Breaking Blockchain’s Communication Barrier with Coded Computation

Canran Wang and Netanel Raviv
Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63103.

Abstract—Although blockchain, the supporting technology of various cryptocurrencies, has offered a potentially effective framework for numerous decentralized trust management systems, its performance is still sub-optimal in real-world networks. With limited bandwidth, the communication complexity for nodes to process a block scales with the growing network size and hence becomes the limiting factor of blockchain’s performance. In this paper, we suggest a re-design of existing blockchain systems, which addresses the issue of the communication burden. First, by employing techniques from Coded Computation, our scheme guarantees correct verification of transactions while reducing the communication complexity dramatically such that it grows logarithmically with network size. Second, by adopting techniques from Information Dispersal and State Machine Replication, our design is provably resilient to Byzantine faults under standard cryptographic assumptions.

I. INTRODUCTION

Blockchain is an append-only decentralized system, in which data resides in a chain of blocks that are periodically proposed and agreed upon by a consensus mechanism. Although it is a promising platform for various applications, its performance is sub-optimal due to the limited bandwidth and the scaling communication complexity.

Although numerous attempts have been made to maximize the efficiency of bandwidth usage [3–8], and hence improve the performance of blockchain, a fundamental obstacle remains. That is, every node must receive every transaction. This requirement is paramount to the safety and decentralization of blockchain systems, but unfortunately leads to an inevitable $\Omega(NP)$ bit complexity for a block $B$ containing $P$ transactions to be confirmed, given a network of $N$ nodes.

Sharding [12] is a novel paradigm proposed to address this problem. The network is sliced into multiple communities of a similar sizes, each individually processes a disjoint set of data [13–15]. The constant community size reduces the communication complexity as one transaction is only propagated within one community. Therefore, the system throughput scales with the number of nodes, as additional nodes form extra communities and process additional transactions.

The introduction of coded computation partially alleviates the security problem brought by sharding, i.e., the concentration of adversaries in one community. Li et al. [16] proposed Polyshard, which formulates the verification of transactions as computation tasks, one for each shard, to be solved across all nodes in a distributed manner. Using Lagrange Coded Computing (LCC) [17], in [16] nodes individually compute a polynomial verification function over a coded chain and a coded block; yet, the cross-shard problem (i.e., how to verify transactions from one shard to another), as well as the communication burden, were not addressed.

Ref. [1] propose a novel re-design of coded blockchain which addresses these issues. Specifically, a 2-D coded sharding scheme is introduced to provide simple and inherent support for cross-shard transactions, that normally involves extra communication mechanisms between shards in uncoded blockchains. Further, [1] employs Product-Matrix codes [22] to propagate a block throughout the system; this method alleviates the need for every node to receive every transaction, but requires a synchronous system, and is vulnerable to a discrepancy attack. Furthermore, the discrepancy attack in [9] shows that [16], and coded blockchain schemes in general, are insecure; in the presence of Byzantine faults, nodes might fail to reach a consensus on the uncoded transactions before coded verification. However, reaching such consensus in a straightforward manner requires nodes to first receive all transactions, which nullifies all communication gains.

Our Contributions

In this paper, we adopt the 2-D coded sharding design from [1], and endow it with a drastic reduction of the communication burden, while providing provable security against Byzantine faults, in a realistic communication model. That is, our design inherits the storage and computation gains of ordinary sharding, as well as the resolution of the security issue by using coded sharding. On top of these gains, our design reduces the communication burden with respect to (either coded or uncoded) designs, and resolves the discrepancy issue common in previous coded sharding solutions.

Specifically, by using a combination of a threshold signature scheme [19], a Byzantine-resilient information dispersal method [18], and a Byzantine Fault Tolerant State Machine Replication protocol [11], we achieve the following.

1. Our design requires an overall communication of $O(P \log^2 M \log N)$ bits to process a block with $P$ transactions, where $M$ is the total number of transactions in a shard.

2. Our design allows a leader node to securely distribute coded transactions under the presence of a certain fraction of Byzantine nodes, with $O(N)$ message complexity in the partial synchrony model, thus resolving the discrepancy attack vulnerability suggested in [9].

