modelled as i.i.d. random variables with uniform distribution in their respective fields, where the storage efficiency is defined using the entropy of the random variables.

Security. Denoted by β_S, it is the maximum value of b such that a verification scheme S is b-secure. That is,

$$\beta_S \triangleq \sup\{b : S \text{ is } b\text{-secure}\}. \tag{3}$$

Throughput. Denoted by λ_S, it is the average number of blocks that are correctly verified per unit discrete round, which includes all the computations performed at all N nodes to verify the incoming K blocks. That is,

$$\lambda_S \triangleq \liminf_{t\to\infty} \frac{K}{\sum_{i=1}^{N}(c(\rho_i^t)+c(\psi_i^t)+c(\chi_i^t))/(Nc(f^t))}. \tag{4}$$

We aim at studying the information-theoretic bounds on these metrics and designing a scheme that can simultaneously achieve all of them.

III. Baseline Performance

In this section, we first present the information-theoretic *upper* bounds on the three performance metrics for any blockchain. We then study the performance of two state-of-the-art blockchain schemes and comment on the gap with the upper bounds.

Information-theoretic upper bounds. In terms of security, the maximum number of adversaries any verification scheme can tolerate cannot exceed half of the number of network nodes N: thus, the security $\beta \leq \frac{N}{2}$. In terms of storage, for the verification to be successful, the size of the chain should not exceed the aggregated storage resources of the N nodes. Otherwise, the chain cannot be fully stored. We thus have $\gamma \leq N$. Finally, to verify the K incoming blocks, the verification function f^t must be executed at least K times in total. Hence, the system throughput $\lambda \leq \frac{K}{K/N} = N$. Therefore, the information-theoretic upper bounds of security, storage efficiency, and throughput all scale linearly with the network size N.

Full replication. In terms of storage efficiency, since each node stores all the K shards of the entire blockchain, full replication scheme yields $\gamma_{\text{full}} = 1$. Since every node verifies all the K blocks, the throughput of the full replication scheme is $\lambda_{\text{full}} = \frac{K}{NKc(f^t)/(Nc(f^t))} = 1$. Thus the full replication scheme does not scale with the network size, as both the storage and the throughput remain constant as N increases. The advantage is that the simple majority-rule will allow the correct verification and update of every block as long as there are less than $N/2$ malicious nodes. Thus, $\beta_{\text{full}} = N/2$.

Uncoded sharding scheme. In *sharding*, the block chain consists of K equal-size (of size $q = N/K$) disjoint sub-chains known as shards. Each group of nodes is responsible for managing a single shard; this is a full replication system with $K' = 1$ shard and $N' = q$ nodes. Since each node stores and verifies a single shard, the storage efficiency and throughput become $\gamma_{\text{sharding}} = K$ and $\lambda_{\text{sharding}} = \frac{K}{Nc^t/(Nc^t)} = K$, respectively. For these two metrics to scale linearly with N, it must be true that $K = \Theta(N)$. Consequently, the group size q becomes a constant. Hence, compromising as few as $q/2$ nodes will corrupt one shard and the chain. Thus, this scheme only has a constant security of $\beta_{\text{sharding}} = q/2 = O(1)$. Although system solutions such as shard rotations can help achieve linearly scaling security guarantees, they are only secure when the adversary is non-adaptive (or very slowly adaptive). When the adversary is dynamic, it can corrupt all nodes belonging to a particular shard instantaneously after the shard assignment has been made. Under this model, the security reduces to a constant.

In summary, neither full replication nor the above sharding schemes can *simultaneously* scale the storage efficiency, security, and throughput towards the information-theoretic bounds. We note that codes for storage or distributed storage [36], [37] cannot be directly used here since we need to be able to calculate validation functions directly over coded blocks. This motivates us to propose `PolyShard` (polynomially coded sharding) in the next section to achieve all of the three bounds simultaneously. For clarity, from now on, we will refer to the sharding scheme as "uncoded sharding".

IV. Main Results

Theorem 1. *For a blockchain system consisting of multiple shards, each of length t, a polynomial verification function f^t with constant degree d, operated on N network nodes, up to μ (for some constant $0 \leq \mu < \frac{1}{2}$) fraction of which may be malicious, the following performance metrics are simultaneously achievable,*

$$\text{Storage efficiency} \quad \gamma = \left\lfloor \frac{(1-2\mu)}{d} N + 1 \right\rfloor = \Theta(N), \tag{5}$$

$$\text{Security} \quad \beta = \mu N = \Theta(N), \tag{6}$$

$$\text{Throughput} \quad \lambda = \left\lfloor \frac{(1-2\mu)}{d} N + 1 \right\rfloor = \Theta(N), \tag{7}$$

for computational complexity of the verification function $c(f^t) = O(t)$. Therefore, the information-theoretically optimal storage efficiency, security, and throughput can be simultaneously achieved within constant multiplicative gaps.

