Feasible Region of Secure and Distributed Data Storage in Adversarial Networks

Jian Ren, *Senior Member, IEEE*, Jian Li, Tongtong Li, *Senior Member, IEEE*, and Matt W. Mutka, *Fellow, IEEE*

***Abstract*—Large volumes of data are being generated daily from IoT networks, healthcare, and many other applications, which makes secure, reliable, and cost-effective data storage a critical infrastructure of the computing system. Existing data storage largely depends on centralized clouds, which is not only costly but also vulnerable to single points of failure and other types of security attacks. Moreover, cloud providers will have full access to user data and revision history beyond user control. To provide data security, data encryption has to be used, which requires extensive computing power and cumbersome key management. Distributed storage system (DSS) is being widely viewed as a natural solution to future online data storage due to improved access time and lower storage cost. However, the existing DSS also has the limitations of low storage efficiency and weak data security. In this article, we investigate multi-layer code-based distributed data storage systems that can achieve inherit content confidentiality and optimal storage efficiency. Our comprehensive performance analysis shows that the optimal code can improve the feasible region in reliable data storage by 50% under various adversarial attack scenarios.**

***Index Terms*—Adversarial networks, distributed data storage, feasible storage region, optimal.**

I. Introduction

IN JUST a few years, the Internet of Things (IoT) has evolved into almost every aspect of our daily life. As a result, a gigantic amount of data is being generated every day, which makes data storage a critical infrastructure of IoT networks. The cloud storage market size was valued at $46.12 billion in 2019, and is projected to reach $222.25 billion by 2027, growing at a CAGR of 21.9% from 2020 to 2027 [1]. Currently, the majority of various data is stored in just a few large central cloud data providers such as Amazon AWS [2]. These providers can get full access to user data and dictate the pricing of their services. Users have to trust the central cloud providers for their data availability and security. To limit the cloud providers from accessing their data, data encryption is the only feasible option, which is not only very costly but also vulnerable to key loss.

To address these issues, distributed storage system (DSS), such as IPFS/FileCoin [3], [4], has been introduced. As the name suggests, distributed storage works by splitting the data to be stored into multiple blocks and distributing the individual blocks across a decentralized network of storage peer nodes. To ensure data *availability*, the IPFS storage systems generally stores multiple copies of each block, which makes distributed storage much more reliable than the centralized storage. FileCoin makes IPFS storage a service marketplace by motivating anybody with available storage to join the peer-to-peer (P2P) network and become a FileCoin storage provider, which will make storage cheaper over time through bidding between uploaders and hosts and marketplace incentivizing. In blockchain [5] terms these peer nodes are the "*miners*" of FileCoin. The importance of this architecture is that significant storage space can be allocated without requiring any additional hardware, which can greatly reduce the storage cost. As a result, the DSS is much less expensive than the traditional data center-based cloud storage solutions.

IPFS can be seen as a compelling foundation for blockchain technologies. In IPFS, files are stored using a content-addressable storage (CAS) system: each file/block can be retrieved by a human-readable address created by the Interplanetary name system (IPNS) based on *what* it is, rather than *where* it is. CAS has three major advantages. First, it can be used to circumvent address-based censorship. In fact, IPFS has been used to create an uncensorable Turkish version of Wikipedia [6] and also sidestep Spain's legal block by the Catalan government during the recent crisis in Catalonia [7]. Second, the CAS system is much more reliable than the address-based data storage system. Currently, information on the Internet is addressed by universal resource locators (URLs), which could be the address that held the correct content at some time in the past. The addresses created this way is very unreliable. As an example, a recent Harvard-led study found that 50% of all hyperlinks cited in U.S. Supreme Court opinions are no longer working [8]. Third, CAS can make the Web not only safer against potential security attacks, but also more robust to content availability.

Unfortunately, the DSS schemes adopted by IPFS/FileCoin have some major limitations. First, the storage efficiency of IPFS is relatively low in that the exact copy of each block has to be collected in order to fully recover the original file. To ensure data availability, each block has to be stored multiple

Manuscript received May 31, 2021; revised September 11, 2021; accepted October 2, 2021. Date of publication October 11, 2021; date of current version May 23, 2022. This work was supported in part by the National Science Foundation under Grant CCF-1919154 and Grant ECCS-1923409. *(Corresponding author: Jian Ren.)*

Jian Ren, Tongtong Li, and Matt W. Mutka are with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: renjian@msu.edu; tongli@msu.edu; mutka@msu.edu).

