Dynamic Distributed Storage for Blockchains

Ravi Kiran Raman and Lav R. Varshney
University of Illinois at Urbana-Champaign

***Abstract*—Blockchain uses the idea of storing transaction data in the form of a distributed ledger wherein each node in the network stores a current copy of the sequence of transactions (ledger) in the form of a hash chain. 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 storage cost to a fraction of the original. When further using dynamic zone allocation, we show the coding scheme can also improve the data integrity.**

A full version of this paper is accessible at: [1]

I. Introduction

Blockchains are distributed, shared ledgers of transactions that reduce the friction in financial networks caused by intermediaries using different technology infrastructures. This has created a new environment of business transactions and self-regulated cryptocurrencies such as bitcoin [2], [3]. Owing to such favorable properties, blockchains are being adopted extensively outside cryptocurrencies in a variety of novel application domains such as medicine, supply chain management, global trade, and government services [4]. Blockchains are expected to revolutionize the way financial/business transactions are done, such as through smart contracts [5], [6].

However, blockchain works on the premise that every peer stores the entire ledger of transactions in the form of a hash chain, even though they are meaningless to peers that are not party to the transaction. Consequently, individual nodes incur a significant, ever-increasing storage cost [7]. Note that *secure* storage may be much more costly than raw hard drives, e.g. due to infrastructure and staffing costs. In current practice, the most common technique for storage reduction is to prune old transactions. However, this is not sustainable for blockchains with a high data arrival rate and thus does not scale.

For instance, bitcoin currently serves an average of just under 3.5 transactions per second [8], translating to about 60GB per year [7]. The superlinear growth in storage in bitcoin [1], highlights the need for better storage techniques. SETL, a blockchain-based payment and settlement platform, claims to support 1 billion transactions per day. This is dwarfed by the Federal Reserve, which processes 14 *trillion* transactions per day. If cryptocurrencies are to become financial mainstays, they would need to scale by several orders [8], amounting to a daily average over 90GB. This is to say nothing about uses for blockchain in global trade, healthcare, agriculture, and various other industries. With storage costs expected to saturate due to the ending of Moore's Law, storage is a pressing concern for the large-scale adoption of blockchain.

This impending end to Moore's law also results in a saturation of computational speeds. Notwithstanding new efforts [9], block validation (mining) in bitcoin-like networks involves an expensive hash computation stage. This requires high-end hardware and much energy. Recent studies have estimated that global energy consumption of bitcoin is of the order of 700MW [10] – enough to power over $325{,}000$ homes, and over 5000 times the energy per transaction on a credit card.

Distributed storage schemes have been considered in the past as information dispersal algorithms (IDA) [11] and as distributed storage codes [12]. In particular [13] considers an IDA secure from adaptive adversaries. Secure distributed storage codes to protect against colluding eavesdroppers [14] and active adversaries [15] have also been studied. The difference in the nature of attacks by adversaries in blockchain calls for a new and stronger coding scheme.

In Sec. III, we use a novel combination of Shamir's secret sharing scheme [16], private key encryption, and distributed storage [12], inspired by [17], to construct a coding scheme that distributes data among subsets of peers. The scheme is optimal in storage up to small additive terms. We also show, using a combination of statistical and cryptographic security, that the distributed storage loses an arbitrarily small amount of data integrity as compared to convention. Further, in Sec. IV, using dynamic zone allocation among peers, we show that the integrity data can be further enhanced.

II. System Model

We now introduce a mathematical model for blockchain.

A. Ledger Construction

Blockchain comprises a connected peer-to-peer network of nodes from three primary categories based on functionality:

1) **Clients:** invoke transactions, have them validated by endorsers, and communicate them to orderers.
2) **Peers:** commit transactions and maintain the ledger.
3) **Orderer:** communicate transactions to peers in chronological order to ensure consistency of the hash chain.

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

A transaction is initiated by participating clients, verified by endorsers (select peers), and broadcast to peers through orderers. The blockchain architecture is shown in Fig. 1.

The ledger is stored as a (cryptographic) hash chain.