The above performance metrics are achieved by a *coded* verification scheme proposed in this paper, named `PolyShard`. To prove Theorem 1, we describe and analyze `PolyShard` in the next section.

Remark 1. Using PolyShard, each node stores locally a coded shard generated as a linear combination of the original uncoded shards. Upon reception of the new blocks, each node linearly combines them to create a coded block, using the same set of coefficients. Then, the node computes the verification function over the coded block and its local storage. The computation results are collected and used to decode the intended verification results using Reed-Solomon decoding. This coded computation technique was originally proposed in [20] for distributed computing multivariate polynomials subject to computation errors, where Lagrange polynomial interpolation was used to generate the coded data.

Remark 2. Compared with the scenario of one-shot computation on static data in [20], the local storage at each network node is growing in a blockchain system as more verified blocks are appended to the chain. The requirement of dynamically updating the local storage that is compatible with the upcoming coded verification poses new challenges on the design of the PolyShard scheme. Utilizing the data structure of the blockchain, and the algebraic properties of the encoding strategy, we propose a simple *incremental* storage update policy for PolyShard that requires accessing the minimum amount of data.

Remark 3. The additional coding overhead, including the number of operations required to encode the input data, decode verification results, and update the local storage, does not scale with the length of the sub-chains t. As a result, the performance of PolyShard depends on the model of the verification polynomial f^t. When the cost of computing f^t, i.e., $c(f^t)$, increases with t (e.g., when $c(f^t) = O(t)$), the coding overhead becomes negligible as the chain grows.

V. Polynomially Coded Sharding (PolyShard) Scheme

A. Storage encoding

We pick K distinct elements $\omega_1, \omega_2, \ldots, \omega_K \in \mathbb{F}$, one each corresponding to each shard k, and create the following Lagrange polynomial

$$u^{t-1}(z) = \sum_{k=1}^{K} Y_k^{t-1} \prod_{i \neq k} \frac{z - \omega_i}{\omega_k - \omega_i}. \tag{8}$$

This polynomial is designed such that $u^{t-1}(\omega_k) = Y_k^{t-1}$ for all $k = 1, 2, \ldots, K$.

Next, as shown in Fig. 3, we pick N distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbb{F}$, one for each node. This creates N coded sub-chains, denoted by $\tilde{Y}_1^{t-1}, \tilde{Y}_2^{t-1}, \ldots, \tilde{Y}_N^{t-1}$, by evaluating the above $u^{t-1}(z)$ at the points $\alpha_1, \alpha_2, \ldots, \alpha_N$. That is, for all $i = 1, 2, \ldots, N$,

$$\tilde{Y}_i^{t-1} = u^{t-1}(\alpha_i) = \sum_{k=1}^{K} Y_k^{t-1} \prod_{t \neq k} \frac{\alpha_i - \omega_t}{\omega_k - \omega_t} = \sum_{k=1}^{K} \ell_{ik} Y_k^{t-1}, \tag{9}$$

We note that $\tilde{Y}_i^{t-1}$ is encoded as a linear combination of the uncoded sub-chains $Y_1^{t-1}, Y_2^{t-1}, \ldots, Y_K^{t-1}$, and the coefficients ℓ_{ik}s do not depend on the time index t. Therefore one can think of each node i as having a fixed linear vector ℓ_{ik} by which it mixes the different shards to store the $\tilde{Y}_i^{t-1}$ on the node. The size of each coded sub-chain is K times smaller than the size of the entire blockchain, and the storage efficiency of PolyShard is $\gamma_{\texttt{PolyShard}} = K$.

For this encoding to be viable, we need large enough field such that $|\mathbb{F}| \geq N$. For small field (e.g., binary field), we can overcome this issue by using field extension and applying PolyShard on the extended field (see details in Appendix A).

Remark 4. The above data encoding is oblivious of the verification function, i.e., the coefficients ℓ_{ik} are independent of f^t. Therefore, the data encoding of PolyShard can be carried out independently of the verification, and the same coded storage can be simultaneously used for all different types of verification items, which could include verifying account balances, digital signatures or checking smart contracts.