Jian Li is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: lijian@bjtu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3119031

times. Second, IPFS splits large files into blocks through file fragmentation. The individual blocks carry information directly related to the original file. If not protected, the blocks could leak significant information about the whole file.

To address the aforementioned issues of IPFS while also maintaining the major advantages, we propose a new multi-layer code-based distributed storage system. The major contribution of this article can be summarized as follows.

1) We analyze the limitations of the existing DSS in storage efficiency and security.
2) We introduce multi-layer code-based reliable data storage schemes and compare our approach with the existing DSS schemes in storage efficiency and security.
3) We also analyze the multi-layer code storage efficiency under various security attack models. In particular, we develope the secure and reliable storage region of the multi-layer code.

The remainder of this article is organized as follows. In Section II, various issues related to distributed data storage are discussed. A comprehensive discussion of the multi-layer code is presented in Section III. In Section IV, we present the optimal storage scheme in adversarial networks. We provide simulation results in Section V and the conclusion in Section VI.

II. Discussion of Distributed Data Storage

A. Security Issues of Centralized Storage

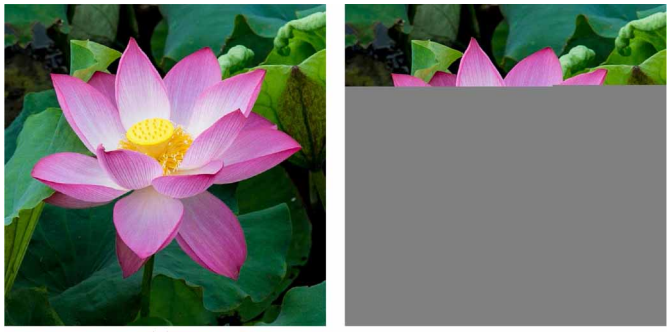
Centralized cloud provides data storage as a service. Users do not need to build a hardware infrastructure and invest a formidable amount of money to store their data and ensure data fault-tolerance or redundancy. However, building a successful cloud data storage business requires a global network of data-centers and robust user interfaces that satisfy many user demands. As a result, only a few massive corporations owning nearly all of the global cloud data storage market.

Since its inception cloud data storage has evolved to be functional but leaves many security concerns and performance issues unaddressed. First, centralized databases are highly dependent on network connectivity and the Internet speed. For data stored far away from the end-user, the database access time can be very long especially when the Internet speed is slow. In fact, the centralized databases could also become a bottleneck as a result of high traffic. Second, users must rely on these large corporations to ensure their data availability and data security. However, due to economic reasons, storage becomes a commodity or a utility for the data providers. The service providers are incentivized to lock in their customers and extract a premium. The nature of these issues could potentially conflict with users' security needs.

B. Limitations of IPFS

The decentralized storage for the IPFS system has major security and efficiency issues. We use the following example to illustrate the security limitation.

Example 1: To store a picture of a lotus flower in IPFS. The file size is 1 034 141 bytes with hash value `PQmaNMtwsCJKVmUEe9eaSesKAKhigDzrHf3J4fbAE3VFsAh`.



(a) (b)

Fig. 1. File storage in IPFS. (a) Original picture. (b) First segment of the picture.

IPFS splits the file into blocks through file fragmentation. The default IPFS block size is $2^{18} + 14 = 262158$ bytes. In this case, the hash values of the 4 blocks are:

```
QmYmpjVoCXD5hiVz8PTDXvnqpBPNiiBQeDP3XfXh1iri9b 262158
QmZYgQBkctTKNRWQoUEzJ7NxwJjoGvYim9Zj9D61jKrHgU 262158
QmXbTiNaNL5AtsG8iRtBQgzUwSGfX4JkuoWAnhyBLfDXYV 262158
QmQuErMLNnXvt7SUwXkgRzQun6PYJGDGEiv2ag342L1Tdq 247723
```

These blocks are stored based on their unique fingerprints and can be accessed through Web addresses (gateways) in the following format:

```
https://ipfs.io/ipfs/<block_cryptographic_hash>
```

This design enables IPFS to remove duplication across the network. For example, the first block would be stored and accessible from the following link:

```
https://ipfs.io/ipfs/QmYmpjVoCXD5hiVz8PTDXvnqpBPNiiBQeDP3XfXh1iri9b
```

Fig. 1 shows the original lotus flower and the first segment of the flower stored in IPFS. From this figure, we can see that the first segment of the lotus flower clearly reveals a significant amount of information about the original flower. This example makes it clear that the current IPFS file storage system cannot ensure content confidentiality.