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* is a deterministic map $h : \mathcal{M} \to \mathcal{H}$. Good hash functions hold several salient properties such as

978-1-5386-4780-6/18/$31.00©2018 IEEE

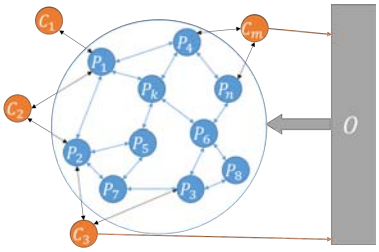

Fig. 1: Blockchain network: clients C_i, peers P_i, orderers O.

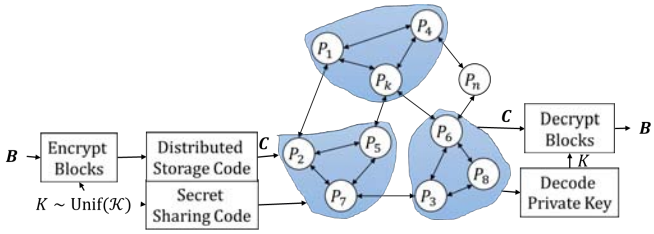

Fig. 2: Encoding and decoding for a given zone allocation. Shaded regions represent individual zones.

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 $h(\mathcal{M}) = H$.
3) **Collision resistance:** It is computationally infeasible to find $M_1, M_2 \in \mathcal{M}$ such that $h(M_1) = h(M_2)$.
4) **Sensitivity:** Change in hash to minor input changes are computationally inestimable.

Let $\mathbf{B}_t$ be the tth data block and let $H_t = h(W_t)$ be the hash value stored with the $(t+1)$th transaction, where $W_t = (H_{t-1}, \mathbf{B}_t)$ is the concatenation. The hash chain stored by each peer is $(H_0, \mathbf{B}_1) - (H_1, \mathbf{B}_2) - \cdots - (H_{t-1}, \mathbf{B}_t)$. For all t, let $\mathbf{B}_t \sim \text{Unif}(\mathbb{F}_q)$ and $H_t \in \mathbb{F}_p$, where $q, p \in \mathbb{N}$ and $\mathbb{F}_q, \mathbb{F}_p$ are finite fields of order q and p respectively. Thus, in conventional implementation, storage cost per peer per transaction is

$$\tilde{R}_s = \log_2 q + \log_2 p \text{ bits.} \tag{1}$$

The transaction recovery methodology usually varies with application. For uniformity, we consider a method wherein all peers return stored data and a majority vote consensus is used.

B. Blockchain Security

We focus on two main aspects of data security. First we expect data *integrity*, making it incorruptible unless a majority of peers is corrupted. Further, we also expect *confidentiality* so that local data leak does not reveal sensitive information.

Blockchains are valued for their *immutability* as data corruption requires not just corrupting a majority of peers, but also to ensure chain consistency by altering succeeding hash values. Bitcoin-like systems enforce additional constraints on hash values to enhance data integrity. Such constraints make the computation of a new hash expensive.

Confidentiality is typically guaranteed through private key encryption. However, a leak at a single node could lead to complete disambiguation.

C. Active Adversary Model

Here, we focus on *active adversaries* who alter a transaction content B_t to some B'_t. We define the semantic rules of a valid corruption. An adversary corrupting a peer can

1) learn the contents stored in the peer;
2) alter block content only if it has access to the block; and
3) alter hash values preserving chain integrity, i.e., attackers cannot invalidate other transactions in the process.

We presume the adversary is computationally limited from exhaustive message/hash search.

Algorithm 1 Encoder $\{\mathbf{C}_t^{(z)}, \mathbf{K}_t^{(z)} : z \in [n']\} = \text{ENC}(\mathbf{B}_t, \mathcal{P}_t)$

1: **for** $z = 1$ to n' **do**
2: Generate private key $K_t^{(z)} \sim \text{Unif}(\mathcal{K})$
3: Encrypt block with key $K_t^{(z)}$ as $\mathbf{C}_t^{(z)} = \Phi(\mathbf{B}_t; K_t^{(z)})$
4: Distribute $\mathbf{C}_t^{(z)}$ and store among peers in $\{i : p_t^{(i)} = z\}$
5: Generate shares $\mathbf{K}_t^{(z)}$ using Shamir's (m, m)-secret sharing code on $K_t^{(z)}$, for peers of zone z
6: **end for**

Another attack of interest is *denial of service* wherein an adversary blocks a peer from revealing stored data. We briefly describe the vulnerability of the system to such attacks.