B. Coded verification

In time t, K blocks $X_1(t), X_2(t), \ldots, X_K(t)$ are submitted to the K shards for verification. The PolyShard scheme verifies these blocks in three steps.

Step 1: block encoding. From the received the K blocks, each node i computes a coded block $\tilde{X}_i(t)$ as a linear combination using the same set of coefficients as in (9). That is, $\tilde{X}_i(t) = \sum_{k=1}^{K} \ell_{ik} X_k(t)$. We note that this encoding operation can be also viewed as evaluating the polynomial $u_t(z) = \sum_{k=1}^{K} X_k(t) \prod_{i \neq k} \frac{z - \omega_i}{\omega_k - \omega_i}$ at the point α_i. This step incurs $O(NK)$ operations across the network, since each of the N nodes computes the linear combination of K blocks.

Step 2: local computation. Each node i applies the function f^t to the coded block $\tilde{X}_i(t)$, and its locally stored coded sub-chain $\tilde{Y}_i^{t-1}$ to compute $g_i^t = f^t(\tilde{X}_i(t), \tilde{Y}_i^{t-1})$. This step requires a total of $Nc(f^t)$ operations across the network since each validation operation takes $c(f^t)$ steps. Having finished the local computations, each node i broadcasts its computation result g_i^t to all other nodes.

Step 3: decoding. Using the computation results $g_1^t, \ldots, g_N^t$, a maximum μ fraction of which may be erroneous from malicious nodes, each node decodes the intended results $\{f^t(X_k(t), Y_k^{t-1})\}_{k=1}^{K}$. Since $f^t(u_t(z), u^{t-1}(z))$ is a univariate polynomial of degree $(K-1)d$, g_i^t can be viewed as the evaluation of $f^t(u_t(z), u^{t-1}(z))$ at α_i, and it can be recovered following the process

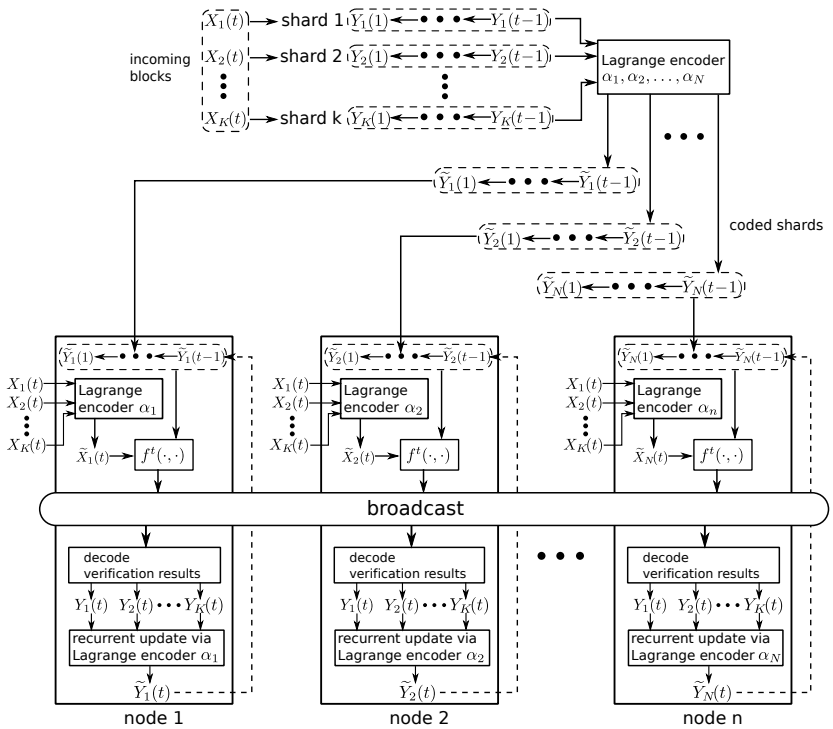


Fig. 3: Illustration of PolyShard scheme.

of decoding a Reed-Solomon code with dimension $(K-1)d+1$ and length N (see, e.g., [38]). In order for this decoding to be robust to μN malicious nodes (i.e., achieving the security $\beta_{\texttt{PolyShard}} = \mu N$), we must have $\mu N \leq (N-(K-1)d)/2$. In other words, a node can successfully decode $f^t(u_t(z), u^{t-1}(z))$ only if the number of shards K is upper bounded as $K \leq \frac{(1-2\mu)N}{d}+1$. Based on this constraint, we set the number of shards of the PolyShard scheme, $K_{\texttt{PolyShard}} = \lfloor \frac{(1-2\mu)N}{d} + 1 \rfloor$, which scales linearly with network size N.