We emphasize that our design assumes a permissioned systems, and offers mechanisms for communication efficient
and Byzantine resilient transaction verification (that is, no invalid transactions are added to the chain). Additional mechanisms such as transaction confirmation (i.e., reporting which transactions were validated) are left for future work. Due to space constraints the paper has been drastically abbreviated, and we refer readers to the full version [2] for details.

II. BACKGROUND

A. Lagrange Coded Computing

Lagrange Coded Computing [17] (LCC) considers the task of computing a multivariate polynomial \( f(X) \) on each of the \( K \) datasets \( \{X_1,\ldots,X_K\} \). By choosing sets \( \{\alpha_i\}_{i=1}^K \), we define the generator matrix

\[
G_L = \begin{bmatrix}
\Phi_1(\alpha_1) & \ldots & \Phi_1(\alpha_N) \\
\vdots & \ddots & \vdots \\
\Phi_K(\alpha_1) & \ldots & \Phi_K(\alpha_N)
\end{bmatrix}, \Phi_k(z) = \prod_{j \neq k} \frac{z - \omega_j}{\omega_k - \omega_j}.
\]

The coded datasets are \( \tilde{X}_1,\ldots,\tilde{X}_N = (X_1,\ldots,X_K) \), \( G_L \). Every worker node \( i \in [N] \) computes and returns a coded result \( f(\tilde{X}_i) \), from which the leader obtains \( f(X_1),\ldots,f(X_K) \) by decoding.

B. Byzantine Fault Tolerant State Machine Replication

A state machine replication (SMR) [10] protocol formulates a service as a state machine to be replicated in participating nodes. To ensure the consistency of the states, nodes must agree on a total order of execution of client-issued service requests; this provides the safety property of an SMR. The liveness property allows the system to progress, i.e., continuously accepts and executes new requests.

We consider partial synchrony [21] as our network model. In this setting, message delivery is asynchronous (only eventual delivery is guaranteed) until an unknown Global Stabilization Time (GST). After GST, the system is synchronous, where message delay is bounded by a known constant.

Specifically, we employ HotStuff [11], a state-of-the-art Byzantine-Fault Tolerant SMR that incur three communication phases, as our core consensus protocol. We further employ techniques from coded computation and information dispersal (defined next) to reduce communication complexity.

C. Information Dispersal

A distributed storage system employs every node to store a MDS coded fragment of a file. Byzantine faults can cause inconsistency during the dispersal, and correct nodes end up storing fragments that do not correspond to the same file.

Efforts has been made on developing protocols to combat Byzantine faults. AVID-FP (where FP stands for fingerprinting) [18] enables a client to distribute coded fragments of some file \( X \) along with a checksum, i.e., a list of fingerprints of every fragment. The fingerprints, generated by a homomorphic fingerprinting function (defined formally in the sequel) preserves the structure of MDS codes, and allows node \( i \) to verify that the received coded fragment corresponds to a unique file.

D. Digital and Threshold Signature Schemes

We assume a public key infrastructure (PKI) exists among nodes, i.e., every node \( i \) can create a signature \( \langle m \rangle_\sigma \) on a message \( m \) using its private key \( \sigma_i \), verifiable by every other.

In addition, we employ a \((t,n)\)-threshold signature [19] scheme. Every node \( i \) can create a partial signature \( \langle m \rangle_\pi_i \) on \( m \), and a threshold signature \( \langle m \rangle_\pi \) can be produced from a set of partial signatures \( \{\langle m \rangle_\pi_i\}_{i \in I} \) of size \(|I| = t\), but not smaller. Therefore, it is guaranteed that \( m \) has been acknowledged by a \( t \) nodes if their exist a valid signature \( \langle m \rangle_\pi \).

III. CODED COMPUTATION

In this section, we first introduce the general settings and assumptions inherited from [1], and demonstrate the incorporation of Lagrange Coded Computing.

