Distributed Storage Meets Secret Sharing on the Blockchain

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

Abstract—**Blockchain systems establish a cryptographically secure data structure for storing data in the form of a hash chain. We use a novel combination of distributed storage, private key encryption, and Shamirs secret sharing scheme to distribute transaction data, without significant loss in data integrity. Additionally, using Shamirs secret sharing scheme on the hash values and dynamic zone allocation, we further enhance the integrity. We highlight the tradeoff in storage cost and data loss probability with varying zone size choices. We also study the tradeoff between recovery cost and security from adversarial corruption with varying recovery mechanisms. Then, we formulate code design, given a probability of data recovery and targeted corruption, as an integer program. Using the coding scheme we establish a mechanism to insure data, for instance in blockchain-based cloud storage systems, based on the value of the data, by understanding the costs involved for the service provider.**

I. Introduction

The invention of bitcoin almost a decade ago brought blockchains into prominence in the business world. Blockchains maintain a shared version of a transaction ledger with each peer in the network storing a copy reducing the friction in financial networks caused by intermediaries using different technology infrastructures. The technology has created a new environment of business transactions and self-regulated cryptocurrencies [1], [2].

Owing to their 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 [3]. Blockchains are expected to revolutionize the way financial/business transactions are done, such as through smart contracts [4], [5]. More recently, cloud storage systems such as STORJ and SIA have been designed using blockchain, offering heightened security guarantees and a new approach to decentralized storage.

However, blockchain works on the premise that every peer stores the entire transaction ledger as a hash chain, even though the data is meaningless to peers that are not party to the transaction. Consequently, individual nodes incur a significant, ever-increasing storage cost [6]. Note that *secure* storage may be much more costly than raw hard drives, e.g. due to infrastructure and staffing costs. 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 work was supported in part by the IBM-Illinois Center for Cognitive Computing Systems Research (C3SR), a research collaboration as part of the IBM AI Horizons Network.

This impending end to Moore's law also results in a saturation of computational speeds. Notwithstanding new efforts [7], block validation (mining) in bitcoin-like networks involves an expensive hash computation stage that requires high-end hardware and much energy. Recent studies have estimated that global energy consumption of bitcoin is of the order of 700MW [8] – enough to power over 325,000 homes, and over 5000 times the energy per transaction on a credit card.

Distributed storage has been considered in the past as information dispersal algorithms (IDA) [9] and as distributed storage codes [10]. In particular [11] considers an IDA secure from adaptive adversaries. Secure distributed storage codes to protect against colluding eavesdroppers [12] and active adversaries [13] have also been studied. Since we consider data confidentiality and integrity under active adversaries, we use a different approach to distribute the data.

To address rising storage costs and increasing transaction volumes [14], we proposed secure, distributed storage [15]. We also showed that by removing constraints on hash values, mining can be made energy efficient without loss in security. Our past work invoked a combination of distributed storage codes [10], private key encryption, and secret key sharing [16], inspired by [17], to distribute data among peers.

The construction of such a code results in tradeoffs not only between storage and recovery costs, but also among the associated integrity and confidentiality guarantees of the system. This work aims to build on the earlier work to explicitly study blockchain systems that are primarily used for archival storage of data. In Sec. III, by decoupling the distributed storage of transaction data from the dynamic zone allocation and secure storage of hash values, we enable the service provider or client to select the coding parameters based on the data and the security guarantees required.

In Sec. IV, we study the effects of denial of service and targeted corruptions on data loss and compute the probabilities of such corruptions. We also study the confidentiality of the data stored by the system, determining the minimum extent of collusion under which a data leak is feasible. We elaborate how the storage and recovery costs depend on the coding parameters as selected in Sec. III-A and III-B respectively.

Having established these tradeoffs, we first consider a model cloud storage system in Sec. V. For given integrity and security guarantees, we establish the code parameter selection as an integer program that minimizes the cost of storage. At the other end, we consider the idea of data insurance [18]. Given a profit margin desired by the service provider, we formulate the code design problem as an integer program minimizing service cost subject to that profit margin.

II. System Model

We now abstract blockchain systems through a mathematical model for the peer network and hash chain.