III. Coding Scheme

At any time t, there exists a partition $\mathcal{P}_t$ of peers $[n]$ into sets of size m each, called *zones*, defined according to Sec. IV. Let $n' = \frac{n}{m}$, and let n be divisible by m. Let $p_t^{(i)} \in [\frac{n}{m}]$ be the zone index for peer i at time t. Encoding and decoding are performed by orderers, treated here as a black box. Details of this process are omitted here for brevity.

A. Coding Data Block

In our coding scheme, a single copy of each data block is stored in a distributed fashion across each zone. 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 [16]. Finally, the encrypted data block is distributed amongst peers in the zone using a distributed storage scheme. The encoding and decoding processes are shown in Fig. 2.

In this work, we presume that the secret shares in Shamir's secret sharing scheme are generated by evaluating the polynomial at non-zero abscissa values chosen uniformly at random without replacement from the corresponding field.

The coding scheme is given by Alg. 1. For ease, assume that each peer stores a component of the code vector $\mathbf{C}_t$. The theory extends naturally to other distributed storage schemes.

The hash values are stored using Shamir's (m, m)-secret sharing scheme as well, i.e., at time t, each peer in the zone stores a secret share of the hash value H_{t-1}.

The storage per transaction per peer is thus

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

Algorithm 2 Decoder, $\mathbf{B} = \text{DEC}(\{\mathbf{L}^{(z)} : z \in [n']\})$

$\mathcal{N} \leftarrow [n]$
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 $z \in [n']$
if $|\{B_t^{(z)} : z \in [n']\}| > 1$ **then**
  for $\tau = t$ to T **do**
    Compute $H_\tau^{(z)}$, for all z, by polynomial interpolation
    Determine $W_\tau^{(z)} = \left(B_\tau^z, H_{\tau-1}^z\right)$, for all $z \in [n']$
    $\mathcal{I} \leftarrow \left\{i : h(W_\tau^{(z)}) \neq H_\tau^{(z')}, z = p_\tau^{(i)}, z' = p_{\tau+1}^{(i)}\right\}$
    $\mathcal{N} \leftarrow \mathcal{N} \backslash \mathcal{I}$
    if $\left|\left\{B_t^{(p_t^{(i)})} : i \in \mathcal{N}\right\}\right| = 1$ **then**
      break
    end if
  end for
end if
return Majority in $\{\{B_t^{(p_t^{(i)})} : i \in \mathcal{N}\}\}$

where $|\mathcal{C}|$ depends on the rate of the code and the encryption scheme used. If the keys are much smaller than the blocks, we have considerable storage savings.

B. Recovery Scheme

To recover B_t, when there are T transactions in total, we use Alg. 2. First, data blocks are recovered from each zone. In case of mismatch, we check hash chain consistencies and eliminate inconsistent peers. An inconsistency exists when the hash corresponding to data stored by a node in the previous instance does not match the current hash value. Finally, the majority of consistent data is returned.

We presume that all computation for recovery is done privately by a black box. That is, the peers and clients are not made aware of the code. Specifics of practical implementation of such a black box scheme is beyond the scope of this paper.

C. Feasible Encryption Scheme

In order to secure data from active adversaries who are aware of the block, we require the encryption and decryption algorithms Φ, Ψ to satisfy the following criteria

$$\max_{\mathbf{B} \in \mathbb{F}_q} \mathbb{P}\left[\Psi(\mathbf{C}, K) = M\right] \leq \epsilon \tag{3}$$

$$\mathbb{P}\left[\Phi(\mathbf{B}; K) = \mathbf{C} | \mathbf{B}, \mathbf{C}_{-j}\right] \leq \tfrac{1}{2}, \text{ for any } C_j, \tag{4}$$

where $\epsilon > \frac{1}{|\mathcal{B}|}$ is a chosen security level, $C_{-j} = \{C_i : i \neq j\}$. We relax the requirement in (4) since perfect security requires a one-time pad that is expensive to store.

Lemma 1: For any valid encryption with (3), (4), $|\mathcal{K}| \geq 2^m$. Proof is from uniform and independent key generation [1].