A. The Setting in [1]

The system includes \( N \) nodes and \( K \) client communities of equal size. The nodes are responsible for verifying and storing transactions; clients create transactions and transfer values between each other. Transactions are proposed by clients and verified by nodes periodically during time intervals, called epochs, denoted by a discrete time unit \( t \).

Define the block containing all transactions in epoch \( t \) as \( B(t) = [b_{k,r}] \in \mathbb{F}_q^{K \times R} \) where every \( b_{k,r} \in \mathbb{F}_q \) is a tiny block containing \( Q \) transactions which incur value transfer from community \( k \) to community \( r \); each transaction \( x \in \mathbb{F}_q \) is a vector of length \( R \) over some finite field \( \mathbb{F}_q \). We define \( h_{k,t} = (b_{k,1},\ldots,b_{k,K}) \in \mathbb{F}_q^{K \times R} \) as an outgoing strip containing only transactions with senders in community \( k \), and let \( v_{k,t} = (b_{1,k},\ldots,b_{K,k}) \in \mathbb{F}_q^{K \times R} \) be an incoming strip with transactions having receivers in community \( k \). Moreover, a shard \( \psi_{k,t} = (v_{k,t}^{(1)},\ldots,v_{k,t}^{(D)}) \) is a concatenation of incoming strips associated with community \( k \) from epoch 1 to epoch \( t \).

Further, \( r^{(t)}(h_{k,t},v_{k,t}) \) is a polynomial of degree \( d = O(\log(M)) \), which verifies an outgoing strip against the corresponding shard, and yields an outgoing result

\[
e_{k,t} = (r_{k,1},\ldots,r_{k,K}) \in \mathbb{F}_q^{K \times C}.
\]

Each tiny result block \( r_{k,r} \) contains \( Q \) length-\( C \) verification results; one per every transaction in the tiny block \( b_{k,r} \). A verification result being an all-zero vector implies the validity of the corresponding transaction, and otherwise it is invalid.

B. Coded Computation

The actual verification is conducted in a coded fashion. By assigning a unique scalar \( \omega_k \) to every shard \( k \) and \( \alpha_i \) to every node \( i \), we define the generator matrix \( G_L \) as in Equation (1).

The coded outgoing strip is a linear combination of outgoing strips, i.e., \( \tilde{h}_{i,t} = (G_L)_{i,:} \cdot B(t) \). Similarly \( \tilde{v}_{i,t} = (G_L)_{i,:} \cdot (B(t))_r \), where \( (B(t))_{i,:} \) is the \( i \)-th column of \( G_L \). Equivalently, \( h_{i,t} \) and \( v_{i,t} \) are evaluations of \( \psi_{i,t}(z) = \sum_{k=1}^K h_{k,t} \Phi_k(z) \) and \( \phi_{i,t}(z) = \sum_{k=1}^K v_{k,t} \Phi_k(z) \) at \( \alpha_i \), respectively. Further, node \( i \) stores a coded shard \( \tilde{V}_{i,t} = (\phi_{i,t}(\alpha_1),\ldots,\phi_{i,t}(\alpha_\pi)) \).

745
In epoch $t$, every node $i$ receives the coded strips $\overline{h}_i^{(t)}$ and $\overline{v}_i^{(t)}$ using protocols proposed in Section IV below, and computes the verification function $F^{(t)}$ on $\overline{h}_i^{(t)}$ and the locally stored $\overline{V}_i^t$, and obtains a coded outgoing result $e_i^{(t)} = F^{(t)}(\overline{h}_i^{(t)}, \overline{V}_i^t)$ which is an evaluation of

$$F^{(t)}(z) = F^{(t)}(\psi^{(t)}(z), \phi^{(t)}(z), \ldots, \phi^{(t)}(z))$$

at $\alpha_i$. (3)

Note that the (uncoded) outgoing result is $e_k^{(t)} = F^{(t)}(\omega_k)$.