The complexity of decoding a length-N Reed-Solomon code at each node is $O(N \log^2 N \log\log N)$, and the total complexity of the decoding step is $O(N^2 \log^2 N \log\log N)$.

Having decoded $f^t(u_t(z), u^{t-1}(z))$, each node evaluates it at $\omega_1, \ldots, \omega_K$ to recover $\{f^t(X_k(t), Y_k^{t-1})\}_{k=1}^K$, to obtain the verification results $\{z_k^t\}_{k=1}^K$.

C. Incremental sub-chain update

The blocks that fail the validation process (i.e, $z_k^t = 0$) should be replaced by the null block in the finalized chain, so we have, $Y_k(t) = z_k^t X_k(t)$. Each node now calculates

$$\tilde{Y}_i(t) = \sum_{k=1}^{K} \ell_{ik} z_k^t X_k(t) = \sum_{k=1}^{K} \ell_{ik} Y_k(t), \tag{10}$$

and append $\tilde{Y}_i(t)$ into its local coded sub-chain to update it to $\tilde{Y}_i^t = (\tilde{Y}_i^{t-1}, \tilde{Y}_i(t))$.

Updating the sub-chains has the same computational complexity with the block encoding step, which is $O(NK)$.

Remark 5. Since the set of coefficients ℓ_{ik}s in (10) are identical to those in (9), appending a coded block to a coded sub-chain is equivalent to appending uncoded blocks to the uncoded sub-chains, and then encoding from the updated sub-chains. This commutativity between sub-chain growth and storage encoding allows each node to update its local sub-chain incrementally by accessing only the newly verified blocks instead of the entire block history.

The total number of operations during the verification and the storage update processes is $O(NK) + Nc(f^t) + O(N^2 \log^2 N \log\log N)$, where the term $O(NK) + O(N^2 \log^2 N \log\log N)$ is the additional coding overhead compared with the uncoded sharding scheme. Since $K_{\texttt{PolyShard}} \leq N$, the coding overhead reduces to $O(N^2 \log^2 N \log\log N)$. The throughput of the PolyShard scheme is

$$\lambda_{\texttt{PolyShard}} = \liminf_{t\to\infty} \frac{K_{\texttt{PolyShard}} N c(f^t)}{N c(f^t) + O(N^2 \log^2 N \log\log N)}. \tag{11}$$

D. Performance analysis of PolyShard

Having demonstrated the robustness of PolyShard against $\mu N = \Theta(N)$ adversaries, we analyze the storage efficiency and the throughput of PolyShard, for the scenario where the verification function f^t has constant degree d, and its computation

complexity $c(f^t) = O(t)$ scales with the chain length t. This is for the verification process that scans through the entire history of the blockchain. For each past block, an operation with fixed complexity is performed (e.g., adding two real vectors for balance checking in Example 1).

For constant degree d, we have $K_{\texttt{PolyShard}} = \lfloor \frac{(1-2\mu)N}{d} + 1 \rfloor = \Theta(N)$, and the storage efficiency $\gamma_{\texttt{PolyShard}} = \Theta(N)$.

When $c(f^t) = O(t)$, i.e., the complexity of computing f^t increases with t, the throughput in (11) becomes

$$\lambda_{\texttt{PolyShard}} = \liminf_{t\to\infty} \frac{K_{\texttt{PolyShard}}}{1 + \frac{O(N \log^2 N \log\log N)}{c(f^t)}} = \Theta(N). \tag{12}$$

This completes the proof of Theorem 1.

We can see that since the complexities of the encoding and decoding operations of `PolyShard` do not scale with t, the coding overhead becomes irrelevant as the chain grows. The `PolyShard` scheme simultaneously achieves optimal scaling on security, storage efficiency, and throughput.

VI. Simulation Results

We perform detailed simulations to assess the performance of `Polyshard` in the payment blockchain system described in Example 1. This system keeps records of all the balance transfers between clients, and verifies new blocks by comparing them with the sum of the previously verified blocks (i.e., computing the verification function in (1)). More specifically, the system contains K shards, each managing M clients. At each time epoch t, one block of transactions is submitted to every shard k. We simulate this system over N nodes using the full replication, uncoded sharding, and `PolyShard` schemes respectively.

