Coding for Scalable Blockchains via Dynamic Distributed Storage

Ravi Kiran Raman and Lav R. Varshney, *Senior Member, IEEE*

***Abstract*— Blockchains store transaction data in the form of a distributed ledger where each node in the network stores a current copy of the sequence of transactions as a hash chain. This requirement of storing the entire ledger incurs a high storage cost that grows undesirably large for high transaction rates and large networks. In this work we use secret key sharing, private key encryption, and distributed storage to design a coding scheme such that each node stores only a part of each transaction, thereby reducing the cold storage cost to a fraction of its original cost. In addition, the storage code ensures the security of the storage from active adversaries that may aim to corrupt prior transactions by altering copies of the ledger. We further employ a dynamic zone allocation algorithm that spreads the node allocation and data distribution across transactions. Under this coding scheme we show that we can also improve the integrity of the transaction data in the network over current schemes.**

***Index Terms*— Blockchains, distributed storage, secret sharing.**

I. Introduction

BLOCKCHAINS are distributed, shared ledgers of transactions that reduce friction in financial networks due to disparate intermediaries using different technology infrastructures, and even reduce the need for intermediaries to validate financial transactions. This has in turn led to the emergence of a new environment of business transactions and self-regulated cryptocurrencies such as bitcoin [3], [4]. Due to such favorable properties, blockchains are being adopted extensively outside cryptocurrencies in a variety of novel application domains [5], [6] such as medicine [7], supply chain management and global trade [8], and government services [9]. Blockchains are expected to revolutionize the way financial/business transactions are done [10], for instance in the form of smart contracts [11].

The blockchain architecture however comes with an inherent weakness. Blockchain works on the premise that the entire ledger of transactions is stored as a hash chain at every node, even though the transactions themselves are meaningless to the peers that are not party to the underlying transaction. Consequently, the individual nodes incur a significant, ever-increasing storage cost [12]. Note that *secure* storage may be much more costly than just raw hard drives, e.g. due to infrastructure and staffing costs. In current practice, the most common technique to reduce the storage overload is to prune old transactions. However, this is not sustainable for blockchains with a high transaction rate. Another approach is to use light clients that only store headers in ledgers, but this is often insufficiently secure. In general, heavy storage requirements limit the scaling of blockchains.

For instance, consider the bitcoin network. Bitcoin currently serves on average 4.6 transactions per second [13]. This low number is due to a variety of reasons including the economics involved in maintaining a high value for the bitcoin. However, even at this rate, this accounts for an average of 160 MB of storage per day [12], i.e., about 60 GB per year. With increasing usage, the number of transactions and hence the size of the ledger grows exponentially, emphasizing the need for better storage techniques that can meet the surging demand.

SETL, an institutional payment and settlement infrastructure based on blockchain, crossed the 1 billion transactions per day mark in 2015 [14]. However, this is dwarfed by the 14 *trillion* transactions processed every day by the Federal Reserve. If cryptocurrencies are to become financial mainstays, they would need to scale by several orders. Specifically, they will eventually need to support to the order of about 2000 transactions per second on average [13]. Note that this translates to an average storage cost of over 90GB per day. This is to say nothing about uses for blockchain in global trade and commerce, healthcare, food and agriculture, and a wide variety of other industries, where permissioned blockchains are more prominent than permissionless ones. With storage cost expected to saturate soon due to the ending of Moore's Law, storage is emerging as a pressing concern with the large-scale adoption of blockchain.

In this work we focus on archival *cold storage* of blockchain ledgers. Specifically, by cold storage we refer to the record of completed transactions that are infrequently accessed on the blockchain. For instance consider a blockchain for land ownership and transfer records (including satellite imagery and maps), e.g. in developing countries where individual transactions are not accessed too frequently [15], [16]. Yet, the scale of such blockchain systems still results in a large ledger. It is important to securely hold such infrequently accessed transactions in cold storage to preserve ownership records. Conventional methods such as pruning the ledgers would not be appropriate in such contexts, and so we develop a distributed storage code for cold storage.

By separating transactions on the ledger into active and cold storage, the blockchain may function efficiently on new

Manuscript received February 5, 2020; revised February 25, 2021; accepted June 30, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Kalyvianaki. Parts of the work were presented at [1], [2]. *(Corresponding author: Ravi Kiran Raman.)*

Ravi Kiran Raman is with Analog Devices Inc., Boston, MA 02110 USA (e-mail: ravikiranraman10@gmail.com).

Lav R. Varshney is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois Urbana-Champaign, Urbana, IL 61801 USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3098613, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3098613

transactions which are incorporated into the ledger, and yet not lose older transactions that are still valuable to the peers of the network. Reducing storage costs also enables more nodes in the network to be full nodes, thereby democratizing the network and preventing the concentration of power.

Since we focus on archival ledger storage at nodes, we do not explicitly consider consensus protocol details, whether approaches like proof-of-work in prominent permissionless blockchains [4] or practical Byzantine fault tolerance (PBFT) in prominent permissioned blockchains [17]. Moreover, we do not explicitly consider the economics of cryptocurencies through the mining process or the energy consumption costs associated with them. Instead we focus on data integrity and storage costs associated with maintaining the blockchain.

A. Our Contributions

In this paper we show that the storage costs in large-scale blockchain networks can be reduced through secure distributed storage codes. Specifically, when each peer stores a piece of the entire transaction instead of the entire block, the storage cost can be diminished to a fraction of the original. In Sec. IV, we use a novel combination of Shamir's secret sharing scheme [18], private key encryption, and distributed storage codes [19], inspired by [20], to construct a coding scheme that distributes transaction data among subsets of peers. The scheme is shown to be optimal in storage up to small additive terms. We also show, using a combination of statistical and cryptographic security of the code, that the distributed storage loses an arbitrarily small amount of data integrity as compared to the conventional method.

Further, in Sec. VI, using a dynamic zone allocation strategy among peers, we show that the data integrity can further be enhanced in the blockchain. We formulate the zone allocation problem combinatorially as decomposing complete hypergraphs into 1-factors. We design an allocation strategy that is order optimal in the time taken to ensure highest data integrity. We show that given enough transaction blocks to follow, such a system is secure from active adversaries. In particular, we strongly characterize these bounds and their scaling with parameters of the code by considering some example instances.

The enhanced integrity and reduced storage do come at the expense of increased recovery and repair costs. In particular, as the data is stored as a secure distributed code, the recovery process is highly sensitive to Denial of Service (DoS) attacks and the codes are not locally repairable, as discussed in Sec. VII. Blockchain-based data storage applications, including with data insurance, are specifically considered in Sec. VIII.

Before getting into the construction and analysis of the codes, we first introduce a mathematical abstraction of the blockchain in Sec. II, and give a brief introduction to the coding schemes that are used in this work in Sec. III.

B. Related Work

Distributed storage schemes have been considered in the past in the form of information dispersal algorithms (IDA) [21], [22] and in the form of distributed storage codes [19], [23]. In particular [22] considers an information dispersal scheme that is secure from adaptive adversaries. We note that the coding scheme we define here is stronger than such methods as it handles active adversaries. Secure distributed storage codes with repair capabilities to protect against colluding eavesdroppers [24] and active adversaries [25] have also been considered.

Two main types of adversaries are of interest in blockchain systems. The first is a DoS adversary that prevents recovery of stored data from specific nodes of the network, analogous to the classical distributed storage framework. The other type of adversary is an active adversary that aims to corrupt a recorded transaction in the blockchain ledger into something different that is validated by the network. Note that this adversary is different from the typical adversary considered in distributed storage that aims to make a recorded transaction *erroneous*. The difference in the nature of attacks by adversaries calls for the new coding scheme in this paper.

Coding approaches have recently been proposed within several blockchain settings. Coded Merkle trees have been developed to address data availability attacks on blockchains [26]. Building on the idea of sharding in blockchains, polynomial coded sharding has been proposed for efficient *hot storage* as part of consensus protocols rather than *cold storage* as part of archiving as we consider here [27]. In considering archiving from a distributed coding viewpoint, like we do here, an approach based on rateless codes has been proposed [28]. Alternative algebraic code constructions within our systems-level framework have also been developed [29].

II. System Model

We now abstract blockchain systems by a mathematical model for the peer network and hash chain. The model described here is mainly based on the Hyperledger Fabric [17]. Blockchain systems have a variety of endorsement mechanisms that invoke different numbers of validations from network peers to add blocks to the chain. However, for ease of description of this work on cold storage of blockchain ledgers, we presume a simple majority vote-based endorsement scheme. Many of the claims of security and storage can be extended to other endorsement schemes, *mutatis mutandis*.

A. Ledger Construction

The blockchain comprises a connected peer-to-peer network of nodes, where nodes are functionally characterized as:

1) **Clients:** nodes that invoke or are involved in a transaction, have the blocks validated by endorsers, and communicate them to the orderers.
2) **Peers:** nodes that commit transactions and maintain a current version of the ledger. Peers may also adopt endorser roles.
3) **Endorsers:** for a given data block, a peer acts as an endorser if it validates the contents of the transaction prior to its inclusion in the blockchain ledger.
4) **Orderer:** nodes that communicate the transactions to the peers in chronological order to ensure consistency of the hash chain.

Note that the classification is only based on function, and individual nodes in the network can serve multiple roles.

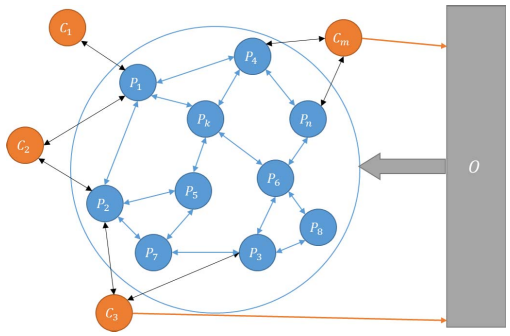

Fig. 1. Architecture of a blockchain network: Here the network is categorized by functional role into clients C_i, peers P_i, and orderers O. As mentioned earlier, the clients initialize transactions. Upon validation by the endorsers of this transaction, the transactions are communicated to peers by orderers. The peers maintain an ordered copy of the ledger of transactions.