We now describe an encryption scheme, order-optimal in key size upto log factor. Let $\mathcal{T}$ be the set of all rooted, connected trees on m nodes. Define the key space by the lossless compression of $T \sim \text{Unif}(\mathcal{T})$, i.e., given K, we are aware of 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.

Algorithm 3 Encryption scheme, $\mathbf{C} = \Phi(\mathbf{B}, K)$

$T \leftarrow \text{Unif}(\mathcal{T})$, $K \leftarrow \text{Key}(T)$; $\mathbf{b} \leftarrow \text{Binom}(m, 1/2)$
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

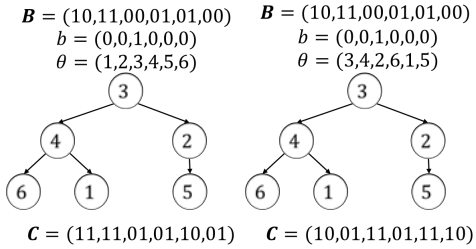

Fig. 3: Encryption examples for zone with 6 peers. Data block, parameters, tree structure, and corresponding codes are shown.

Consider the encryption function given in Alg. 3. The algorithm proceeds by first selecting a rooted, connected tree uniformly at random. Then, each peer is assigned to a vertex at random. For each node other than the root, the codeword is created as the mod-2 sum of the corresponding data block and that corresponding to its parent. Finally, the root is encrypted as the mod-2 sum of all codewords at other vertices and the data block of the root. The bits stored at the root are flipped with probability half. The encryption scheme for a sample data block is shown in Fig. 3. We refer to Alg. 3 as Φ from here on. The corresponding decryption algorithm, Ψ, is Alg. 4.

Lemma 2: The encryption scheme Φ satisfies (3) and (4). The proof follows from the security offered by Shamir's secret sharing in storing the private key, and is given in [1].

Lemma 3: Storage per peer per transaction under (Φ, Ψ) is

$$R_s(\Phi, \Psi) = \tfrac{1}{m} \log_2 q + m(2 \log_2 m + 1) + 2 \log_2 p \text{ bits} \tag{5}$$

The proof follows from (2) and Cayley's formula [18] as shown in [1]. Since blocks are drawn uniformly at random, we note that the scheme yields order-optimal storage cost per peer per transaction up to log factor in key size. The security can be enhanced by increasing inter-data dependency, for instance using directed acyclic graphs (DAGs) with bounded in-degree instead of rooted trees, or using stronger encryption methods.

D. Individual Block Corruption

We now consider security of individual blocks in each zone.

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

Algorithm 4 Decryption scheme, $\mathbf{B} = \Psi(\mathbf{C}, K)$

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 result follows from Shamir's coding scheme and the proof can be found in [1]. That is, in order to corrupt a hash value, the adversary practically needs to corrupt all nodes in the zone.

Lemma 5: Consider an adversary, aware of $\mathbf{B}$ and the peers in a zone. If the adversary corrupts $c < m$ peers, then

$$\mathbb{P}\left[\mathbf{B} \to \mathbf{B}' \text{ in zone } z\right] \leq \tfrac{c}{m}, \text{ for all } \mathbf{B} \neq \mathbf{B}', z \in [n']. \quad (6)$$

The proof can be found in [1]. Successful corruption requires corruption of at least $\frac{n'}{2}$ zones to achieve simple majority.

Theorem 1: Consider an active adversary who corrupts $c_1, \ldots, c_{n/2m}$ peers respectively in each zones. Then,

$$\mathbb{P}\left[\text{Successful corruption } \mathbf{B} \to \mathbf{B}'\right] \leq \exp\left(n' \log\left(\frac{2\sum_{i=1}^{n'/2} c_i}{n}\right)\right), \text{ for all } \mathbf{B} \neq \mathbf{B}'.$$

The proof follows from Lem. 5 and can be found in [1]. Note that $\sum_{i=1}^{n/2m} c_i \leq \frac{n}{2}$, and thus the bound decays exponentially with the network size, if less than half of it is corrupted. From Thm. 1, we immediately get the following.