We measure the throughput of each scheme under different values of N and t to understand its scalability. Throughput is defined as the number of blocks verified per time unit, and is measured by dividing K (the number of blocks generated per epoch) by the average verification time (to be measured) of the N nodes. For `PolyShard`, the verification time also includes the time each node spent on encoding the blocks. However, since the encoding time is a constant, whilst the balance summation time increases with t as the chain becomes longer, it is expected that the encoding time is becoming negligible.

We note that the storage efficiency and security level of each scheme are decided by system parameters and, thus, do not need measurements.

We simulate this system for $t = 1000$ epochs, using different number of shards $K \in [5, 50]$. Each shard manages $M = 2000$ clients. We fix the ratio $N/K = 3$. Thus, the number of nodes is $N \in [15, 150]$. We plot the complete relation between N, t, and throughput of the three schemes in Fig. 4. For a closer look, we plot the relation between t and throughput when $N = 150$ in Fig. 5, and the relation between N and throughput when $t = 1000$ in Fig. 2 in Section I.

Results and discussions

1) Throughput: As expected, `PolyShard` provides the same throughput as uncoded sharding, which is about K times of the throughput of full replication. From Fig. 5, we observe that the throughput of all three schemes drops as the time progresses. This is because that the computational complexity of verifying a block increases as more blocks are appended to each shard. In terms of scalability, Fig. 2 indicates that the throughput of `PolyShard` and uncoded sharding both increases linearly with the network size N (and K), whilst the throughput of full replication almost stays the same.
2) Storage: It is straightforward to see that `PolyShard` provides the same storage gain over full replication as uncoded sharding, with a factor of K. Thus, `PolyShard` and uncoded sharding are scalable in storage, but full replication is not (Table IIa).
3) Security: As we have analyzed, full replication can tolerate up to 50% of malicious nodes, achieving the maximum security level $\beta_{\text{full}} = \frac{N}{2}$. The error-correcting process of `PolyShard` provides robustness to $\beta_{\texttt{PolyShard}} = \frac{N-K}{2} = \frac{N-N/3}{2} = \frac{N}{3}$ malicious nodes. In contrast, under uncoded sharding, each shard is only managed by 3 nodes. Thus, its security level is only 1 regardless N, which is not scalable (Table IIb).

TABLE II: Storage and security of the three schemes under different network size N.

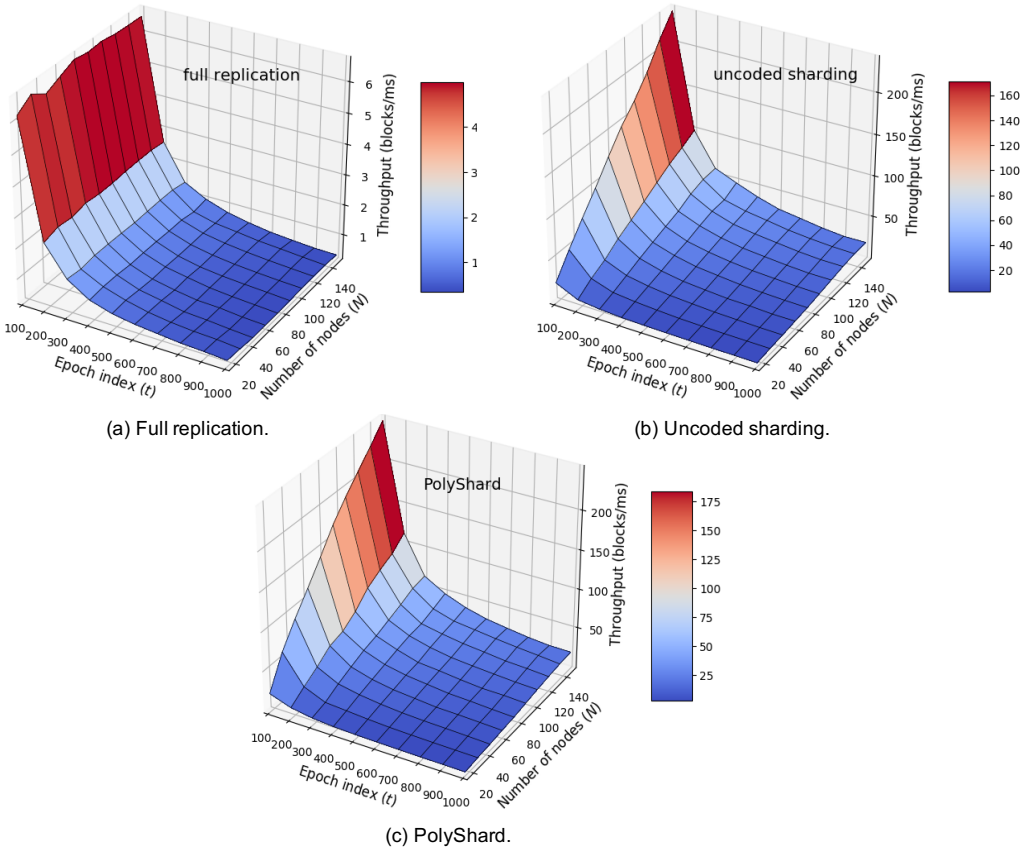
(a) Storage efficiency.