The distributed ledger of the blockchain maintains a current copy of the sequence of transactions. A transaction is initiated by the participating clients and is verified by endorsers. Subsequently, the verified transaction is communicated to the orderer, who then broadcasts them to peers to store in the ledger. The nodes in the blockchain are as depicted in Fig. 1. Here nodes C_i are clients, P_i are peers, and O is the set of orderers in the system, categorized by function.

The transaction and the nature of the data associated with it is application-specific such as proof of fund transfer across clients in bitcoin-like cryptocurrency networks, smart contracts in business applications, patient diagnoses/records in medical record storage, and raw data in cloud storage. We use the term *transaction* broadly to represent all such data. A transaction is initiated by participating clients, verified by endorsers (select peers), and broadcast to peers through orderers.

The blockchain ledger stores the sequence of transactions as a (cryptographic) *hash chain*. Hash chains are constructed using hash functions that are defined as follows.

Definition 1: Let $\mathcal{M}$ be a set of messages of arbitrary lengths, $\mathcal{H}$ the set of (fixed-length) *hash values*. A *cryptographic hash function* family is a function $h : \mathcal{I} \times \mathcal{M} \to \mathcal{H}$, where $\mathcal{I}$ is the set of parameters that dictate the deterministic map that is employed.

Good hash functions satisfy salient properties [30] such as

1) **Computational ease:** Hash values are easy to compute.
2) **Pre-image resistance:** Given $H \in \mathcal{H}$, it is computationally infeasible to find $M \in \mathcal{M}$ such that $h(\mathcal{M}) = H$. To be precise, given a randomized and computationally limited adversary who samples the message $M' = A(H, I)$, we consider the pre-image resistance in terms of the hitting probability

$$P_{pre-image} = \mathbb{P}\left[h(I, A(H, I)) = H\right]. \tag{1}$$

3) **Collision resistance:** It is computationally infeasible to find $M_1, M_2 \in \mathcal{M}$ such that $h(M_1) = h(M_2)$. Again, to be precise, given a randomized and computationally limited adversary who sample messages $(M, M') = A(I)$, we consider collision resistance in terms of the hitting probability

$$P_{collision} = \mathbb{P}\left[\{M \neq M'\} \cap \{h(I, M) = h(I, M')\}\right]. \tag{2}$$

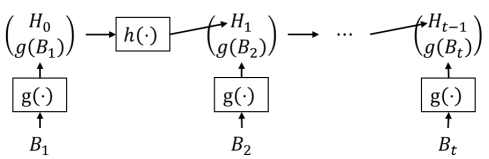

Fig. 2. Hash chain structure for the ledger. The chain is constructed by hashing a hash value of the data for easier recovery and consistency verification.

A hash chain is a sequence of data blocks such that each block includes a header pointing to the previous block in the form of the hash value of the previous (header included) block. To be precise, if the transaction data at time t is B_t, and its header is H_{t-1}, then the header for the block at time $t+1$ is $H_t = h(I_t, (H_{t-1}, B_t))$. Here $h(\cdot)$ is the hash function. Thus, the hash chain is stored as

$$(H_0, \mathbf{B}_1) - (H_1, \mathbf{B}_2) - \cdots - (H_{t-1}, \mathbf{B}_t).$$

Here, we presume the following operations are executed by the nodes, depending on their role in the transaction:

- WRITE(B): initiate a block B of data that include transaction data which are verified and appended to the blockchain ledger.
- READ(t): call with index t to recover block B_t from the blockchain ledger.

The endorsers perform the following operations:

- VERIFY(B): check the details of the transaction and verify authenticity.
- MINE(B, t): recover hash value H_{t-1} and use B to compute hash H_t and report to orderer to include block in blockchain ledger.

The operations of the orderer are:

- VALIDATE(B, H, t): validate block and hash value reported by endorser.
- APPEND(B, H, t): encode data and hash, and communicate to peers, to append block at index t in the ledger.

More complicated data structures are used in Bitcoin-type blockchains such as the Merkle tree, with the hash chain constructed using the Merkle root as the data in the block [13]. We consider a simple form of this as shown in Fig. 2. Let $\mathbf{B}_t$ be the data block corresponding to the t-th transaction. Let g, h be two hash functions. Let $W_t = (H_{t-1}, g(\mathbf{B}_t))$ be the concatenation of the previous hash and a hash of the current data. Then, $H_t = h(I_t, W_t)$ is the hash value stored with the $(t+1)$-th block, where the index I_t is sampled uniformly.

Using such a hashed form to construct the chain simplifies consistency verification and reduces recovery costs, while retaining all the salient features of the hash chain that directly hashed block values would have. In more general forms, the data block can be replaced by a Merkle tree structure with the results extending directly.

For all t, let $\mathbf{B}_t \sim$ Unif$(\mathbb{F}_q)$ and $g(B_t), H_t \in \mathbb{F}_p$, where $q, p \in \mathbb{N}$ and $\mathbb{F}_q, \mathbb{F}_p$ are finite fields of order q and p respectively. Uniform sampling of blocks is assumed so as to consider the worst-case, i.e. maximum entropy, storage scenario. Thus, the cost of storage per peer per transaction in conventional implementation is

$$\tilde{R}_s = \log_2 q + 2\log_2 p \text{ bits}. \tag{3}$$

In practice, data blocks can be of varying sizes and the results follow *mutatis mutandis*. For the uniform random oracle model, the pre-image and collision resistance characteristics of the hash family in use is

$$P_{pre-image} \approx \frac{1}{p}, \quad P_{collision} \approx \frac{1}{p}.$$

We analyze with respect to the random oracle model [31].

Transactions stored in the ledger may at a later point be recovered, in order to validate claims or verify details, by nodes that have read access to the data. Different implementations of the blockchain invoke different recovery mechanisms depending on the application. One such method is authentication where select peers return the data stored in the ledger and the other peers validate (sign) the content. Depending on the application, one can envision varying the number of authorizations necessary to validate the content. As described earlier, for ease of description, we restrict this work to the majority rule, i.e., in order to recover the t-th transaction, each peer returns its copy of the transaction and the majority rule is applied to recover the block.

B. Blockchain Security

Two aspects of security are of interest in blockchains—*integrity* and *confidentiality*. Whereas an integrity property guarantees that the stored data cannot be corrupted unless most of the peers are corrupted, a confidentiality property ensures that local information from individual peers does not reveal sensitive transaction information.

Corruption of data in the blockchain requires corrupting a majority of the peers in the network to alter the data stored in the distributed, duplicated copy of the transaction ledger. Thus, the blockchain system automatically ensures a level of integrity in the transaction data.

Conventional blockchain systems such as bitcoin enforce additional constraints on the hash values to enhance data integrity. For instance, in the bitcoin network, each transaction block is appended with a nonce which is typically a string of zeros, such that the corresponding hash value satisfies a difficulty target i.e., is in a specified constraint set. The establishment of such difficulty targets in turn implies that computing a nonce to satisfy the hash constraints is computationally expensive. Thus data integrity can be tested by ensuring the hash values are consistent as it is computationally infeasible to alter the data.

The hash chain, even without the difficulty targets, offers a mechanism to ensure data integrity in the blockchain. Specifically, note that the pre-image resistance of the cryptographic hash function ensures that any change to H_{t-1} or B_t would require recomputing H_t with high probability as it is computationally infeasible for an adversary to determine W_t such that $h(I_t, W_t) = H_t$. To be precise, due to collision resistance, any alteration of data in W_t results in an alteration of H_t with probability at least $1 - \frac{1}{p}$.

Thus storing the ledger in the form of a hash chain ensures that corrupting a past transaction not only requires the client to corrupt at least half the set of peers to change the majority, but also maintain a consistent hash chain following the corrupted transaction. That is, say a participating client wishes to alter transaction B_1 to B_1'. Let there be T transactions in the ledger. Then, corrupting B_1 implies that the client would also have to replace H_1 with $H_1' = h(I_1, W_1')$ at the corrupted nodes. This creates a domino effect, in that all subsequent hashes must in turn be altered with high probability as well, to maintain integrity of the chain. This strengthens the integrity of the transaction data in blockchain systems.

Confidentiality of information is typically guaranteed in these systems through the use of private key encryption, where the key is shared with a select set of peers who are authorized to view the transaction. Note however that in such implementations, a leak at a single node could lead to a complete disambiguation of information. One approach to enhance data confidentiality would be to distribute data so that a single point of leakage does not divulge all the information.

Let us now define the adversaries to protect the data from.

C. Adversary Model

In this work, we explore the construction of a distributed storage coding scheme for the *cold storage* of the blockchain ledger that ensures heightened confidentiality and integrity of the data, even when the mining process is computationally inexpensive. Let us assume that each transaction B_t also has a corresponding *access list*, which is the set of nodes that have permission to read and edit the content in B_t. Note that this is equivalent to holding a private key to decrypt the encrypted transaction data stored in the ledger.

Here, we primarily focus on *active adversaries* who alter transaction content B_t to a desired value B_t'. Let us explicitly define the semantic rules of a valid corruption for such an adversary. If a client corrupts a peer, then the client can

1) learn the contents stored in the peer;
2) alter block content only if it is in the access list of the corresponding block; and
3) alter hash values as long as chain integrity is preserved, i.e., an attacker cannot invalidate the transaction of another node in the process.

We assume the active adversary is aware of the contents of the hash chain and the block it wishes to corrupt. This adversary is stronger than a typical adversary who is unaware of contents of the transaction and only wishes to corrupt data. Further, this adversary aims to not just corrupt the block, but to precisely alter it to some preferred content B_t' that would be validated by the network. In a land ownership blockchain, this might involve changing land ownership to the adversary itself.

Moreover, the active adversary adapts to the deployed storage algorithm and can further learn from the contents shared by corrupted peers. We elaborate on the integrity of our coding scheme against such active adversaries. We also briefly discuss data confidentiality as guaranteed by our system against local information leaks.