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 nodes to be corrupted satisfies

$$\sum_{i=1}^{n/2m} c_i \geq \frac{n}{2}(1-\epsilon)^{\frac{m}{n}}. \quad (7)$$

By Cor. 1, when the network is large, an adversary practically needs to corrupt at least half the network to have a sufficiently high probability of successful corruption. Thus, for a fixed zone division, the distributed storage loses an arbitrarily small amount of integrity as compared to the conventional scheme.

With regard to denial of service attacks, since it is infeasible to recover the data block without shares from all peers in the zone, the system is capable of handling upto 1 denial of service attack per zone, i.e., a total of n' such attacks.

E. Data Confidentiality

Assume that an external adversary receives all data stored by the ith peer for a slot, including 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. Since the block data are chosen uniformly at random,

$$H(\mathbf{B}|\mathbf{C}) \leq H(K, \mathbf{B}|\mathbf{C}) = H(K). \quad (8)$$

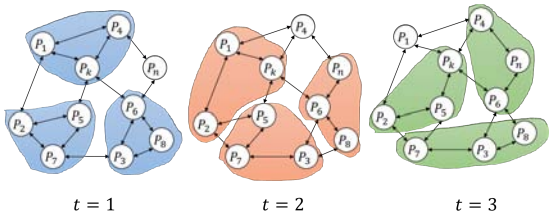

Fig. 4: Dynamic zone allocation: vary peer grouping over time.

The entropy of the data block is actually $H(\mathbf{B}) = \log_2 q > H(K)$. However, the knowledge of just C_i does not reveal any of the other components. Thus, local leaks reveal very little information regarding the data block. Thus, we also ensure a high degree of confidentiality.

IV. Dynamic Zone Allocation

We now describe the zone allocation strategy. Consider a blockchain in the state $(H_0, \mathbf{B}_1) - \cdots - (H_{t-1}, \mathbf{B}_t)$. Let us assume an adversary wishes to corrupt $\mathbf{B}_1$ to $\mathbf{B}'_1$. Then $(H_0, \mathbf{B}'_1) - \cdots - (H'_{t-1}, \mathbf{B}_t)$ is the consistent version of the corrupted chain. If we vary the zone allocation in each slot, then each uncorrupted peer eventually pairs with a corrupt one.

For any τ, to successfully alter $H_{\tau-1}$ to $H'_{\tau-1}$ in a zone, the adversary needs to corrupt the uncorrupted peers in the zone. On the other hand, if the client does not corrupt these nodes, the hash remains unaltered indicating inconsistency.

It is thus evident that if the zones are altered sufficiently often, 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. 4.

A sufficient condition to ensure that every corrupted peer is eventually grouped with an uncorrupted peer is to ensure that every peer is eventually grouped with every other peer. Additionally, the system needs to ensure uniform security for every transaction, i.e., the allocation needs to be fair.

A. m-way Handshake Problem

The sufficient condition is equivalent to the following m-way handshake problem. Consider a group of n people. At each time slot, they are grouped into sets of size m such that they shake hands with all peers in the set.

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 as we have at most $m-1$ new pairings per slot.

We now design the zone allocation strategy. Partition peers into $2n'$ sets of size $\frac{m}{2}$, given by $\nu_1, \ldots, \nu_{2n'}$. We use matchings of $K_{2n'}$, as determined by Alg. 5 to perform the zone allocation. This is depicted in Fig. 5.

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

The result follows from the construction. This scheme matches the lower bound on the number of slots for coverage in the order sense. Thus we use this strategy cyclically for the dynamic zone allocation as it is also fair.

Algorithm 5 Dynamic Zone Allocation Strategy

Let $\nu_2 \ldots, \nu_{2n'}$ be vertices of a $2n'-1$ regular polygon, and ν_1 its center
for $i = 2$ to $2n'$ **do**
  Let L be the line passing through ν_1 and ν_i
  $M \leftarrow \{(\nu_j, \nu_k) :$ line $\nu_j - \nu_k$ is perpendicular 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

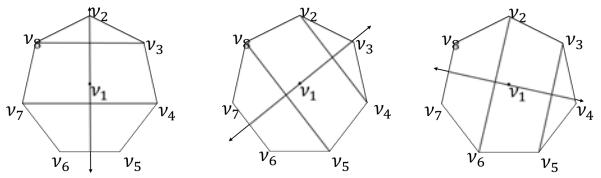

Fig. 5: Dynamic Zone Allocation for $n = 4m$. The policy cycles through matchings by viewing them as a regular polygon.