N	15	30	60	90	120	150
γ_{full}	1	1	1	1	1	1
γ_{sharding}	5	10	20	30	40	50
$\gamma_{\texttt{PolyShard}}$	5	10	20	30	40	50

(b) Security.

N	15	30	60	90	120	150
β_{full}	7	15	30	45	60	75
β_{sharding}	1	1	1	1	1	1
$\beta_{\texttt{PolyShard}}$	5	10	20	30	40	50

In summary, `PolyShard` outperforms both full replication and uncoded sharding because it is the only scheme that can simultaneously 1) alleviate the storage load at each node; and 2) boost the verification throughput by scaling out the system, and 3) without sacrificing the safety requirement even when the number of adversaries also grows with network size.



(a) Full replication.

(b) Uncoded sharding.

(c) PolyShard.

Fig. 4: Throughput of the three schemes with respect to time and number of nodes.

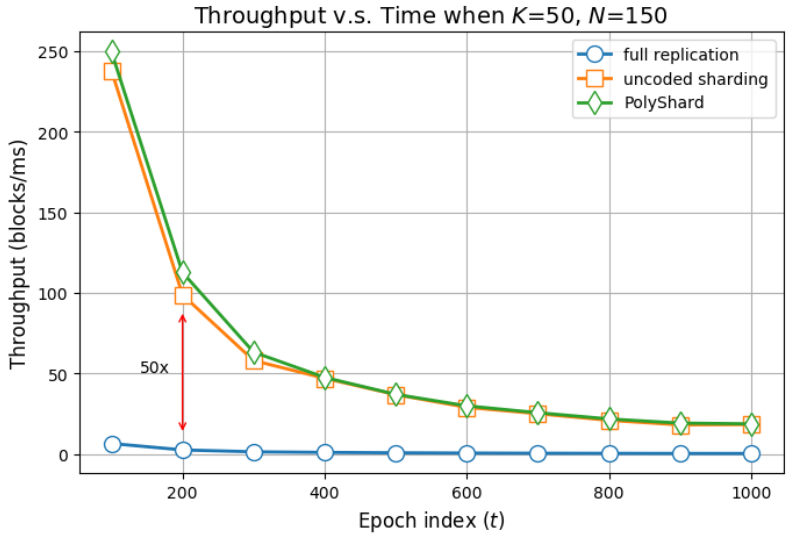


Fig. 5: Throughput of the three schemes when number of nodes N =150.

VII. Discussions

In this section, we discuss how `PolyShard` fits into the overall architecture of a contemporary blockchain system.

Integration into blockchain systems. We note that `Polyshard` has so far been described in a simple setting where each shard produces one block in lock-step. We highlight one instantiation of how `Polyshard` could fit into the architecture of an existing blockchain system, which combines a standard sharding method for proposal followed by `Polyshard` for finalization. The K shards are obtained by assigning users to shards via a user-assignment algorithm. The N nodes are partitioned into K shards using a standard sharding system (see [8]). Inside of the shard, the nodes run a standard blockchain along with a finalization algorithm to get a *locally finalized* version of the block.

Each node is also assigned a coded shard via a coded-shard-assignment algorithm, which assigns a random field element $\alpha_i \in \mathbb{F}$ to a node so that the node can compute which linear combination it will use for coding. We point out here that it is easy to handle churn (users joining and leaving) by this method if the size of the finite field $\mathbb{F}$ is much larger than N - since at this point, the probability of collision (two users getting assigned the same field element) becomes negligible. Thus each node plays a role in both an uncoded shard as well as a coded shard, thus its storage requirement will be doubled; however, our system still has storage efficiency scaling with N. The `Polyshard` algorithm now gets the locally finalized blocks from the different shards at regular intervals and it acts as a global finalization step performing coded validation at the level of the locally finalized blocks. We point out that users requiring high trust should wait for this global finalization stamp before confirming a payment, whereas users requiring short latency can immediately utilize the local-finalization for confirmation.