Another typical attack of interest in such blockchains is the *denial of service* attack where an adversary corrupts peers in the network to deny the requested service, which in this case is returning the data stored in the ledger. Here we study the probability of data loss arising from denial of service by an adversary with an average corruption budget of B_{dl}. That is, the adversary can, on average prevent B_{dl} peers per transaction

from sharing their stored ledger content. The adversary may choose different numbers of peers for different transactions as long as the average number of corrupted peers is limited to the budget B_{dl}. This allows more generic denial of service policies than the classical model where precisely d peers are corrupted per transaction. Apart from the choice of number of peers to corrupt, the DoS adversaries are static in nature.

Finally, we briefly consider the problem of data confidentiality in Sec. V-D. Here external observers are allowed to observe encrypted data stored at the individual peers of the network, but not the keys used to encrypt the data. In such a context, we ask how much information about the data can be inferred by the observer.

Before we describe the code construction, we first give a preliminary introduction to coding and encryption schemes that we use as the basis to build our coding scheme.

III. Preliminaries

This work uses a private key encryption scheme with a novel combination of secret key sharing and distributed storage codes to store the transaction data and hash values. We now provide a brief introduction to these elements.

A. Shamir's Secret Sharing

Consider a secret $S \in \mathbb{F}_q$ that is to be shared with $n < q$ nodes such that any subset of size less than k get no information regarding the secret upon collusion, while any subset of size at least k get complete information. Shamir's (k, n) secret sharing scheme [18] describes a method to explicitly construct such a code over a finite field $\mathbb{F}_q$ as follows.

Let $a_0 = S$. Draw $a_i \overset{i.i.d}{\sim} \text{Unif}(\mathbb{F}_q)$ and compute

$$y_i = a_0 + a_1\, x_i + a_2\, x_i^2 + \cdots + a_{k-1} x_i^{k-1}, \text{ for all } i \in [n],$$

where $x_i \in \mathbb{F}_q \backslash \{0\}$. Node i receives the share y_i. Since the values are computed using a polynomial of order $k-1$, the coefficients of the polynomial can be uniquely determined only when we have at least k values. Thus the secret key is recovered using polynomial interpolation over at least k shares of the secret.

In this work, we presume that each secret share is given by (x_i, y_i), and that the unique abscissa values are chosen uniformly at random, without replacement from $\mathbb{F}_q \backslash \{0\}$. Then, given any $k-1$ shares and the secret, the final share is uniformly likely in a set of size $q-k$.

It is worth noting that Shamir's scheme is minimal in storage as the size of each share is the same as the size of the secret key. Shamir's scheme however is not secure to active adversaries. In particular, by corrupting $n-k+1$ nodes, the secret can be completely altered. Secret key sharing codes have been widely studied in the past [32]–[34]. In particular, it is known that Reed-Solomon codes can be adopted to define secret shares. Linear codes for minimal secret sharing have also been considered [35]. In this work however, we restrict to Shamir's secret sharing scheme for simplicity.

B. Data Encryption

Shannon famously concluded that perfect secrecy required the use of keys drawn from a space as large as the message space [36]. This is practically unusable as it is difficult to securely store such large keys. Thus practical cryptographic systems leverage computational limitations of an adversary to guarantee security over perfect statistical secrecy.

We define a notion of encryption that is slightly different from that used typically in cryptography. Consider a message $\mathbf{M} = (M_1, \ldots, M_m) \in \mathcal{M}$, drawn uniformly at random. Let $K \in \mathcal{K}$ be a private key drawn uniformly at random.

Definition 2: Given message, key, and code spaces $\mathcal{M}, \mathcal{K}, \mathcal{C}$ respectively, a *private key encryption scheme* is a pair of functions $\Phi : \mathcal{M} \times \mathcal{K} \to \mathcal{C},\ \Psi : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$, such that,

$$\Psi(\Phi(\mathbf{M}; K), K) = \mathbf{M}, \text{ for all } \mathbf{M} \in \mathcal{M}, K \in \mathcal{K}$$

and it is ϵ-secure if it is statistically impossible to decrypt the codeword in the absence of the private key K with probability greater than ϵ, i.e.,

$$\max_{\mathbf{M} \in \mathcal{M}, \mathbf{C} \in \mathcal{C}} \mathbb{P}\left[\Psi(\mathbf{C}, K) = M\right] \leq \epsilon. \tag{4}$$

The definition indicates that the encryption is invertible, and that it is statistically infeasible to decrypt the cipher beyond a degree of certainty. For convenience, we assume that the encrypted codewords are vectors of the same length as the message from an appropriate alphabet, i.e., $\mathbf{C} = (C_1, \ldots, C_m)$.

Since we want to secure the system from adversaries who are aware of the plaintext message, we define a stronger notion of secure encryption. We assume that an attacker, given message, $\mathbf{M}$, and $\mathbf{C}_{-j} = (C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_m)$, is statistically incapable of guessing C_j, in the absence of knowledge of the key K, i.e.,

$$\mathbb{P}\left[\Phi(\mathbf{M}; K) = \mathbf{C} | \mathbf{M}, \mathbf{C}_{-j}\right] \leq \frac{1}{2}, \text{ for any } C_j. \tag{5}$$

A more general requirement would bound the probability of recovering the key, given incomplete cipher text, by a parameter δ. In this description, for ease of description let us work with the factor $\frac{1}{2}$. The generalization of the analysis to a parameter δ is straightforward. Note that this criterion indicates that the adversary is unaware of at least 1 bit of information in the unknown code fragment, despite being aware of the message, i.e.,

$$H(\mathbf{C} | \mathbf{M}, \mathbf{C}_{-j}) \geq 1 \text{ bit},$$

where $H(\cdot)$ is the Shannon entropy.

C. Distributed Storage Codes

Distributed storage codes have been widely studied [19], [37] in different contexts. In particular, aspects of repair and security, including explicit code constructions have been explored widely [23]–[25], [38], [39]. In addition, information dispersal algorithms [21], [22] have also considered the question of distributed storage of data. This is a non-exhaustive listing of the existing body of work on distributed storage and most algorithms naturally adapt to the coding scheme defined here. However, we consider the simple form of distributed storage that just divides the data evenly among nodes.

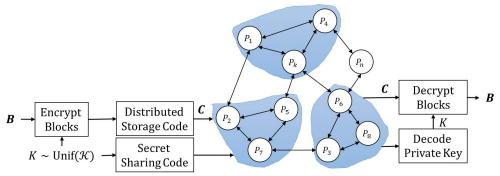

Fig. 3. Encryption and decryption process for a given zone allocation. The shaded regions represent individual zones in the peer network. The data is distributed among peers in each zone and the data from all peers in a zone are required to recover the transaction data.

Algorithm 1 Coding Scheme for Data Block

for $z = 1$ to $\frac{n}{m}$ **do**
  Generate private key $K_t^{(z)} \sim \text{Unif}(\mathcal{K})$
  Encrypt block with key $K_t^{(z)}$ as $\mathbf{C}_t^{(z)} = \Phi(\mathbf{B}_t; K_t^{(z)})$
  Distribute $\mathbf{C}_t$ and store among peers in $\{i : p_t^{(i)} = z\}$
  Use Shamir's (m, m) secret sharing on $K_t^{(z)}$ and distribute shares $(K_1^{(z)}, \ldots, K_{m_t}^{(z)})$ among peers in the zone
end for

IV. Coding Scheme

For this section, assume that at any point of time t, there exists a partition $\mathcal{P}_t$ of the set of peers $[n]$ into sets of size m each. In this work we presume that n is divisible by m. Let each set of the partition be referred to as a *zone*. Without loss of generality, the zones are referred to by indices $1, \ldots, \frac{n}{m}$. At each time t, for each peer $i \in [n]$, let $p_t^{(i)} \in [\frac{n}{m}]$ be the index that represents the zone that includes peer i. We describe the zone allocation scheme in detail in Sec. VI.

A. Coding Data Block

In our coding scheme, a single copy of each data block is stored in a distributed fashion across each zone. Consider the data block $\mathbf{B}_t$ corresponding to time t. We use a technique inspired by [20]. First a private key K is generated at each zone and the data block is encrypted using the key. The private key is then stored by the peers in the zone using Shamir's secret key sharing scheme. Finally, the encrypted data block is distributed amongst peers in the zone using a distributed storage scheme. The process involved in storage and recovery of a block, given a zone division is shown in Fig. 3.

More generally we can allow the zone sizes at time t to be chosen by the client. For ease however, we describe the coding scheme for constant zone sizes m. To customize the storage, we just need to replace the zone sizes by m_t and use a corresponding key space $\mathcal{K}_{m_t}$. The coding scheme is given by Alg. 1. In this discussion we will assume that the distributed storage scheme just distributes the components of the code vector $\mathbf{C}_t$ among the peers in the zone. The theory extends naturally to other distributed storage schemes.

To preserve the integrity of the data, we use secure storage for the hash values as well. In particular, at time t, each zone $Z \in \mathcal{P}_t$ stores a secret share of the hash value H_{t-1} generated using Shamir's (m, m) secret sharing scheme.