A. Ledger Construction

Blockchain comprises a connected peer-to-peer network of nodes with three primary functional categories:

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.

These classifications are only function-based, and individual nodes can serve multiple roles across transactions.

A transaction and the nature of the data associated with it is application-specific such as proof of fund transfer across clients in bitoin-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 categories.

A transaction is initiated by participating clients, verified by endorsers (select peers), and broadcast to peers through orderers. 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

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 changes in the input are computationally indeterminable.

A hash chain is a sequence of data blocks such that each block includes a header, which is the hash value of the previous (header included) block.

Bitcoin-type blockchains use the hash chain structure but store the individual transaction data as a Merkle tree, with the hash chain constructed using the Merkle root as the data in the block [14]. We consider a simpler form of this wherein the hash chain and verification is performed using hash values of original transactions that form the hash chain. Let us elaborate.

Let $\mathbf{B}_t$ be the data block corresponding to the tth transaction. Let g, h be two hash functions. Let $W_t = (H_{t-1}, g(\mathbf{B}_t))$ is the concatenation of the previous hash and a hash of the current data. Then, $H_t = h(W_t)$ is the hash value stored with the $(t+1)$th block. Thus, the hash chain is as shown in Fig. 1.

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

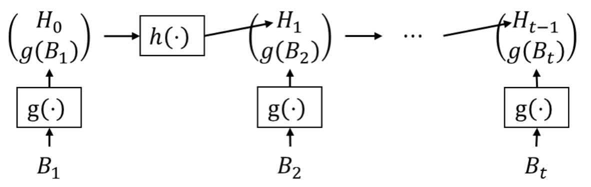

Fig. 1. Hash chain structure for the ledger. The chain is constructed by hashing a hash value of the data for easier recovery and consistency verification.

data block can be replaced by a Merkle tree structure with the results herein extending directly.

For all t, let $\mathbf{B}_t \sim \text{Unif}(\mathbb{F}_q)$ 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. Thus, the cost of storage per peer per transaction in conventional implementation is

$$\tilde{R}_s = \log_2 q + 2 \log_2 p \text{ bits.} \quad (1)$$

In practice, data blocks can be of varying sizes and the results described here follow *mutatis mutandis*.

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. This method is mathematically equivalent to most standard techniques used for data recovery on the blockchain.

B. Blockchain Security

Two aspects of security are particularly important in blockchain systems – *integrity* and *confidentiality*. Whereas an integrity property guarantees that the stored data stored 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. We study both aspects of security in this work.

Blockchain is preferred for business applications primarily because of the immutability guarantee on stored data. This is owing to the fact that data corruption requires corrupting a majority of the peers. Additionally, such corruptions can be detected unless chain consistency is also ensured. Thus, an adversary would also need to alter succeeding hash values, establishing integrity of stored data.

Some systems enforce additional constraints on hash values to enhance data integrity, e.g., difficulty targets in bitcoin. These make recomputation of a valid hash expensive, thereby adding to the difficulty of data corruption. Such methods however have recently faced criticism for being energy intensive in the ledger creation process. We design a method in this work that enhances data integrity without such extensive demands.

Conventional implementations of the blockchain involve each peer storing a copy of the transactions. Thus confidentiality is typically established through the use of private key encryption of the data by the client, where the key is shared only with a subset of peers with read authorization. However, for such implementations, a leak at a single authorized peer could lead to complete disambiguation.

C. Adversary Model

Here we consider two main attacks: non-adaptive, randomized denial of service (DoS) attacks, and system-aware, targeted data corruption attacks. Whereas the former prevents data recovery, resulting in loss of stored data, the latter results in potential alteration of stored data by an active adversary.

We study these two types of attacks separately with the focus on complete data loss and on adversarial corruption. In addition, let us presume that in any slot, a peer could fail (nodal failure or local data loss) independently with probability ρ. Let $\bar{\rho} = 1 - \rho$. We now elaborate on the adversaries.

First, we consider a non-adaptive adversary who is unaware of the system parameters and the distribution of data among peers. We refer to such adversaries as *DoS adversaries*. We assume that the adversary selects C peers at random, where C is chosen according to a distribution $P_{\text{dl}}(\cdot)$, to perform a DoS attack such that corrupted peers do not return the data corresponding to the requested data block. The choice of P_{dl} is made either by a budget-limited adversary aiming to maximize probability of data loss.