Beyond the aforementioned issues, there may be cross-shard transactions present in the system, which are payments or smart contracts with inputs and outputs distributed across multiple shards. In such a case, we will use a locking-based method, which locks the payment at the source shard and produces a certificate to the destination shard so that the amount can be spent; this idea has been proposed as well as implemented in `Elastico` [7] and `Omniledger` [8].

Relationship to verifiable computing. An alternative paradigm for accelerating computing in blockchain is verifiable computing [39]–[43], where a single node executes a set of computations (for example, payment validation) and integrity of these computations are then cryptographically certified. A major difference between our framework and verifiable computing is that our scheme is *information-theoretically secure* against a computationally unbounded adversary as against the computational security offered by verifiable-computing schemes. However, verifiable computing schemes can provide zero-knowledge proofs, whereas our scheme does not offer zero-knowledge capability. Finally, verifiable computing is relevant in an asymmetric setting, where one computer is much more powerful than the others, unlike `Polyshard` which is designed for a symmetric setup comprising of equally powerful and decentralized nodes.

Future research directions. Polyshard currently works with polynomials whose degree scales sub-linearly with the number of nodes. An interesting direction of future work is to remove this limitation. In particular, computations that can be represented as *low-depth* arithmetic circuits can be implemented iteratively using low-degree polynomials. Another important direction of future research is the design of validation schemes that can be represented as low-degree polynomials or low-depth arithmetic circuits.

VIII. Acknowledgement

We thank support from the Distributed Technology Research Foundation, Input-Output Hong Kong, the National Science Foundation under grants CCF 1705007, CCF-1763673 and CCF-1703575, and the Army Research Office under grant W911NF1810332. This material is also based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] A. Bahga and V. K. Madisetti, "Blockchain platform for industrial internet of things," *Journal of Software Engineering and Applications*, vol. 9, no. 10, p. 533, 2016.
[3] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. IEEE, 2016, pp. 1–3.
[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.
[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.
[6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol." in *NSDI*, 2016, pp. 45–59.
[7] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.
[8] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger." *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.
[9] A. E. Gencer, R. van Renesse, and E. G. Sirer, "Short paper: Service-oriented sharding for blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 393–401.
[10] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[11] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 3–16.
[12] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded MapReduce," *53rd Allerton Conference*, Sept. 2015.
[13] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, Jan. 2018.
[14] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.
[15] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," *IEEE Workshop on Network Coding and Applications*, Sept. 2016.
[16] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *NIPS*, 2016, pp. 2100–2108.
[17] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *NIPS*, 2017, pp. 4406–4416.

[18] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *NIPS*, 2017, pp. 5440–5448.
[19] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, Aug. 2017, pp. 3368–3376.
[20] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *e-print arXiv:1806.00939*, 2018.
[21] Y. Gao and H. Nobuhara, "A proof of stake sharding protocol for scalable blockchains," *Proceedings of the Asia-Pacific Advanced Network*, vol. 44, pp. 13–16, 2017.
[22] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: A fast blockchain protocol via full sharding," *Cryptology ePrint Archive*, https://eprint.iacr.org/2018/460.pdf.
[23] S. Bano, M. Al-Bassam, and G. Danezis, "The road to scalable blockchain designs," *USENIX; login: magazine*, 2017.
[24] Z. Ren and Z. Erkin, "A scale-out blockchain for value transfer with spontaneous sharding," *e-print arXiv:1801.02531*, 2018.
[25] H. Yoo, J. Yim, and S. Kim, "The blockchain for domain based static sharding," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1689–1692.
[26] S. Cai, N. Yang, and Z. Ming, "A decentralized sharding service network framework with scalability," in *International Conference on Web Services*. Springer, 2018, pp. 151–165.
[27] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 122–128.
[28] M. H. Manshaei, M. Jadliwala, A. Maiti, and M. Fooladgar, "A game-theoretic analysis of shard-based permissionless blockchains," *e-print arXiv:1809.07307*, 2018.
[29] "Ethereum sharding FAQs," https://github.com/ethereum/wiki/wiki/Sharding-FAQs.
[30] A. E. Gencer, R. van Renesse, and E. G. Sirer, "Service-oriented sharding with aspen," *e-print arXiv:1611.06816*, 2016.
[31] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *e-print arXiv:1708.03778*, 2017.
[32] S. Forestier, "Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph," *arXiv preprint arXiv:1803.09029*, 2018.
[33] Z. teacm, "The zilliqa technical whitepaper."
[34] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 444–460.
[35] Y. M. Zou, "Representing boolean functions using polynomials: more can offer less," in *International Symposium on Neural Networks*. Springer, 2011, pp. 290–296.
[36] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.
[37] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.
[38] R. Roth, *Introduction to coding theory*. Cambridge University Press, 2006.
[39] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Annual Cryptology Conference*. Springer, 2010, pp. 465–482.
[40] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM, 2012, pp. 326–349.
[41] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.
[42] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." in *USENIX Security Symposium*, 2014, pp. 781–796.
[43] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptol. ePrint Arch., Tech. Rep*, vol. 46, p. 2018, 2018.