The storage per transaction per peer is thus given by

$$R_s = \frac{1}{m} \log_2 |\mathcal{C}| + 2 \log_2 |\mathcal{K}| + 2 \log_2 p \text{ bits}, \tag{6}$$

Algorithm 2 Recovery Scheme for Data Block

$\mathcal{N} \leftarrow [n]$
Compute $K_t^{(z)}$, for all z, by polynomial interpolation
Decode blocks $B_t^{(z)} \leftarrow \Psi\left(\mathbf{C}_t^{(z)}; K_t^{(z)}\right)$, for all $z \in [\frac{n}{m}]$
if $|\{B_t^{(z)} : z \in [\frac{n}{m}]\}| > 1$ **then**
  for $\tau = t$ to $\min\{t + d_t, T\}$ **do**
    Compute $H_\tau^{(z)}$, for all z, by polynomial interpolation
    Determine $W_\tau^{(i)} = \left(g(B_\tau^i), H_{\tau-1}^i\right)$, for all $i \in [n]$
    $\mathcal{I} \leftarrow \left\{i : h(W_\tau^{(i)}) \neq H_\tau^{(z)}, z = p_{\tau+1}^{(i)}\right\}$
    $\mathcal{N} \leftarrow \mathcal{N} \backslash \mathcal{I}$
    if $|\{B_t^{(p_t^{(i)})} : i \in \mathcal{N}\}| = 1$ **then**
      break
    end if
  end for
end if
return Majority in $\{\{B_t^{(p_t^{(i)})} : i \in \mathcal{N}\}\}$

where $|\mathcal{C}| \geq q$ depending on the encryption scheme. In particular, when the code space of encryption matches the message space, i.e., $|\mathcal{C}| = q$, the gain in storage cost per transaction per peer is given by

$$\text{Storage Gain} = \tilde{R}_s - R_s = \frac{m-1}{m} \log_2 q - \log_2(p|\mathcal{K}|^2) \text{ bits}. \tag{7}$$

Thus, when the size of the private key space is much smaller than the size of the blocks, we have a storage reduction.

B. Recovery Scheme

We now describe the algorithm to retrieve a data block B_t in a blockchain with a total of T blocks. However, instead of exploring the entire length until we identify a unique consistent version, we provide the client the freedom to choose the depth d_t of blocks that follow in the hash chain, and return the majority consistent version. The algorithm to recover block B_t is in Alg. 2.

According to Alg. 2, each peer first communicates the codeword corresponding to the data block and the secret share of the encryption key. This corresponds to $\frac{1}{m_t} \log_2\ q + m_t(2 \log_2\ m_t + 1)$ bits. Additionally, each peer also communicates the secret shares of hash values corresponding to the next d blocks, each of which contributes $2 \log_2\ p$ bits, and the corresponding data blocks for consistency check. Thus the total worst-case cost of recovering the t-th data block is

$$R_r^{(t)} = C_r \left(\tfrac{1}{m} \log_2 q + \log_2 |\mathcal{K}| + 4 d_t \log_2 p\right), \tag{8}$$

where C_r is the cost per bit of communication.

The recovery algorithm exploits information-theoretic security in the form of the coding scheme, and also invokes the hash-based computational integrity check established in the chain. First, the data blocks are recovered from the distributed, encrypted storage from each zone. In case of a data mismatch, the system inspects the chain for consistency in the hash chain. The system scans the chain for hash values and eliminates peers that have inconsistent hash values. A hash value is said

Algorithm 3 Encryption Scheme

$T \leftarrow \text{Unif}(\mathcal{T})$, $K \leftarrow \text{Key}(T)$; $\mathbf{b} \leftarrow \text{Binom}(n, 1/2)$
Assign peers to vertices, i.e., peer i is assigned to node θ_i
For all $i \neq v_0$, $\tilde{C}_i \leftarrow B_i \oplus B_{\mu_i}$; flip bits if $b_i = 1$.
$\tilde{C}_{v_0} \leftarrow \left(\oplus_{j \neq v_0} \tilde{C}_j\right) \oplus B_{v_0}$
if $b_{v_0} = 1$ **then**
  Flip the bits of $\tilde{C}_{v_0}$
end if
Store $C_i \leftarrow \tilde{C}_{\theta_i}$ at each node i in the zone
Store (K, θ) using Shamir's secret sharing at the peers
Store the peer assignment θ_i locally at each peer i

to be inconsistent if the hash value corresponding to the data stored by a node in the previous instance does not match the current hash value. Through the inconsistency check, the system eliminates some, if not all corrupted peers. Finally, the majority consistent data is returned.

In the implementation, we presume that all computation necessary for the recovery algorithm is done privately by a black box. In particular, we presume that the peers and clients are not made aware of the code stored at other peers or values stored in other blocks. Specifics of practical implementation of such a black box scheme are beyond the scope of this work.

C. Feasible Encryption Scheme

The security of the coding scheme from corruption by active adversaries depends on the encryption scheme used. We first describe the necessary condition on the size of the key space.

Lemma 1: For any valid encryption scheme satisfying (5), $|\mathcal{K}| \geq 2^m$.

The proof can be found in App. A. More generally, if $\delta = \frac{1}{2^\ell}$, then $|\mathcal{K}| \geq 2^{\ell m}$.

We now describe an encryption scheme that is order optimal in the size of the private key space upto log factors. Let $\mathcal{T}$ be the set of all rooted, connected trees defined on m nodes. Then, by Cayley's formula [40],

$$|\mathcal{T}| = m^{(m-1)}.$$

Let us define the key space by the entropy-coded form of uniform draws of a tree from $\mathcal{T}$. Hence the encryption scheme presumes that given the private key K, we are aware of all edges in the tree. Let $V = [m]$ be the nodes of the tree and v_0 be the root. Let the parent of a node i in the tree be μ_i.

Consider the encryption function, Φ, given in Alg. 3. Here, $\text{Key}(T)$ is the sampling function that generates a key K from the set of all keys corresponding to the chosen tree architecture T. Without loss of generality we assume the keys are sampled uniformly at random. The encryption algorithm proceeds by first selecting a rooted, connected tree uniformly at random on m nodes. Then, each peer is assigned to a particular node of the tree. For each node other than the root, the codeword is created as the modulo 2 sum of the corresponding data block and that corresponding to the parent. Finally, the root is encrypted as the modulo 2 sum of all codewords at other nodes and the corresponding data block. The bits stored at the root node are flipped with probability half. The encryption scheme for a sample data block is shown in Fig. 4.

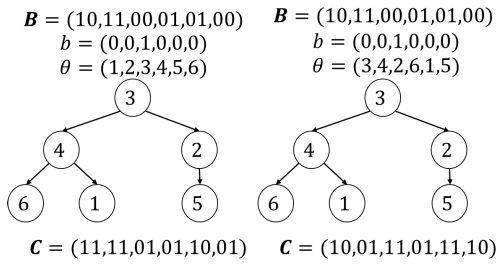

Fig. 4. Encryption examples for a zone with six peers. The data block, parameters, tree structure, and corresponding codes are shown. The two cases consider the same rooted tree with varying peer assignments. The corresponding change in the code is shown.

Algorithm 4 Decryption Scheme

Use polynomial interpolation to recover $(K, \mathbf{b}, \theta)$
Define $\tilde{\theta}_i \leftarrow j$ if $\theta_j = i$
Flip the bits of $C_{\tilde{\theta}_{v_0}}$, if $b_{v_0} = 1$
$B_{v_0} \leftarrow C_{\tilde{\theta}_{v_0}} \oplus_{j \neq \tilde{\theta}_{v_0}} C_j$
For all $i \in [n] \backslash \{v_0\}$, flip bits of $\mathbf{C}_i$ if $b_i = 1$
Iteratively compute $B_i \leftarrow C_{\tilde{\theta}_i} \oplus B_{\mu_i}$ for all $i \neq v_0$
return $\mathbf{B}$

The decryption of the stored code is as given in Alg. 4. That is, we first determine the private key, i.e., the rooted tree structure, the bit, and peer assignments. Then we decrypt the root node by using the codewords at other peers. Then we sequentially recover the other blocks by using the plain text message at the parent node.

Lemma 2: The encryption scheme Φ satisfies (4) and (5).

The proof is given in App. A.

Lemma 3: The storage cost per peer per transaction under (Φ, Ψ) is

$$R_s(\Phi, \Psi) = \frac{1}{m} \log_2 q + m \left(2 \log_2 m + 1\right) + 2 \log_2 p \text{ bits.} \tag{9}$$

The proof is given in App. A. From Lem. 3, we can see that the encryption scheme guarantees order-optimal storage cost per peer per transaction up to $\log$ factor in the size of the key space. The security of the encrypted data can be enhanced by increasing the inter-data dependency by using directed acyclic graphs (DAGs) with bounded in-degree in place of the rooted tree. Then, the size of storage for the private key increases by a constant multiple.

If we work with the Merkle tree representation of the hash chain, then each peer stores two hash values per block, the Merkle root and the hash header for the block. Thus, the storage cost in (9) is increased by $2 \log_2\ p$ bits for the Merkle root which is also stored using Shamir's secret sharing.

V. Performance of Coding Scheme

We now evaluate the data integrity and confidentiality that arise from the coding scheme described here.

A. Individual Block Corruption

We now establish the security guarantees of individual blocks in each zone from active adversaries. To begin, we know Shamir's secret sharing scheme results in secure storage from an active adversary as long as the number of corruptions $\tilde{k} \leq \frac{n-m+1}{2}$. Our coding scheme leverages the hash chain to protect the data from stronger active adversaries. We describe security in terms of the probability that an active adversary might corrupt a data block successfully.

First, consider an adversary who is aware of the hash value H_t and wishes to alter it to H'_t.

Lemma 4: Say an adversary, aware of the hash value H_t and the peers in a zone, wishes to alter the hash value to H'_t. Then, the probability of successful corruption of such a system when at least one peer is honest, is $O(1/p)$.

The proof is given in App. B. This indicates that in order to corrupt a hash value, the adversary practically needs to corrupt all nodes in the zone.

To understand corruption of data blocks, we first consider the probability of successful corruption of a zone without corrupting all peers of the zone.

Lemma 5: Consider an adversary, aware of the plain text $\mathbf{B}$ and the peers in a zone. If the adversary corrupts $c < m$ peers of the zone, then the probability that the adversary can alter the data to $\mathbf{B}'$ is at most $\frac{c^2}{m^2}$, i.e.,

$$\mathbb{P}\left[\mathbf{B} \to \mathbf{B}' \text{ in zone } z\right] \leq \exp\left[2\log\tfrac{c}{m} - \left(1-\tfrac{2}{m}\right)\left(1-\tfrac{c}{m}\right)\right], \tag{10}$$

for all $\mathbf{B} \neq \mathbf{B}', z \in [\frac{n}{m}]$.