C. Coded Appending

By exchanging coded outgoing result, nodes obtain an indicator vector $g \in \{0, 1\}^{QK}$ (see Section IV-D). Recall that every coded transaction is a linear combination of $K$ transactions. The corresponding entry of $g$ equals to 0 if all these $K$ transactions are valid, and equals to 1 if at least one is invalid. The appending operation of node $i$ is instructed by $g$: after the coded verification, each node $i$ appends the coded incoming strip $\overline{v}_i^{(t)}$ to their coded shard, after setting to zero the coded transactions whose corresponding entry of $g$ is 1.

IV. Coded Consensus

Our scheme tolerates these Byzantine nodes in the partial synchrony model, given that $N \geq (K - 1)d + 3f + 1$, by guaranteeing consistency, homology, and validity, defined next.

First, our design must guarantee consistency, i.e., at every epoch $t$, correct nodes must perform coded verification on coded outgoing strips generated from the same block $B^{(t)}$.

**Condition 1 (Consistency).** Every correct node $i$ obtains $\overline{h}_i$, defined as $\overline{h}_i = (G_{\mathcal{L}})^T \cdot B^{(t)}$, where $(G_{\mathcal{L}})_i$ denotes the $i$-th column of the generator matrix $G_{\mathcal{L}}$.

This condition imposes that correct nodes obtain coded outgoing strips that are consistent with each other, i.e., correspond to the same block $B^{(t)}$. Otherwise, correct verification is impossible, as suggested in [9]. Moreover, our design must maintain homology.

**Condition 2 (Homology).** Every correct node $i$ obtains $\overline{v}_i$, defined as $\overline{v}_i = (G_{\mathcal{L}})^T \cdot B^{(t)}$, where both $B^{(t)}$ and $G_{\mathcal{L}}$ are as in Condition 1.

The second condition suggests that every correct node obtains the coded incoming strip that is homologous to the coded outgoing strip, i.e., generated from the same block $B^{(t)}$. Otherwise, we say they are nonhomologous; such nonhomology can cause a discrepancy between the verified and the appended, i.e., nodes verify valid transactions, but append invalid ones. Lastly, nodes must not store invalid transactions.

**Condition 3 (Validity).** Every correct node $i$ appends the coded incoming strip $\overline{v}_i^{(t)}$ to its local coded chain after setting the invalid coded transactions to zero, i.e. coded transactions which were not formed exclusively from valid transactions.

This condition requires every node $i$ to obtain the indicator vector $g$ defined in Section III-C. Together, we say that a protocol provides coded consensus if it simultaneously achieves the above three conditions. We propose such a protocol, employing a leader to distribute coded strips and provide coded consensus. Our approach adopts HotStuff [11], a BFT SMR with linear message complexity, and techniques from Information Dispersal for consistency and homology. In addition, we employ coded computation that maintains validity of the system. Further, the superscript $(t)$ is omitted for brevity.

A. Overview

In order to maintain the aforementioned three properties, we employ HotStuff to maintain a chain of headers, each corresponds to a block. HotStuff provides the safety and liveness property of the header chain. Together with information dispersal techniques, our scheme maintains the consistency property. We provide detailed discussion in Section IV-B. Further, we incorporate extra mechanisms in HotStuff to maintain homology (Section IV-C) and validity (Section IV-D). In Section V, we show that our scheme indeed provides coded consensus, and inherits the liveness property from HotStuff.

B. Maintaining Consistency (Condition 1)

Our method depends on a data structure called checksum, introduced by AVID-FP [18]. A checksum allows nodes to verify that the received coded strip is consistent with ones received by others, i.e., computed from the same block. It contains a list of $K$ fingerprints, each generated from an (uncoded) outgoing strip (a row of $B$), using some $\varepsilon$-fingerprinting function $fp$ defined as follows.