B. Security Enhancement

We are interested in the number of slots for all uncorrupted peers to be paired with corrupt peers. Any adversary corrupts at least $n/2$ peers in the first slot.

Lemma 8: Under the cyclic zone allocation strategy, the adversary needs to corrupt at least m new nodes with each following transaction for successful corruption.

The proof follows from pigeonhole principle and is given in [1]. Thus, with $\frac{n'}{2}$ transactions only a corruption of *all peers* (not just majority) leads to a successful corruption.

V. Data Recovery and Repair

We briefly study data recovery and repair for the coding scheme. A more detailed study can be found in [1].

A. Recovery Cost

In practice, peers may be temporarily inactive or undergo data failure, making recovery from the corresponding zone infeasible. Consider a simple model wherein a peer is inactive in a slot independent of other peers with probability ρ.

Theorem 2: For any $\delta > 0$, probability of recovery of a block in any slot is at least $1-\delta$ if and only if $m = \Theta(\log n)$

The proof follows from the fact that we require all peers in at least one zone to be active and is given in [1]. Thus zone sizes have to be of the order of $\log n$ to guarantee a desired probability of recovery when node failures are possible.

B. Data Repair

Individual entries stored at a peer can not be recovered from the knowledge of other entries in the zone implying infeasibility of local recovery. 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 owing to the encryption, it is difficult to repair nodes upon failure. Thus, the system can handle up to n' node failures.

VI. Conclusion

In this work, we used a novel combination of secret key sharing, private key encryption, and distributed storage to design a secure distributed storage code for blockchains. Further, using dynamic zone allocation, we enhanced the integrity under a combination of cryptographic and information-theoretic security. We also highlighted the drawback of the system with respect to data repair and recovery. Improved energy efficiency of the scheme is described in [1].

Future work would focus on the design of specific codes that leverage the cyptographic security guarantees of the hash chain. Additionally, reducing the rate of the codes used, we can incorporate erasure tolerance and repair capabilities at the expense of increased storage cost.

References

[1] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for scaling blockchains," arXiv:1711.07617, Nov. 2017.
[2] 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. 2015 IEEE Symp. Security Privacy*, May 2015, pp. 104–121.
[3] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton: Princeton University Press, 2016.
[4] D. Tapscott and A. Tapscott, *Blockchain Revolution*. New York: Penguin, 2016.
[5] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Bus. Rev.*, vol. 95, no. 1, pp. 118–127, Jan. 2017.
[6] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. 2016 IEEE Symp. Security Privacy*, May 2016, pp. 839–858.
[7] Blockchain info. [Online]. Available: https://blockchain.info/home
[8] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin: Springer, 2016, vol. 9604, pp. 106–125.
[9] M. Vilim, H. Duwe, and R. Kumar, "Approximate bitcoin mining," in *Proc. 53rd Des. Autom. Conf. (DAC '16)*, Jun. 2016, pp. 97:1–97:6.
[10] P. Fairley, "Blockchain world - feeding the blockchain beast if bitcoin ever does go mainstream, the electricity needed to sustain it will be enormous," *IEEE Spectr.*, vol. 54, no. 10, pp. 36–59, Oct. 2017.
[11] M. O. Rabin, "The information dispersal algorithm and its applications," in *Sequences*, R. M. Capocelli, Ed. New York: Springer-Verlag, 1990, pp. 406–419.
[12] A. G. Dimakis and K. Ramchandran, "Network coding for distributed storage in wireless networks," in *Networked Sensing Information and Control*, V. Saligrama, Ed. New York: Springer, 2008, pp. 115–136.
[13] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symp. Rel. Distrib. Sys. (SRDS'05)*, Oct. 2005, pp. 191–201.
[14] 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.
[15] 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.
[16] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[17] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology — CRYPTO '93*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed. Berlin: Springer, 1994, vol. 773, pp. 136–146.
[18] R. Durrett, *Random Graph Dynamics*. Cambridge: Cambridge University Press, 2007.