The proof of the lemma is given in App. B.

A consistent corruption of a transaction by an active adversary requires corruption of at least $\frac{n}{2m}$ zones. This characterizes the probability of successful corruption as shown below.

Theorem 1: Consider an active adversary who corrupts $c_1, \ldots, c_{n/2m}$ peers respectively in $\frac{n}{2m}$ zones. Then, the probability of successful corruption across the set of all peers is

$$\mathbb{P}\left[\text{Successful corruption } \mathbf{B} \to \mathbf{B}'\right] \leq \exp\left(\frac{n}{m}\left[\log\left(\frac{2s}{n}\right) - \left(\frac{m-2}{m}\right)\left(\frac{1}{2}-\frac{s}{n}\right)\right]\right), \tag{11}$$

for all $\mathbf{B} \neq \mathbf{B}'$, where $s = \sum_{i=1}^{n/2m} c_i$.

The proof is detailed in App. B.

Note that $\sum_{i=1}^{n/2m} c_i \leq \frac{n}{2}$, and thus the corruption probability decays with network size if less than half the network is corrupted. From Thm. 1, we get the following corollary.

Corollary 1: If an adversary wishes to corrupt a data block with probability at least $1-\epsilon$, for some $\epsilon > 0$, then the total number of peers the adversary needs to corrupt satisfies

$$\sum_{i=1}^{n/2m} c_i \geq \frac{n}{2}(1-\epsilon)^{\frac{m}{n}}. \tag{12}$$

Corollary 1 indicates that when the network size is large, the adversary practically needs to corrupt at least half the network to have the necessary probability of successful corruption. Thus we observe that for a fixed zone division, the distributed storage system loses an arbitrarily small amount of data integrity as compared to the conventional scheme.

In Sec. VI, we introduce a dynamic zone allocation scheme to divide the peer network into zones for different time slots. We show that varying the zone allocation patterns over time appropriately yields even better data integrity.

B. Alternative Corruption Check

One method to increase the strength of the encryption is to generalize the scheme using directed acyclic graph with constant in-degree. Instead of encrypting the data prior to distribution, in this section we propose a hashing-based idea to enable a consistency check within each zone, to highlight the applicability of classical encryption alternatives. A thorough evaluation of other encryption methods from the literature is beyond the scope of this paper.

Given a block B_t to be distributed among the peers of the zone z, sample a nonce $N_z \sim \text{Unif}(\mathcal{N})$, and compute the hash value $G_t = h(I, (B_t, N_z))$, where $I \sim \text{Unif}(\mathcal{I})$. Then, we share the nonce and the hash value as a secret among the peers of the zone. Then, B_t is stored according to a distributed storage code among the peers of the zone as plaintext.

When recovering the data, note that each zone can verify data integrity by recovering the secret nonce, padding it with the data, and checking the consistency of the stored hash value. Note that the storage cost of this scheme is characterized by

$$\mathcal{R}'_s = \frac{1}{m}\log_2\ q + 2\left(\log_2|\mathcal{N}| + 2\log_2\ p\right) \text{bits}.$$

When $|\mathcal{N}| \gg p$, the probability that an active adversary, who corrupts only $c < m$ peers of a zone, can successfully corrupt a data block B is essentially the same as finding a collision in the hash value. This is due to the fact that the adversary has no way to recover the nonce and hash value stored as a secret in the zone, and so has to essentially guess a transformation to the data that happens to share the same hash value. This is the same as the probability of collision and so for any zone z, and any block B,

$$\mathbb{P}\left[\text{Successful corruption of zone } z\right] \leq P_{\text{collision}}.$$

Thus, the probability of successful data corruption by adversary who corrupts fewer than $n/2$ peers is upper bounded due to the independence across zones as

$$\mathbb{P}\left[\text{Corruption}\right] \leq \exp\left(-\frac{n\log\frac{1}{P_{\text{collision}}}}{2m}\right) \approx \exp\left(-\frac{n\log p}{2m}\right).$$

Note however that the efficacy of this method requires the padded nonce is drawn from a sufficiently large space $\mathcal{N}$. As such, the storage overhead $\log_2|\mathcal{N}|$ can be considerably larger than the $m\log_2\ m$ bound of Alg. 1. As such, Alg. 1 is better optimized for the security requirements of practical systems. On the other hand, this hash-based scheme provides a simple, generalizable implementation that is easily adapted across distributed large-scale networks.

C. Data Loss

A data block is lost when some peers undergo a DoS attack and a sufficiently large number incurs random data loss. Consider an adversary that wishes to prevent the recovery of a block $\mathbf{B}$ distributed according to the parameter m. Let

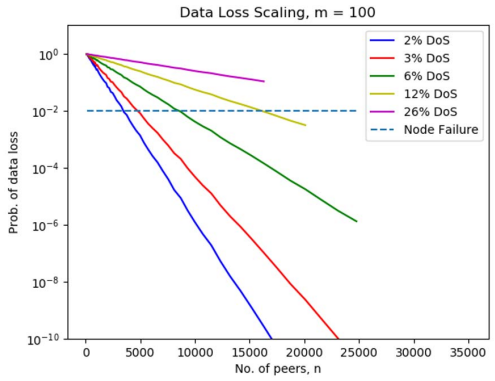

Fig. 5. Data loss scaling with number of peers.

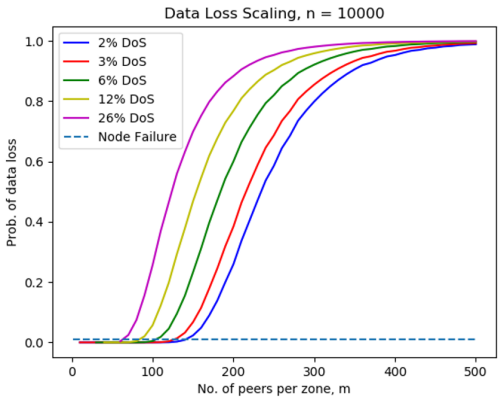

Fig. 6. Data loss scaling with number of peers per zone.

$r = n/m$ be the number of copies of the data in the peer network. The data is lost when there exists at least one node failure or DoS attack in each zone.

The adversary picks a random number $C \sim P_{\text{dl}}$ and performs a DoS attack on C uniformly random peers. Such a generalized adversarial model allows for different DoS policies by the adversary, beyond the classical model of an adversary that corrupts exactly C peers at random. Let the number of peers corrupted in zone i be X_i and Y_i be the number that undergo data loss. Thus, $(X_1, \ldots, X_r)$ are distributed according to the multivariate hypergeometric distribution with n objects, C draws, and m objects of each of the r types.

Theorem 2: Consider a denial of service adversary who can deny service at B_{dl} peers on average. Then, the worst-case probability of data loss is the solution to the LP

$$\begin{aligned} P_{\text{dl}} \in \arg\max_{p} \quad & \sum_{c=0}^{n} p(c)\mathbb{E}\left[(1-\bar{\rho}^m)^{(r-\sum_{i=1}^{r}\mathbf{1}\{X_i>0\})}|C=c\right] \\ \text{s.t.} \quad & \sum_{c=0}^{n} cp(c) \le B_{\text{dl}}, \ \sum_{c=0}^{n} p(c) = 1, \\ & \text{and } p(c) \ge 0, \text{ for all } c. \end{aligned} \tag{13}$$

The proof is given in App. C.

Computing the conditional expectation in (13) requires knowledge of the probability mass function (pmf) of the number of zones with non-zero corruption, given by

$$\mathbb{P}\left[\sum_{i=1}^{r}\mathbf{1}\{X_i>0\}=\tilde{r}\right] = \binom{r}{\tilde{r}}\mathbb{P}\left[\sum_{i=1}^{\tilde{r}}X_i=c\right] \tag{14}$$

$$= \binom{r}{\tilde{r}}\binom{c-1}{\tilde{r}-1}\Big/\binom{r+c-1}{c}, \tag{15}$$

where (14) follows from the symmetry in the zones. Then, $\sum_{i=1}^{\tilde{r}} X_i$ follows a hypergeometric distribution with parameters $n, c, \tilde{r}m$ representing population size, number of draws, and number of successes respectively. This results in (15).

To understand the scaling of the data loss probability, let us consider the effect of a DoS adversary who corrupts c peers. First, in Fig. 5, we observe that the data loss probability decreases exponentially with the number of peers in the system for a given m. Further, as expected, the smaller the fraction of peers corrupted by the DoS adversary, the lesser the data loss probability. Naturally, practical systems with a large number of participating peers result in considerably low data loss probability, and prove to be inherently robust.

Next, in Fig. 6, we show the data loss probability as a function of the number of peers per zone, m, given a network with $n = 10,000$ peers. The larger the number of peers per zone, the higher the probability of data loss, as there are fewer shares of the data in the network. Thus, any reduction in storage comes at the added cost of data loss.

D. Data Confidentiality

We earlier stated that the two aspects of security needed in blockchains are data integrity and confidentiality. We addressed the question of data integrity in the previous subsection. We now consider confidentiality of transaction data.

Consider the situation where an external adversary receives the data stored by the a peer i in a zone for one particular slot. This includes the secret share of the private key K_i, the encrypted block data C_i, and secret share corresponding to the hash of the previous block. From Shamir's secret sharing scheme, we know that knowledge of K_i gives no information regarding the actual private key K. Thus, the adversary has no information on the rooted tree used for encryption.

We know that the transaction data are chosen uniformly at random. Since the adversary is unaware of the relation of the nodes to one another, from the encryption scheme defined, we know that given the entire encrypted data $\mathbf{C}$, the probability of recovering the block $\mathbf{B}$ is uniformly distributed on the set of all possible combinations obtained for all possible tree configurations. That is, each possibly rooted tree yields a potential candidate for the transaction data.