**Definition 1.** [18, Def. 2.1, Def. 2.5] A function $fp : T \times \mathbb{P}^q \rightarrow \mathbb{P}^q$ is an $\varepsilon$-fingerprinting function if

$$\max_{d, d' \in \mathbb{P}^q, d \neq d'} \Pr_{T} \left[ fp(r, d) = fp(r, d') \right] \leq \varepsilon.$$

That is, the probability for two distinct $d, d' \in \mathbb{P}^q$ to have the same fingerprint is at most $\varepsilon$, where the key $r$ is chosen uniformly at random from some input space $T$. A fingerprinting function $fp : T \times \mathbb{P}^q \rightarrow \mathbb{P}^q$ is homomorphic if $fp(r, d) + fp(r, d') = fp(r, d + d')$ and $b \cdot fp(r, d) = fp(r, b \cdot d)$ for any $r \in T$, any $b \in \mathbb{P}^q$, and any $d, d' \in \mathbb{P}^q$.

We let $fp : T \times \mathbb{P}^q \rightarrow \mathbb{P}^q$ be an $\varepsilon$-fingerprinting function where $\delta = \frac{2q}{K}$ is the size of a strip. As done in AVID-FP [18], the random selection of $r$ from $T$ is simulated by deterministic cryptographic hash functions [20]; referring to the use of a hash function as a random oracle is a common practice in blockchain systems, e.g., in [13]. In addition to the $K$ fingerprints, a list of hash values $cc = [hash(\overline{h}_1), \ldots, hash(\overline{h}_N)]$ is included in the checksum, generated using a cryptographic hash function $hash : \mathbb{P}^q \rightarrow \mathbb{P}^q$ (not to be confused with hash1 and hash2 mentioned earlier). The selection of $r$ is achieved by another cryptographic hash function $select : (\mathbb{P}^q)^N \rightarrow T$, which takes the list $cc$ as input and outputs an element in $T$.  

This condition requires every node $i$ to obtain the indicator vector $g$ defined in Section III-C. Together, we say that a protocol provides coded consensus if it simultaneously achieves the above three conditions. We propose such a protocol, employing a leader to distribute coded strips and provide coded consensus. Our approach adopts HotStuff [11], a BFT SMR with linear message complexity, and techniques from Information Dispersal for consistency and homology. In addition, we employ coded computation that maintains validity of the system. Further, the superscript $(t)$ is omitted for brevity.
The leader first generates a header $=(cksH, cksV)$, where $cksH$ is the checksum generated from the (uncoded) outgoing strips and $cksV$ is the checksum generated from the (uncoded) incoming strips. In order to verify that the a coded strip agrees with the checksum, i.e., $cksH$ (resp. $cksV$) and $\hat{h}_i$ (resp. $\hat{v}_j$) are computed from the same $B$, we require the fingerprinting function to be homomorphic.

This property enables the node to verify that the coded fragment satisfies the required linear combination (defined by the generator matrix) with uncoded strips, by having access only to the fingerprints of uncoded strips, and not to the uncoded strips. As any coded strip is a linear combination of $K$ uncoded strips, the homomorphism guarantees that its fingerprint must be equal to the same linear combination of $K$ fingerprints of uncoded strips.

Note that $h_k$ is the $k$-th row of the matrix $B$, and $\hat{h}_i$ is the $i$-th row of the matrix $B=(G_L)^\top B$. Similarly, $v_k$ is the $k$-th row of the matrix $B^\top$, and $\hat{v}_i$ is the $i$-th row of the matrix $B^\top=(G_L)^\top B^\top$. Thus, each node can confirm the agreement between the received (coded) strip and the checksum, as long as the fingerprints of the coded strip match the encoding of the $K$ fingerprints in the checksum: this is guaranteed with high probability, as shown [18, Theorem 3.4].

In this respect, each node can assure that the received coded strip is consistent with ones received by others by reaching a consensus on the checksum which is in agreement with all coded strips. Treating checksums as requests, we can employ BFT SMR protocols that allow correct nodes to reach a consensus on the total order of them, and hence maintain the consistency of strips at any epoch $t$. Specifically, we adopt HotStuff [11], a leader-based BFT SMR protocol that works in partial synchrony.