On the other hand, *active adversaries* aim to alter a value B_t to some B'_t. We define the semantic rules of a valid corruption. An active 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 that the adversary is computationally limited from performing exhaustive search on the message/hash space. For ease of understanding we presume that local data loss does not occur in the presence of active corruption. This can however be incorporated into the study separately through careful analysis. Additionally, as the adversary is adaptive, we also presume that it is aware of the parameters that define the data storage.

Let the power of the adversary to corrupt any peer in the network be characterized by the parameter $P_{\text{tc}} \in [0, 1]$, the probability that a peer can be successfully corrupted. We presume homogeneity across the network and that the corruptions are independent and identical across peers.

III. Coding Scheme

We now describe the coding scheme for distributed secure storage. We presume that all computation for the encoding and decoding is done privately by a black box. That is, peers, clients, and especially active adversaries are not made aware of the code. Specifics of practical implementation of such a black box scheme is beyond the scope of this paper.

A. Coding Data Block

We devise a coding mechanism that distributes the data stored on the blockchain in a secure manner. At any time t, for a transaction B_t, the client can choose the number $k_t \in [n]$ such that each copy of the transaction data is distributed among a set of k_t peers. For convenience, we assume that n is divisible by k_t. Thus at any time t, select a partition of the peers of size k_t uniformly at random and let each set of the

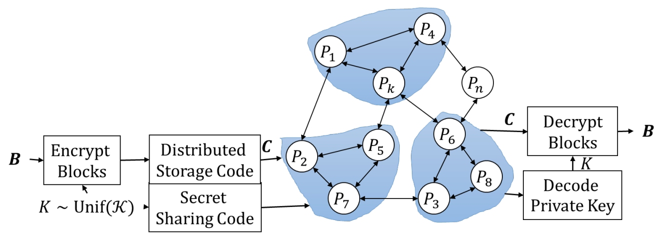

Fig. 2. Encoding and decoding for a given zone allocation. Shaded regions represent individual zones.

Algorithm 1 Coding scheme for data block

```
for $z = 1$ to $\frac{n}{k_t}$ do
    Generate private key $K_t^{(z)} \sim \text{Unif}(\mathcal{K}_t)$
    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 $(k_t, k_t)$ secret sharing on $K_t^{(z)}$ and distribute shares $(K_1^{(z)}, \dots, K_{k_t}^{(z)})$ among peers in the zone
end for
```

partition be referred to as a *data zone*. For each peer $i \in [n]$, let $p_t^{(i)} \in [\frac{n}{m}]$ be the zone index. The data B_t will be securely distributed in each zone.

The coding scheme essentially follows the same construction as in [15] except that the zone size varies with each block as shown in Fig. 2 and summarized in Alg. 1. Here, $\mathcal{K}_t$ is the space of key values as determined by the parameter k_t. Throughout 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.

For ease, assume that each peer stores a component of the code vector $\mathbf{C}_t$. The theory extends naturally to other MDS style distributed storage codes.

We now encode the hash values. Let us assume that there exists a known algorithm that deterministically chooses a partition $\mathcal{P}_t$ of the peers into sets of size m each at time t. We refer to the sets of this partition as *hash zones*. We assume that n is divisible by m. The hash values are then stored as in [15] using Shamir's secret sharing, i.e., at time t, each peer in a hash zone stores a secret share of the hash value H_{t-1} generated using Shamir's (m, m) secret sharing scheme. The same scheme is also used for the Merkle root $g(B_t)$.

The storage for the tth transaction per peer is thus

$$R_s^{(t)} = \frac{1}{k_t} \log_2 |\mathcal{C}| + 2 \log_2 |\mathcal{K}_t| + 4 \log_2 p \text{ bits}, \qquad (2)$$

where q and p are the data and hash field sizes, and $|\mathcal{C}|$ depends on the encryption. If the keys are much smaller than the blocks, we have considerable storage savings.