This observation implies that

$$H(\mathbf{B}|\mathbf{C}) \le H(K, \mathbf{B}|\mathbf{C}) = H(K) + H(\mathbf{B}|K, \mathbf{C}) = m\log_2 m. \tag{16}$$

We know that the entropy of the transaction block is actually $H(\mathbf{B}) = \log_2 q > m\log_2 m$. That is, the adversary does learn the transaction data partially, given the entire codeword.

More generally, if the external observer learns k of the m blocks pertaining to a zone, with code blocks

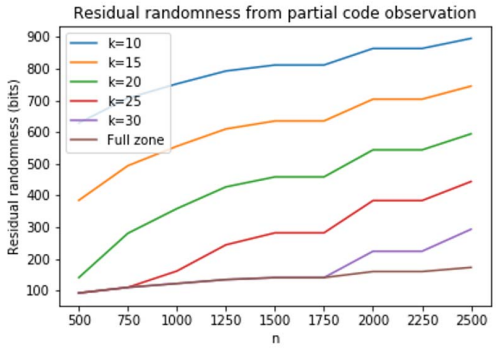

Fig. 7. Residual randomness in data from partial observation of code: When an external observer learns the code blocks corresponding to k of the m peers in a zone, the added confidentiality increases with the size of the network.

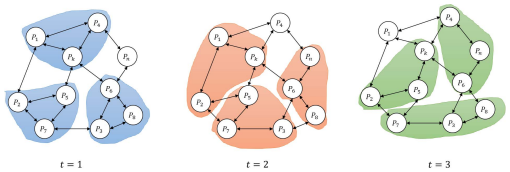

Fig. 8. Dynamic zone allocation: Iterate zones among peers so that increasing number of peers need to be corrupted to maintain a valid hash chain.

$\mathcal{C}^k = \{C_1, \ldots, C_k\}$, then, if $\mathcal{C}^{(-k)} = \mathbf{C} \backslash \mathcal{C}^k$,

$$\begin{aligned} H(\mathbf{B}|\mathcal{C}^k) &\leq H(K, \mathcal{C}^{(-k)}\mathbf{B}|\mathcal{C}^k) \\ &= H(\mathcal{C}^{(-k)}|\mathcal{C}^k) + H(K|\mathbf{C}) + H(\mathbf{B}|K, \mathbf{C}) \\ &= \left(1 - \tfrac{k}{m}\right) \log_2 q + m \log_2 \ m. \end{aligned}$$

The scaling of this residual randomness as a function of network size for a block of size 1 KB is shown in Fig. 7. Notice the confidentiality added by the system increases with the size of the network. Thus, local leaks reveal very little information about the transaction, i.e., the coding scheme also ensures a high degree of confidentiality in case of data leaks from up to $m - 1$ peers in a zone.

VI. Dynamic Zone Allocation

Earlier we presumed the existence of a zone allocation strategy over time. Here we make it explicit.

Cor. 1 and Lem. 4 prove that the distributed secure encoding process ensures that corrupting a transaction or a hash requires an adversary to corrupt all peers in the zone. This can be exploited to ensure that with each transaction following the corrupted transaction, the client would need to corrupt an increasing set of peers to maintain a valid hash chain.

In particular, let us assume a blockchain in the state

$$(H_0, \mathbf{B}_1) - (H_1, \mathbf{B}_2) - \cdots - (H_{t-1}, \mathbf{B}_t).$$

Let us assume without loss of generality that an adversary wishes to corrupt the transaction entry $\mathbf{B}_1$ to $\mathbf{B}'_1$. The validated, consistent version of such a corrupted chain is

$$(H_0, \mathbf{B}'_1) - (H'_1, \mathbf{B}_2) - \cdots - (H'_{t-1}, \mathbf{B}_t).$$

If the zone segmentation used for encoding is static, the adversary can easily maintain such a corrupted chain at half the peers to validate its claim. If each peer is paired with varying sets of peers across blocks, then, for sufficiently large t, each corrupted peer eventually is grouped with an uncorrupted peer.

Let us assume this occurs at slot $\tau + 1$. Then, in order to successfully corrupt the hash H_τ to H'_τ, the adversary would need to corrupt the uncorrupted peers in this zone. If not, the hash values reveal an inconsistency at the corrupted peers.

Thus, it is evident that if the zones are sufficiently well distributed, corrupting a single transaction would eventually require corruption of the entire network, and not just a majority. A sample allocation scheme is shown in Fig. 8.

However, the total number of feasible zone allocations is

$$\text{No. of zone allocations} = \frac{n!}{(m!)^{\frac{n}{m}}} \approx \frac{\sqrt{2\pi n}}{\left(\sqrt{2\pi m}\right)^{\frac{n}{m}}} \left(\frac{n}{m}\right)^n,$$

which increases exponentially with the number of peers and is monotonically decreasing in the zone size m. This indicates that naive deterministic cycling through this set of all possible zone allocations is practically infeasible.

To ensure that every corrupted peer is eventually grouped with an uncorrupted one we need to ensure that every peer is eventually grouped with every other. Further, the allocation needs to ensure uniform security for every transaction.

In order to better understand the zone allocation strategy, we first study a related combinatorial problem.

Algorithm 5 Dynamic Zone Allocation Strategy

Let $\nu_2 \ldots, \nu_{2r}$ be the vertices of a $2r - 1$ regular polygon, and ν_1 its center
for $i = 2$ to $2r$ **do**
  Let L be the line passing through ν_1 and ν_i
  $M \leftarrow \{(\nu_j, \nu_k) :$ line through ν_j, ν_k perp. to $L\}$
  $M \leftarrow M \cup \{(\nu_1, \nu_i)\}$
  Construct zones as $\{\nu_j \cup \nu_k : (\nu_j, \nu_k) \in M\}$
end for
restart for loop

A. K-Way Handshake Problem

Consider a group of n people. At each time slot the people are to be grouped into sets of size m. A peer gets acquainted with all peers in the group whom they have not met before. The problem is thus an m-way handshake between people.

Lemma 6: The minimum number of slots required for every peer to shake hands with every other peer is $\frac{n-1}{m-1}$.

Proof: Follows since a peer meets at most $m - 1$ new peers in a slot. ∎

We use the tightness observed for the two-way handshake problem to design a strategy to assign the peers in zones. Let $r = \frac{n}{m}$. Partition the peers into $2r$ sets, each containing $m/2$ peers. Let these sets be given by $\nu_1, \ldots, \nu_{2r}$. Then, we can use matchings of K_{2r} to perform the zone allocation.

The Alg. 5 provides a constructive method to create zones such that every peer is grouped with every other over time. The functioning of the algorithm is as in Fig. 9.

Lemma 7: The number of slots required for every peer to be grouped with every other peer is $2r - 1$.

Proof: The result follows directly from the cyclic decomposition and Baranyai's theorem for $m = 2$ [41]. ∎

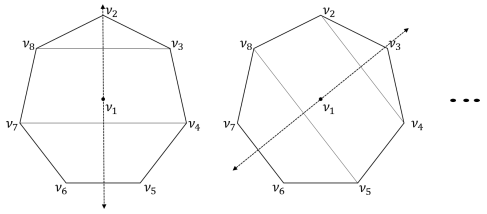

Fig. 9. Dynamic zone allocation when $n = 4m$. Algorithm cycles through matchings of K_n by viewing them as regular polygon orientations.

We see that the scheme matches the lower bound on the number of slots for coverage in the order sense. Thus we consider this allocation strategy in the following discussion. In addition to the order optimality, the method is also fair in its implementation to all transactions over time.

B. Security Enhancement

From Alg. 2, we know that inconsistent peers are removed from consideration for data recovery. While Lem. 7 guarantees coverage in $2\frac{n}{m}$ slots, we are in fact interested in the number of slots for all uncorrupted peers to be paired with corrupt peers. We now give insight into the rate at which this happens.

We know that an adversary who wishes to corrupt a block corrupts at least $n/2$ nodes originally.

Lemma 8: Consider an adversary who successfully corrupts W_t to W'_t. Further, let us assume the adversary requires successful corruption with probability at least $1-\epsilon$, where $1-\epsilon > \frac{1}{p}$. Then under the cyclic zone allocation scheme, the adversary needs to corrupt at least m new nodes with probability at least $1-\frac{1}{p}$, in order to successfully alter H_t.

Proof: From the cyclic zone allocation strategy and the pigeonhole principle at least two honest nodes in the graph are paired with corrupt nodes in each slot. These nodes are to be corrupted by the adversary in order to preserve hash consistency if the corresponding hash value changes.

By the collision resistance property of the hash family, the probability that the corruption of a block W_t corrupts H_t is at least $1 - P_{\text{collision}} = 1 - \frac{1}{p}$. Thus the result follows. ■

Suppose the adversary corrupts a peer independently with probability $P_{\text{tc}} \in (0,1)$. This probability represents the ability of the adversary to corrupt other peers in the network.

Theorem 3: Let the consistency check depth be d. Then, the successful targeted corruption probability is

$$\mathbb{P}\left[\text{Targeted Corruption}\right] \leq \binom{r}{\frac{r}{2}} \rho^{\left(\frac{r}{2}\right)} \left[\frac{\left[1-\left(\rho(1-\frac{1}{p})\right)^d\right]}{p\left[1-(1-\frac{1}{p})\rho\right]} + (1-\tfrac{1}{p})^d \rho^d \right], \quad (17)$$

where $r = n/m$ and $\rho = P_{\text{tc}}^{2m}$.

The proof of the theorem is given in App. D. Naturally this implies that in the worst-case, with $\frac{n}{2m}$ transactions, the data becomes completely secure in the network. That is, only a corruption of *all peers* (not just a majority) leads to a consistent corruption of the transaction.

We now characterize the scaling of the probability of targeted corruption by its upper bound in the presence of an active adversary. We presume the probability of successful corruption of a node by the peer is P_{tc}. In Fig. 10 we note that the probability of targeted corruption decays exponentially with n, and thus, a large network is practically incorruptible.