HotStuff allows nodes to reach a consensus on a chain of headers. HotStuff always ensures safety given bounded number of faulty nodes ($N \geq 3f+1$). That is, no two correct nodes should accept conflicting headers; by conflicting we mean the chain led by either one extends the chain led by the other. Hence, correct nodes will never accept different headers at any epoch $t$. When the system becomes synchronous, HotStuff provides the liveness property, such that the consensus on headers will be reached when the leader is correct. As discussed earlier, such a consensus on a header guarantees the consistency of coded fragment generated from the corresponding block.

HotStuff runs in consecutive views associated with increasing integer view numbers. In each view there is a designated node LEADER(viewNumber) that proposes new headers and distribute coded strips. In order to append a header to the chain, the leader must collect partial signatures on its proposal from a quorum of nodes in each of three phases, namely PREPARE, PRE-COMMIT and COMMIT. The partial signatures are generated with a $(N-f,N)$-threshold signature scheme $\pi$.

In the PREPARE phase, the leader broadcasts the header to every node $i$ piggybacked with the corresponding coded strips $h_i$ and $\tilde{v}_i$ in the PREPARE message. Every node $i$ acknowledge the message if the checksums in the header agrees with the coded strips. After the PRE-COMMIT and COMMIT phase, node $i$ is assured that the consensus on the header is reached, and the consistency of coded strips is maintained.

C. Maintaining Homology (Condition 2)

Although [11] allows nodes to be consistent on the chain of headers, the homology problem remains. With a Byzantine leader, even though a correct node may obtain a consistent coded outgoing strip $h_i=\langle G_L \rangle^\top B$ and consistent coded incoming strip $\tilde{v}_i=\langle G_L \rangle^\top (B^\top)^\top$, they might correspond to different blocks $B \neq B'$. To maintain homology, we integrate the following design with Hotstuff’s three-phase protocol.

After verifying the consistency of coded strips, node $i$ multiplies the coded outgoing strip $\langle G_L \rangle^\top B$ from the right with $G_L$, creating a length-$N$ vector $w_{i,\ast}$, which equals to the $i$-th row of $W=\langle G_L \rangle^\top B G_L$. Similarly, it creates a vector $u_{i,\ast}=\langle G_L \rangle^\top (B^\top)^\top G_L$ as the $i$-th row of $U=\langle G_L \rangle^\top (B^\top)^\top G_L$. Node $i$ then defines a length-$N$ signature vector $px_i$, whose $j$-th entry stores its digital signature (not to be confused with partial signature) on $\langle w_{i,j},j \rangle$. Formally,

$$px_i[j]=\langle w_{i,j},j \rangle_{\sigma_i}, \text{ for } j \in [N].$$

Node $i$ sends back $px_i$ in the acknowledgement of the PREPARE message received from the leader. After collecting such vectors from nodes in a quorum $I$ of size $|I|=N-f$, the leader stacks the $px_i$’s in an $(N-f) \times N$ matrix ordered by the indices of nodes. It sends the $j$-th column of the resulting matrix to every node $j \in [N]$ in the PRE-COMMIT message together with a quorum identifier $QI(I)$ that specifies the members of $I$.

Upon receiving the PRE-COMMIT message from the leader, node $j$ learns the members of $I$ from the quorum identifier $QI(I)$. For every $i \in I$, node $j$ verifies if the received $\langle w_{i,j},j \rangle_{\sigma_i}$ is a valid signature on $\langle u_{j,i},j \rangle$. If the verification passes, node $j$ creates partial signature $\langle QI(I) \rangle_{\sigma_j}$ on the quorum identifier and sends it back to the leader as an acknowledgement of the PRE-COMMIT message.

Upon receiving acknowledgements from a quorum $J$ of nodes, the leader combines theses partial signatures and broadcasts a COMMIT message with a valid signature $\langle QI(I) \rangle_{\sigma_{\pi}}$ on the quorum identifier and sends it to the leader as an acknowledgement of the PRE-COMMIT message.

D. Maintaining Validity (Condition 3)

So far, we have developed mechanisms that maintain consistency and homology such that every correct node $i$ is performing verification on the coded outgoing strip $h_i=\langle G_L \rangle^\top B$ and appending the coded incoming strip $\tilde{v}_i=\langle G_L \rangle^\top (B^\top)^\top$. Finally, we employ coded computation to guarantee validity, such that no invalid transactions can be appended to the blockchain. Specifically, we show how nodes can securely obtain the correct indicator vector $g$ used to instruct coded appending.