One example of a feasible encryption and the corresponding decryption algorithm, that is optimal in the size of the key space up to log factors, is described in [15, Alg. 3,4]. The resulting storage cost for such a scheme is

$$R_s^{(t)} = \frac{1}{k_t} \log_2 q + k_t (2 \log_2 k_t + 1) + 4 \log_2 p \text{ bits}. \qquad (3)$$

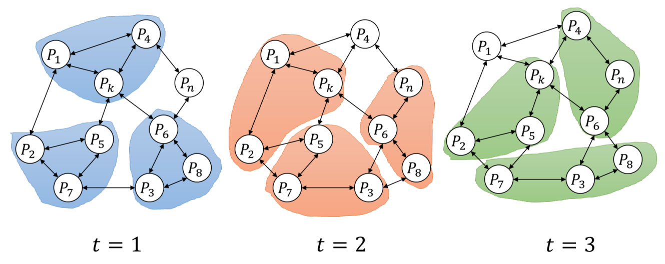

Fig. 3. Dynamic zone allocation: vary peer grouping over time.

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 $\{\{B_t^{(p_t^{(i)})} : i \in \mathcal{N}\}\}$
```

It is evident that the storage cost initially decreases with k_t, minimizing for a value of the order of $O(\sqrt{\log_2 q})$.

Additionally, we note that using the dynamic zone allocation strategy from [15, Alg. 5] for the partitions used for storing the hash values implies that the integrity guarantees hold here as well. That is, an active adversary needs to corrupt at least $n/2$ peers and over time almost all peers to perform a consistent, targeted corruption of a data block. Further, we note that the adversary would need to corrupt at least $2m$ new peers with each new block in the chain. This is depicted in Fig. 3 and the detailed proofs of the claims are in [15].

B. Recovery Scheme

For recovering a block, we make use of the shares stored at all peers and return the most consistent version. 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 to look into the hash chain to return the majority consistent version.

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 the hash chain for a maximum depth of d_t for consistency 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. If a hash is not recoverable owing to data loss or DoS, we skip the inconsistency check for corresponding peers in that zone in that slot. Finally, the majority of consistent data is returned.

The recovery process involves obtaining the data fragments stored at each peer and the communication cost involved therein typically supersedes any other computational costs. Let C_r be the cost of communicating one unit of data by all peers in the network.

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 $\frac{1}{k_t}\log_2 q + k_t(2\log_2 k_t + 1)$ 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 tth data block is

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

As is evident, the recovery cost decreases initially with k_t and subsequently increases with a minimum achieved when $k_t = O(\sqrt{\log_2 q})$. For most practical applications $q \gg n$ and thus, choosing a large value of k_t reduces the recovery cost. On the other hand, the larger the depth d_t, the higher the recovery cost as it grows linearly.

IV. Corruption Costs

We now characterize the probability of data loss and targeted corruption by budget-limited DoS and active adversaries.

A. Data Loss

A data block is lost when some peers undergo a DoS attack and a sufficiently large number incur random data loss. Consider an adversary that wishes to prevent the recovery of a block $\mathbf{B}$ distributed according to the parameter k. Let $r = n/k$ 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. 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 k objects of each of the r types.

The probability of data loss given the adversary attacks C peers is

$$\begin{aligned}\mathbb{P}[\text{Data Loss}|C] &= \mathbb{P}[X_i + Y_i > 0,\ \text{for all } i \in [r]|C] \\ &= \mathbb{P}[Y_i > 0,\ \text{for all } i \in [r] \text{ s.t. } X_i = 0|C] \\ &= \mathbb{E}\left[(1-\bar{\rho}^k)^{(r-\sum_{i=1}^r \mathbf{1}\{X_i>0\})}|C\right], \quad (5)\end{aligned}$$

where (5) follows from the independence of nodal failure and the expectation is taken over the multivariate hypergeometric distribution described above. Here $\mathbf{1}\{\cdot\}$ is the indicator function. The probability of data loss is then obtained by averaging (5) over C.

Then, for a DoS adversary limited by a budget B_{dl} of the expected number of peers it can corrupt, P_{dl} can be determined by solving the following linear program

$$P_{\text{dl}} \in \arg\max_{p} \sum_{c=0}^{n} p(c)\mathbb{E}\left[(1-\bar{\rho}^{k})^{(r-\sum_{i=1}^{r}\mathbf{1}\{X_i>0\})}|C=c\right] \tag{6}$$

$$\text{s.t.} \sum_{c=0}^{n} cp(c) \le B_{\text{dl}}, \ \sum_{c=0}^{n} p(c) = 1, \text{ and } p(c) \ge 0, \text{ for all } c.$$

Computing the conditional expectation in (6) requires knowledge of the probability mass function (pmf) of the number of zones with non-zero corruption, given by