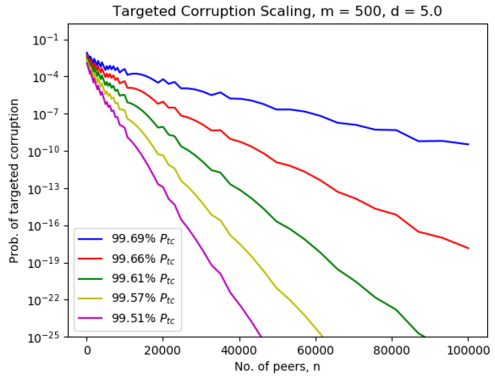

Fig. 10. Targeted corruption scaling with number of peers.

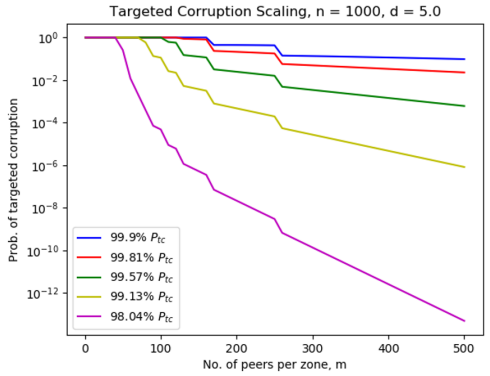

Fig. 11. Targeted corruption scaling with number of peers per zone.

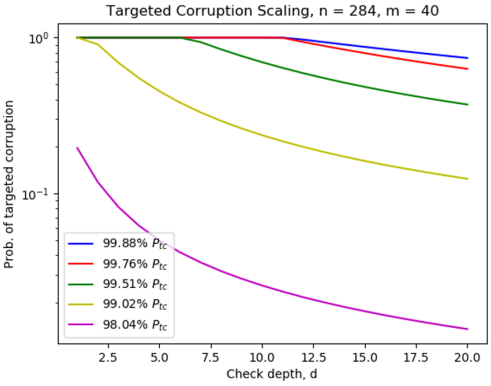

Fig. 12. Targeted corruption scaling with recovery depth.

From Fig. 11 we note that the probability of successful corruption decays with the number of peers per zone. Naturally, when each zone is comprised of more peers, the harder it is for the active adversary to corrupt the data. This is because corruption of a zone fails even if a single peer in the network refuses to corrupt the data as dictated by the adversary.

Finally, in Fig. 12 we study the targeted corruption probability as a function of the number of blocks, d, used to check data validity. For better exposition, we consider a network with $n = 284$ and $m = 40$. The probability decays with d, indicating that the more blocks we check for consistency, the harder it is for the adversary to corrupt the data block.

Thus, as can be noted here, it is beneficial to choose larger numbers of peers in each zone, and check deeper into the chain for consistency at recovery, as it makes the data more robust to targeted corruptions by active adversaries. On the contrary, as observed from Fig. 6, larger zone sizes result in higher data loss. Thus an operating point in this tradeoff must be chosen.

VII. Data Recovery and Repair

In this section we consider data recovery and repair. First, we derive necessary and sufficient conditions on code design to recover data stored on the ledger with a desired probability. Second, we briefly consider the problem of repairing a nodal failure by repurposing data from other zones and identify the limiting number of failures the system can handle.

A. Recovery Cost

The recovery of a transaction block or hash value requires the participation of all peers in the zones. Thus the cost of data recovery using the (Φ, Ψ)-encryption scheme is

$$R_r^{(t)} = C_r \left(\tfrac{1}{m} \log_2 q + m(2 \log_2 m + 1) + 4 d_t \log_2 p \right). \quad (18)$$

However, practical systems encounter inactive peers making data recovery from the corresponding zones infeasible. Consider a simple model in which peers are inactive with probability ρ, and the activity across slots and peers is independent and identically distributed. Then, the successful data recovery satisfies the following necessary and sufficient conditions.

Theorem 4: For any $\delta > 0$, the ideal zone size, m^*, such that the probability of successful recovery is at least $1 - \delta$ is

$$\frac{(\log n - \log\log(1/\delta))}{\log\left(\frac{1}{1-\rho}\right)} \leq m^* \leq \frac{(\log n - \log(1-\delta))}{\log\left(\frac{1}{1-\rho}\right)}. \quad (19)$$

The proof is given in App. E. This implies zones of size $O(\log n)$ are needed to satisfy the desired recovery probability.

B. Data Repair

The distributed secure storage ensures that individual entries stored at each peer cannot be recovered from the knowledge of other entries in the zone. Thus it is not feasible to repair nodes locally within a zone. However, it suffices to substitute a set of bits such that the code structure is retained.

A node failure indicates that the private key is lost. Thus repairing a node involves recoding the entire zone using data from a neighboring zone. Thus due to the encryption, it is difficult to repair nodes upon failure.

Thus, the transaction data is completely lost if one peer from each of the n/m zones undergoes failure. This emphasizes the need to ensure that m is small in comparison to n, so as to avoid data loss from node repair.

VIII. Blockchain-Based Cloud Storage

Blockchain-based storage systems are of interest due to the immutability guarantees on stored data. We now describe a scheme selection mechanism by which the client can opt for a service that best serves the data being stored [1]. Additionally, it is important to note that the clients can inherently value data differently by appropriately varying the code design.

A. Security-Based Scheme Selection

Consider a cloud storage system that implements our code to store data on the blockchain. Without loss of generality, let us assume the cost of storing one unit of data at all peers per unit time is one. The communication cost for data recovery is C_r. Let the frequency of data retrieval be ν, and let the parameters be m, d. Then, the storage cost per unit time is

$$\begin{aligned} \text{Service Cost} &= R_s + \nu R_r \\ &= \left(\tfrac{1}{m} \log_2 q + m \log_2 2m^2\right)(1 + \nu_r) \\ &\quad + \log_2 p^2 (1 + d\nu_r), \end{aligned} \quad (20)$$

which is obtained from (6) and (8) with $\nu_r = \nu C_r$.

Given the parameters, the data loss, targeted corruption, and the fraction of colluding peers for information leak are determined by the maxima of (13), (17), and f_{il} respectively. Thus, the client can choose the design parameters by solving

$$\begin{aligned} (m^*, d^*) \in \arg\min_{m,d} \; & R_s + \nu R_r \quad (21) \\ \text{such that} \quad & \mathbb{P}\left[\text{Data Loss}\right] \leq \delta_{\text{dl}}, \\ & \mathbb{P}\left[\text{Targeted Corruption}\right] \leq \delta_{tc}, \text{ and} \\ & f_{il} \geq \delta_{il}. \end{aligned}$$

Note that (21) is a non-linear integer program and presumes knowledge of the parameters that define the adversary strength. In general such integer programs are NP-hard, but the details of the optimization are beyond the scope of this work. However, note that the optimization is over the space of zone sizes m that scales logarithmically with n, and the optimal depth d which scales logarithmically with the targeted corruption probability. Alternately, we can consider the real-value relaxation of the parameters which can then be rounded, since we know the objective function and the probabilities of data loss and targeted corruption are monotonic. As such, this integer program can be solved easily in practice.

We now consider a simple context to evaluate the reduction in the costs from the use of the dynamic distributed storage scheme. Let the size of the data block be D bits, and let the hash be $H = 256$ bits long. Let the probability of independent node failure be 0.005, and consider a DoS adversary who corrupts 0.01 of the nodes of the network, and an active adversary who corrupts a peer successfully with probability 0.98. Let us presume that the desired probability of data loss is $\delta_{\text{dl}} = 0.005$ and the probability of targeted corruption is $\delta_{\text{tc}} = 0.001$. Consider the recovery cost parameters are such that $\nu R_r = 0.1$, i.e., recovery costs are 10% of storage costs.

Fig. 13 shows the optimal cost achieved under dynamic distributed storage. When n is small, the scheme uses smaller zone sizes to guarantee the target corruption probabilities, resulting in higher costs. However, as n increases, the cost decreases considerably. Further, with increasing block sizes, the gains from data sharing are also more pronounced.

As the number of peers increases, the optimal cost saturates. This is a result of the increased difficulty of data corruption with increasing n. We know from Thms. 2 and 3 that the probability of data loss and targeted corruption fall exponentially with n. Thus, when the number of peers is sufficiently large, the security requirements are easily met, effectively moving the bottleneck of (21) to the storage cost

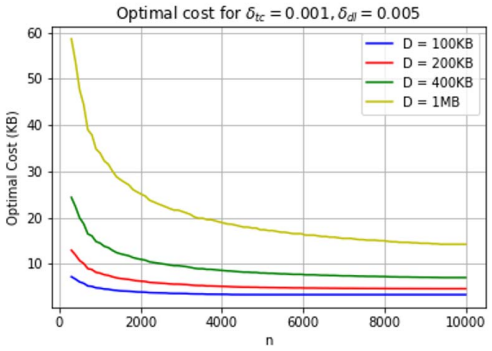

Fig. 13. Optimal cost resulting from the use of dynamic distributed storage. The larger the data block, the better the savings.

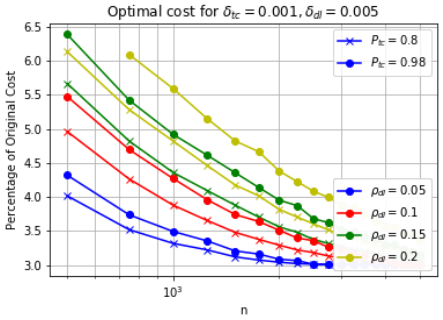

Fig. 14. Optimal cost resulting from the use of dynamic distributed storage. The larger the data block, the better the savings.

term $\left(\frac{1}{m}\log_2\ q + m(2\log_2\ m + 1)\right)$. This is minimized for $m = O(\sqrt{\log_2\ q})$ and hence the cost saturates to this limit.

Fig. 14 shows the effect of the strength of adversaries for blocks of size $D = 100\ KB$ and hashes of size 256 bits. Let the probability of independent node failure be 0.005 and let the desired probabilities of targeted corruption and data loss be 0.001 and 0.005 respectively. We consider active adversaries who corrupt a peer with probability $P_{\text{tc}} \in \{0.8, 0.98\}$, and DoS adversaries that corrupt ρ_{dl} fraction of peers in the network.