Recall that the degree of the polynomial verification function $F(z)$ is $(K-1)d$, and hence it is uniquely defined by evaluations at any $L=(K-1)d+1$ distinct points. For any
distinct $\beta_1, \ldots, \beta_L$, one can represent $F(z)$ as a linear combination of Lagrange basis polynomials $\Psi_1(z), \ldots, \Psi_L(z)$, i.e.,

$$F(z) = \sum_{\ell \in [L]} F(\beta_{\ell}) \Psi_{\ell}(z), \quad \Psi_{\ell}(z) = \prod_{j \in [L] \setminus \{\ell\}} \frac{z - \beta_j}{\beta_{\ell} - \beta_j}.$$ 

This implies the following representations of the coded outgoing result $F(\alpha_1), \ldots, F(\alpha_N)$ and the (uncoded) outgoing result $F(\omega_1), \ldots, F(\omega_N)$, as

$$
\begin{bmatrix}
F(\alpha_1) \\
\vdots \\
F(\alpha_N)
\end{bmatrix} = G_{F,\alpha}^T 
\begin{bmatrix}
F(\beta_1) \\
\vdots \\
F(\beta_L)
\end{bmatrix},
$$

and

$$
\begin{bmatrix}
F(\omega_1) \\
\vdots \\
F(\omega_K)
\end{bmatrix} = G_{F,\omega}^T 
\begin{bmatrix}
F(\beta_1) \\
\vdots \\
F(\beta_L)
\end{bmatrix},
$$

where $G_{F,\alpha}, G_{F,\omega}$ are the appropriate Lagrange matrices.

Upon receiving the prepare message, node $i$ computes the verification function $F$ on the received coded outgoing strip $\hat{h}_i$ and coded shard $\hat{v}_i$, obtaining its coded outgoing result $\hat{e}_i = F(\alpha_i)$. Node $i$ multiplies $\hat{e}_i$ from the right with the Lagrange matrix $G_L \in \mathbb{F}_q^{K \times N}$, obtaining its coded outgoing result $e_i = (\hat{e}_i, \ldots, \hat{e}_i) \cdot G_L$, which is the $i$-th row of the matrix $C = G_{F,\alpha}^T \cdot [F(\beta_1)^T, \ldots, F(\beta_L)^T]^T \cdot G_L$. Node $i$ sends $e_i$ back to the leader in acknowledging the prepare message, with its signature on each of the $N$ entries.

Upon receiving a quorum of $N - f$ such vectors, the leader stacks them on top of each other to form an $(N - f) \times N$ matrix. The leader sends the $j$-th column of the resulting matrix, which is the encoding of the $j$-th column of the matrix $[F(\beta_1)^T, \ldots, F(\beta_L)^T]^T \cdot G_L$ using the generator matrix $G_{F,\alpha}$, to every node $j \in [N]$ in the pre-commit message.

After verifying the signature of each entry, nodes perform Reed-Solomon decoding, and obtains the $j$-th column of the matrix $G_{F,\omega}^T \cdot [F(\beta_1)^T, \ldots, F(\beta_L)^T]^T \cdot G_L$ by left multiplying the decoded column with $G_{F,\omega}^T$. By Equation (5), this is the $j$-th column of $[\hat{e}_1^T, \ldots, \hat{e}_K^T]^T \cdot G_L$, which further equals to $\hat{s}_i = (s_1^T, \ldots, s_K^T) \cdot G_L$, where $s_k = (b_{1,k}, \ldots, b_{K,k})$.

Recall that the tiny result block $r_{k,r}$ stores the verification results of the $Q$ transactions in the tiny block $b_{k,r}$, and each is a length-$C$ vector (see Section III-A). For $\ell \in [QK]$, the $\ell$-th entry of $\hat{s}_i$ is a linear combination of verification results of the $\ell$-th transaction in each of $v_1, \ldots, v_K$. Hence, the entries with index $\ell$ from all $\hat{s}_1, \ldots, \hat{s}_N$ form a codeword of a $[N, K]$ MDS code, and hence contains either all zero vectors, or at most $K - 1$ zero vectors by the MDS property.