$$\mathbb{P}\left[\sum_{i=1}^{r}\mathbf{1}\{X_i>0\}=\tilde{r}\right] = \binom{r}{\tilde{r}}\mathbb{P}\left[\sum_{i=1}^{\tilde{r}} X_i = c\right] \tag{7}$$

$$= \binom{r}{\tilde{r}}\binom{\tilde{r}k}{c}\bigg/\binom{n}{c}, \tag{8}$$

where (7) follows from the symmetry in the zones. Then, the random variable $\sum_{i=1}^{\tilde{r}} X_i$ follows the hypergeometric distribution with parameters n, c, and $\tilde{r}k$ representing size of population, number of draws, and number of successes respectively. This results in (8).

Solving the LP (6) gives us an idea of the budget-limited DoS adversary and so the data loss probability can be subsequently computed from (5). The optimal design choice to account for the worst case DoS adversary would then be to pick k such that it minimizes the worst case data loss probability, i.e.,

$$k^* = \arg\min_{k}\max_{P_{\text{dl}}} \mathbb{P}\left[\text{Data Loss}\right].$$

However, the design choice is more nuanced and application-specific as it has to also account for other costs.

B. Targeted Corruption

We now consider the security from active adversaries that perform targeted corruption. In particular, let us presume that the adversary aims to corrupt a block $\mathbf{B}$ to $\mathbf{B}'$ when each copy of the data is distributed across k peers, and the recovery algorithm searches a depth of d along the chain. Again, let $r = n/k$.

From the integrity guarantees established, we know that the minimum number of peers that the active adversary needs to successfully corrupt to perform the targeted corruption is $\frac{n}{2} + 2dm$. However, the corruption is feasible only if the adversary can find a set of $r/2$ zones to first corrupt the block data in half the zones, and at least the subsequent $2dm$ peers, as determined by the dynamic zone allocation scheme to alter the hash values along the chain. For ease, we presume the worst case, wherein the adversary has to corrupt exactly $2dm$ following peers.

Then, the probability of successful targeted corruption of such a system is

$$\mathbb{P}\left[\text{Targeted Corruption}\right] = \binom{r}{r/2} P_{\text{tc}}^{\left(\frac{n}{2}+2dm\right)}. \tag{9}$$

Note that both design parameters k and d influence the integrity of stored data. Naturally, a larger d implies a lower feasibility of targeted corruption. This however leads to a higher recovery cost as indicated by (4).

Also, the larger the value of k, the smaller the probability of targeted corruption. However, large k values also imply a higher probability of data loss and thus, there exists a tradeoff in the choice.

C. Data Confidentiality

With relation to confidentiality of stored data, we consider two scenarios. First, we consider the amount of information an eavesdropper, external to the system, receives from the knowledge of the complete cipher code stored in a zone, about the data block. Secondly, we study the minimum fraction of peers that need to collude to recover stored data.

First, assume that an external eavesdropper receives the complete cipher text, $\mathbf{C}$, but not the encryption key K corresponding to a zone. Since the block data are chosen uniformly at random,

$$H(\mathbf{B}|\mathbf{C}) \le H(K, \mathbf{B}|\mathbf{C}) = H(K). \tag{10}$$

The entropy of the transaction block is actually $H(\mathbf{B}) = \log_2 q > H(K)$. Thus at any time t, the lower bound on the information revealed from the knowledge of the encrypted storage is

$$I(\mathbf{B}_t; \mathbf{C}_t) \ge \log_2\left(\frac{q}{k_t(2\log_2 k_t + 1)}\right). \tag{11}$$

Note that this is the minimum amount of information that the eavesdropper is aware of regarding the stored data. Ideally, a client might require this information leak, and at least the lower bound to be low. It is evident that this is facilitated by a large value of k_t.

Next, consider peers who collude to read stored data. From Shamir's secret sharing of the encryption key, the data can be recovered with probability one, only from the collusion of at least k peers. That is, the minimum fraction of peers that need to collude to read stored data is

$$f_{il} = \frac{k_t}{n}. \tag{12}$$

This again emphasizes the need for a large k_t.

V. Blockchain-based Cloud Storage