In the absence of the dynamic distributed storage, the effective cost of the system is given by

$$R_{base} = \log_2\ q(1 + \nu C_r) + 2\log_2\ p(1 + d\nu C_r)$$

Fig. 14 shows the resulting cost of dynamic distributed storage as a percentage of the original cost. Observe that the stronger the active adversary, the larger the effective cost of storage, since the system must employ smaller zones (small m^*) and deeper checks (large d^*) to prevent targeted corruption.

In this experiment, we have considered DoS adversaries who can corrupt upto 20% of the peers of the network. As the DoS adversary corrupts a larger share of the network, more copies of each data block are to be retained in the network, through the use of smaller zones. This in turn results in higher effective cost of the storage system. We again observe that at high n, the normalized cost of the network saturates.

Thus, dynamic distributed storage considerably reduces storage costs without compromising data security in the chain.

B. Data Insurance

Distributed storage on blockchains provide us with the opportunity to offer data insurance [42] for stored data due to the security guarantees. Here, we briefly describe code design to store data valued at a certain level such that the service provider on average obtains a certain desired profit margin.

Consider storing a data block valued by the client at V. Let $\mu \in [0, 1]$ be the profit margin desired by the service provider. Let $w_1 \in [0, 1]$ be the fraction of DoS adversaries, and $w_2 = 1 - w_1$ be the fraction of active adversaries. Here we do not consider information leak through collusion. Now, in any slot, the probability that the data is lost or successfully corrupted in a slot is given by

$$\theta = w_1\mathbb{P}\left[\text{Data Loss}\right] + w_2\mathbb{P}\left[\text{Targeted Corruption}\right].$$

Let $R = R_s + \nu R_r$ be the cost per unit time that the insured storage is priced at. The time T for data loss or successful corruption is distributed geometrically with the parameter θ. Then the expected time for the loss of insured data is $\frac{1}{\theta}$. Thus, the service provider can design the code according to

$$(m^*, d^*) \in \arg\min_{m,d} R, \quad \text{such that } R \geq (1 + \mu)V\theta. \quad (22)$$

Again this is a non-linear integer program that is to be solved to obtain the desired profit margin on insured data blocks. As described in the previous subsection, this integer program can be solved easily as the parameters scale logarithmically with the network size and security requirements.

Thus, the code provides the opportunity to design efficient data storage and insurance mechanisms over the blockchain. A variety of other applications may also be developed by studying the corresponding tradeoff in operational costs, required security and privacy, and the requirements of the application.

IX. Conclusion

This paper introduced a simple mathematical model of blockchain systems and leveraged information-theoretic secrecy, distributed storage coding, and a novel grouping mechanism to allow efficient cold storage of blockchain ledgers in the presence of active adversaries. We also studied the storage and recovery costs, and the data security and privacy guarantees. We believe such methods enhance blockchain systems to a scale to address practical applications in a variety of industries.

In the future, a closer coding-theoretic analysis can help develop efficient and robust codes such as [29], [43], [44]. Optimizing the code construction with regard to the network costs of communication and security, can further help develop economical practical implementations of the scheme.

References

[1] R. K. Raman and L. R. Varshney, "Distributed storage meets secret sharing on the blockchain," in *Proc. Inf. Theory Appl. Workshop (ITA)*, Feb. 2018, pp. 1–6.

[2] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for blockchains," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2619–2623.

[3] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 104–121.

[4] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ, USA: Princeton Univ. Press, 2016.

[5] D. Tapscott and A. Tapscott, *Blockchain Revolution: How Technology Behind Bitcoin is Changing Money, Business, and the World*. New York, NY, USA: Penguin, 2016.

[6] S. Underwood, "Blockchain beyond Bitcoin," *Commun. ACM*, vol. 59, no. 11, pp. 15–17, 2016.

[7] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Aug. 2016, pp. 25–30.

[8] M. J. Casey and P. Wong, "Global supply chains are about to get better, thanks to blockchain," *Harvard Bus. Rev.*, vol. 3, pp. 1–6, Mar. 2017. [Online]. Available: https://hbr.org/2017/03/global-supply-chains-are-about-to-get-better-thanks-to-blockchain

[9] M. Swan, *Blockchain: Blueprint for a New Economy*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[10] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Bus. Rev.*, vol. 95, no. 1, pp. 118–127, 2017.

[11] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 839–858.

[12] *Blockchain Info*. Accessed: Feb. 25, 2021. [Online]. Available: https://blockchain.info/home

[13] K. Croman *et al.*, "On scaling decentralized blockchains," in *Financial Cryptography Data Security* (Lecture Notes in Computer Science), vol. 9604, J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Germany: Springer, 2016, pp. 106–125.

[14] (Oct. 2015). *SETL Breaks Through 1 Billion Transactions per day on Blockchain*. [Online]. Available: https://setl.io/blog/setl-breaks-through-1-billion-transactions-per-day-on-blockchain/

[15] N. Kshetri and J. Voas, "Blockchain in developing countries," *IT Prof.*, vol. 20, no. 2, pp. 11–14, Mar. 2018.

[16] V. Thakur, M. N. Doja, Y. K. Dwivedi, T. Ahmad, and G. Khadanga, "Land records on blockchain for implementation of land titling in India," *Int. J. Inf. Manage.*, vol. 52, Jun. 2020, Art. no. 101940.

[17] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–5.

[18] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[19] A. G. Dimakis and K. Ramchandran, "Network coding for distributed storage in wireless networks," in *Networked Sensing Information Control*, V. Saligrama, Ed. New York, NY, USA: Springer, 2008, pp. 115–136.

[20] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 773, D. R. Stinson, Ed. Berlin, Germany: Springer, 1994, pp. 136–146.

[21] M. O. Rabin, "The information dispersal algorithm and its applications," in *Sequences*, R. M. Capocelli, Ed. New York, NY, USA: Springer-Verlag, 1990, pp. 406–419.

[22] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *Proc. 24th IEEE Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2005, pp. 191–201.

[23] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. 47th Annu. Allerton Conf. Commun. Control Comput.*, Sep. 2009, pp. 1243–1249.

[24] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 212–236, Jan. 2014.

[25] S. Pawar, S. El Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," *IEEE Trans. Inf. Theory*, vol. 57, no. 10, pp. 6734–6753, Oct. 2011.

[26] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded Merkle tree: Solving data availability attacks in blockchains," 2019, *arXiv:1910.01247*. [Online]. Available: http://arxiv.org/abs/1910.01247

[27] S. Li, M. Yu, C.-S. Yang, A. Salman Avestimehr, S. Kannan, and P. Viswanath, "PolyShard: Coded sharding achieves linearly scaling efficiency and security simultaneously," 2018, *arXiv:1809.10361*. [Online]. Available: http://arxiv.org/abs/1809.10361

[28] S. Kadhe, J. Chung, and K. Ramchandran, "SeF: A secure fountain architecture for slashing storage costs in blockchains," 2019, *arXiv:1906.12140*. [Online]. Available: http://arxiv.org/abs/1906.12140

[29] Y. Kim, R. K. Raman, Y.-S. Kim, L. R. Varshney, and N. R. Shanbhag, "Efficient local secret sharing for distributed blockchain systems," *IEEE Commun. Lett.*, vol. 23, no. 2, pp. 282–285, Feb. 2019.

[30] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *Fast Software Encryption*, B. Roy and W. Meier, Eds. Berlin, Germany: Springer, 2004, pp. 371–388.

[31] D. R. Stinson, "Some observations on the theory of cryptographic hash functions," *Des., Codes Cryptogr.*, vol. 38, no. 2, pp. 259–277, Feb. 2006.

[32] R. J. McEliece and D. V. Sarwate, "On sharing secrets and Reed–Solomon codes," *Commun. ACM*, vol. 24, no. 9, pp. 583–584, Sep. 1981.

[33] E. Karnin, J. Greene, and M. Hellman, "On secret sharing systems," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 1, pp. 35–41, Jan. 1983.

[34] W. Huang, M. Langberg, J. Kliewer, and J. Bruck, "Communication efficient secret sharing," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 7195–7206, Dec. 2016.

[35] J. L. Massey, "Minimal codewords and secret sharing," in *Proc. 6th Joint Swedish-Russian Int. Workshop Inf. Theory*, Aug. 1993, pp. 276–279.

[36] C. E. Shannon, "Communication theory of secrecy systems," *Bell Labs Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[37] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.

[38] K. W. Shum, "Cooperative regenerating codes for distributed storage systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2011, pp. 242–249.

[39] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1837–1852, Mar. 2012.

[40] R. Durrett, *Random Graph Dynamics*. Cambridge, U.K.: Cambridge Univ. Press, 2007.

[41] Z. Baranyai, "The edge-coloring of complete hypergraphs I," *J. Comb. Theory, Ser. B*, vol. 26, no. 3, pp. 276–294, 1979.

[42] X. Ma, "On the feasibility of data loss insurance for personal cloud storage," in *Proc. 6th USENIX Conf. Hot Topics Storage File Sys.*, Jun. 2014, pp. 1–5.

[43] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Aug. 2012.

[44] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes via rank-metric codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 1819–1823.

Ravi Kiran Raman received the B.Tech. degree in electrical engineering and the M.Tech. degree in communication systems from Indian Institute of Technology Madras, Chennai, India, in 2014, and the M.S. degree in mathematics and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, IL, USA. He is currently a Research Scientist with the Algorithmic Systems Group, Analog Devices Inc., Boston.

Lav R. Varshney (Senior Member, IEEE) received the B.S. degree *(magna cum laude)* in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2004, and the S.M., E.E., and Ph.D. degrees in electrical engineering and computer science from Massachusetts Institute of Technology, Cambridge, MA, USA, in 2006, 2008, and 2010, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign. His research interests include artificial intelligence and blockchain.