APPENDIX A
FIELD EXTENSION FOR GENERAL BOOLEAN FUNCTIONS

For general blockchain systems that verify incoming blocks based on the most recent P verified blocks, (e.g., a payment system that keeps records of clients' account balances and updates the records every P sets of transactions), we can generally model each incoming block $X_k(t)$, and each verified block $Y_k(j)$ as a binary bit stream of length T, and the verification function $f^t : \{0,1\}^{T(P+1)} \to \{0,1\}$ as a Boolean function that indicates whether $X_k(t)$ is valid or not.

Using the construction of [35, Theorem 2], we can represent any arbitrary Boolean function $f : \{0,1\}^n \to \{0,1\}$ whose inputs are n binary variables as a multivariate polynomial p of degree n as follows. For each vector $\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$, we define $h_{\boldsymbol{a}} = z_1 z_2 \cdots z_n$, where $z_i = x_i$ if $a_i = 1$, and $z_i = y_i$ if $a_i = 0$. Next, we partition $\{0,1\}^n$ into two disjoint subsets S_0 and S_1 as follows.

$$S_0 = \{\boldsymbol{a} \in \{0,1\}^n : f(\boldsymbol{a}) = 0\}, \tag{13}$$

$$S_1 = \{\boldsymbol{a} \in \{0,1\}^n : f(\boldsymbol{a}) = 1\}. \tag{14}$$

The polynomial p is then constructed as

$$f(x_1, \ldots, x_n) = p(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{\boldsymbol{a} \in S_1} h_{\boldsymbol{a}} = 1 + \sum_{\boldsymbol{a} \in S_0} h_{\boldsymbol{a}}, \tag{15}$$

where $y_i = x_i + 1$.

We note that this model applies to verifying the digital signatures in the incoming blocks, where the verification does not depend on past blocks, i.e., $P = 0$. Utilizing the above technique, we can transfer the required non-polynomial computations like inversions and hash functions over some large prime field into polynomial evaluations.

For Boolean verification polynomials over binary field as in (15), the `PolyShard` data encoding (9) does not directly apply since it requires the underlying field size $|\mathbb{F}|$ to be at least the network size N. To use `PolyShard` in this case, we can

embed each element $y_k[i] \in \{0,1\}$ of a verified block Y_k (time index omitted) into a binary extension field $\mathbb{F}_{2^m}$ with $2^m \geq N$. Specifically, the embedding $\bar{y}_k[i] \in \mathbb{F}_{2^m}$ of the element $y_k[i]$ is generated such that

$$\bar{y}_k[i] = \begin{cases} \underbrace{00\cdots0}_{m}, & y_k[i] = 0, \\ \underbrace{00\cdots0}_{m-1}1, & y_k[i] = 1. \end{cases} \tag{16}$$

Then we can select distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_N \in \mathbb{F}_{2^m}$ to apply the encoding strategy in (9) on the block elements in the extension field.

Verification over extension field still generates the correct result. To see that, we can easily verify that the value of the verification polynomial p as constructed in (15) is invariant with the embedding operation in (16). That is, since the polynomial p is the summation of monomials in $\mathbb{F}_2$, when we replace each input bit with its embedding, the value of p equals $\underbrace{00\cdots0}_{m}$ if the verification result is 0, and equals $\underbrace{00\cdots0}_{m-1}1$ if the result is 1.