Every node $i$ creates a binary results vector $g_{s,i} \in \{0, 1\}^{QK}$. Each entry of $g_{s,i}$ is associated with an entry of $\hat{s}_i$; it equals to 0 if the associated entry is a zero vector, and equals to 1 otherwise. Let $G \in \{0, 1\}^{QK \times N}$ be the concatenation of all $g_{s,i}$‘s. Clearly, the indicator vector $g$ equals to the reduction of all columns of $G$ with bitwise OR.

Let $\lambda$ be a $(K + f, N)$ threshold signature scheme, and let $\tau$ be a $(f + 1, N)$ threshold signature scheme. Node $i$ creates a partial indicator vector $gw_i$, such that for every $\ell \in [QK]$,

$$gw_i[\ell] = \begin{cases} 
\langle \ell, 0, header \rangle_{\lambda \ell} & g_{t,i} = 0 \\
\langle \ell, 1, header \rangle_{\tau \ell} & g_{t,i} = 1 
\end{cases}.$$
REFERENCES

[1] C. Wang and N. Raviv, “Low Latency Cross-Shard Transactions in Coded Blockchain,” in *IEEE International Symposium on Information Theory*, pp. 2678–2683, 2021.

[2] C. Wang and N. Raviv, “Breaking Blockchain’s Communication Barrier with Coded Computation,” *Journal on Selected Areas in Information Theory*, 2022.

[3] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, “Bitcoin-NG: A scalable blockchain protocol,” *USENIX Security Symposium*, pp. 279–296, 2016.

[4] R. Pass and E. Shi, “Hybrid Consensus: Efficient consensus in the permissionless model,” *International Symposium on Distributed Computing*, 2017.

[5] 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”, *USENIX Security Symposium*, pp. 279–296, 2016.

[6] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” *Annu. Int. Cryptology Conference*, pp. 357–388, 2017.

[7] P. Daian, R. Pass, and E. Shi, “Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake,” in *International Conference on Financial Cryptography and Data Security*, 2019, pp. 23–41.

[8] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling Byzantine Agreements for Cryptocurrencies”, *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 51–68, 2017.

[9] N. A. Khooshemehr and M. A. Maddah-Ali, “The Discrepancy Attack on Polyshard-ed Blockchains,” in *IEEE Int. Symp. Information Theory*, pp. 2672–2677, 2021.

[10] F. B. Schneider, “Implementing fault-tolerant services using the state machine approach: A tutorial,” *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, 1990.

[11] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, “HotStuff: BFT consensus with linearity and responsiveness,” *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 347–356, 2019.

[12] G. Wang, Z. J. Shi, M. Nixon, and S. Han, “Sok: Sharding on blockchain,” *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pp. 41–61, 2019.

[13] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” *ACM SIGSAC Conference on Computer and Communications Security*, pp. 17–30, 2016.

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

[15] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: Scaling blockchain via full sharding,” *ACM SIGSAC Conference on Computer and Communications Security*, pp. 931–948, 2018.

[16] S. Li, M. Yu, C.-S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, “Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 249–261, 2020.

[17] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, “Lagrange coded computing: Optimal design for resiliency, security, and privacy,” in *International Conference on Artificial Intelligence and Statistics*, pp. 1215–1225, 2019.

[18] J. Hendricks, G.R. Ganger, and M. K. Reiter, “Verifying distributed erasure-coded data,” *Proceedings of ACM symposium on Principles of distributed computing*, pp. 139–146, 2007.

[19] A. Boldyreva, “Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme,” in *International Workshop on Public Key Cryptography*, pp. 31–46, 2003.

[20] M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm for designing efficient protocols,” in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 62–73,1993.

[21] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, “Consensus in the presence of partial synchrony,” *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[22] 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.