As mentioned earlier, the individual client choices of design parameters are influenced by the variety of tradeoffs established here. Blockchain-based storage systems are of interest owing to the immutability guarantees on stored data. Using the tradeoffs established here, we describe a scheme selection mechanism by which the client can opt for a service that best serves the data being stored. Additionally it is important to note that the clients can inherently value different data blocks differently by appropriately varying the design choices.

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 that the cost of storing one unit of data at all peers per unit time is one. The communication cost for data recovery, C_r is priced in relation to this.

Let the frequency of data retrieval be ν, and let the parameters be k, d. Then, the storage cost per unit time is

$$\begin{aligned}\text{Service Cost} &= R_s + \nu R_r \\ &= \left(\tfrac{1}{k}\log_2 q + k(2\log_2 k + 1)\right)(1+\nu C_r) \\ &\quad + 2\log_2 p(1 + d\nu C_r), \qquad (13)\end{aligned}$$

which is obtained from (3) and (4).

Naturally, given the set of parameters, the probability of data loss, targeted corruption, and the fraction of colluding peers for information leak are determined by the maximum value of the LP (6), (9), and (12) respectively.

Thus, given a particular data type, the client can choose the design parameters by solving the following integer program

$$(k^*, d^*) \in \arg\min_{k,d} R_s + \nu R_r \qquad (14)$$

such that

$$\mathbb{P}\left[\text{Data Loss}\right] \le \delta_{\text{dl}}, \qquad (15)$$

$$\mathbb{P}\left[\text{Targeted Corruption}\right] \le \delta_{tc}, \text{ and} \qquad (16)$$

$$f_{il} \ge \delta_{il}. \qquad (17)$$

Note that (14) is a non-linear integer program and presumes knowledge of the parameters that define the adversary strength.

B. Data Insurance

Distributed storage on blockchain systems provide us with an interesting opportunity to offer data insurance [18] for saved blocks of data owing to the security guarantees. Here, we briefly describe parameter selection (storage 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 service cost per unit time. The time T for data loss or successful corruption is distributed geometrically with the parameter θ. Then the expected time for payout of the insured data upon losing it is $\frac{1}{\theta}$. Thus, the service provider can select the storage parameters by solving the following problem

$$(k^*, d^*) \in \arg\min_{k,d} C, \quad \text{such that } C \ge (1+\mu)Vp. \qquad (18)$$

Again this is a non-linear integer program that is to be solved to obtain the desired profit margin on insured data blocks.

VI. Conclusion

In this work, we used a novel combination of secret key sharing, private key encryption, and distributed storage to design a code to distribute transaction data securely among peers in a blockchain. Decoupling the code for data and hash values, and using dynamic zone allocation, we enhanced the integrity of data storage on a blockchain. We studied the tradeoffs in storage and recovery cost, and the probabilities of data loss and targeted corruption in the presence of DoS and active adversaries. We formulate a cloud storage system and formulate the parameter selection problem as an integer program subject to conditions on the probability of data loss and corruption. Further, we described the parameter selection problem for insuring data stored on the blockchain.

References

[1] 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.

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

[3] D. Tapscott and A. Tapscott, *Blockchain Revolution*. New York: Penguin, 2016.

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

[5] 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.

[6] Blockchain info. [Online]. Available: https://blockchain.info/home

[7] 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.

[8] 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.

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

[10] 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.

[11] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symp. Rel. Distrib. Sys. (SRDS'05)*, Oct. 2005, pp. 191–201.

[12] 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.

[13] 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.

[14] 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.

[15] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for scaling blockchains," arXiv:1711.07617v2 [cs.IT], Jan. 2018.

[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] 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. 2–2.