In addition to security limitations, the data storage efficiency of IPFS is also quite low. As described before, the data blocks in IPFS are generated through file fragmentation-based approaches. While the individual blocks are stored in a decentralized way, it is fundamentally different from the general DSS in that the blocks of DSS are generated through algebraic encoding. For IPFS-based decentralized storage, to recover the original file, at least one copy of each block must be collected. To increase data availability, IPFS has to store multiple copies of each block. While for encoding-based (n, k) DSS, the file to be stored is first *algebraically* split into n blocks so that the original file can be recovered from *any* $k \leq n$ blocks. However, the original file remains *information-theoretically secure* for anyone who can access even *up to any* $k - 1$ *blocks and with unlimited computing power*. Therefore, no encryption or key management is required to ensure data confidentiality of the file stored. To increase data availability, we only need to increase the number of n, which makes it much more efficient than the IPFS-based decentralized data storage. Fig. 2 shows the dramatic data availability

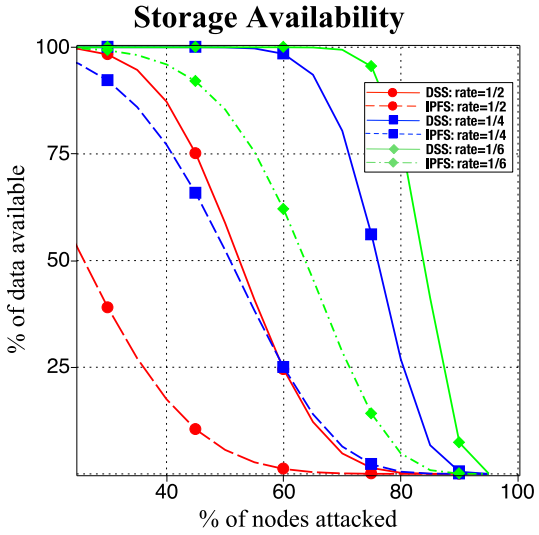


Fig. 2. Data availability comparison between DSS and IPFS.

differences for three different storage efficiencies of these two approaches.

C. Error and Erasure Correcting Codes

For integers n, k, δ, and a symbol set Σ, an (n, k, δ)-code over Σ is a subset $\mathcal{C} \subset \Sigma^n$ of n-letter words over the alphabet Σ with $|\mathcal{C}| = |\Sigma|^k$ and the property that any two strings in $\mathcal{C}$ differ in at least δ places. Given a code $\mathcal{C}$, the largest δ for which $\mathcal{C}$ is an (n, k, δ)-code is referred to as the Hamming distance of the code $\mathcal{C}$.

Reed–Solomon (RS) code is a class of maximum distance separable (MDS) code with a set of parameters $(n, k, \delta = n - k + 1)$ that operates on a block of n data symbols over a finite-field $\mathbb{F}_q$ (i.e., $\Sigma = \mathbb{F}_q$), where q is a prime number or power of a prime number. It is formed by adding $n-k$ check symbols to k data symbols. An RS code can locate and correct up to and including $\lfloor(\delta-1)/2\rfloor$ erroneous symbols at unknown locations. As an erasure code, it can correct twice as many erasures as errors at locations that are known and any combination of errors and erasures as long as the relation $2t + s \leq \delta - 1$ is satisfied, where t is the number of errors and s is the number of erasures in the block.

D. Distributed Storage System

In DSSs, the file to be stored is split into blocks and distributed across multiple network peer nodes. Unlike traditional hardware-based data storage, DSS outsources the data storage to a number of P2P storage servers that act as a single storage unit while the data is distributed amid the specific number of servers. DSS is being viewed as a more advanced form of the concept of software-defined storage (SDS).

DSS allows storage to act as a software application that is more like a database or even part of the operating system. While greatly simplifying the information technology stack, it also works with a single block centralized cloud storage data center. DSS gives the freedom of scaling by adding more peer nodes through economic incentives to promote the number of participating peers, foster a distributed storage market, and make storage more cost-efficient. Whenever more nodes are added, the increment in the total components increases the scalability, reliability, and speed of the entire DSS.

Moreover, compared to the cloud-based centralized storage system, the DSS can achieve great security inherit from the DSS design since the data being stored are decomposed into blocks and then spread over many different peer nodes.

In DSS, data replication and erasure coding are two widely employed data redundancy techniques. While replication has the advantage of simplicity and low access overhead, it imposes higher repair traffic and storage overheads. Moreover, the data stored in this system is essentially similar to the IPFS system described in Section II-B. The data components directly contain information of the original files. Therefore, to conceal the contents of the files, data encryption has to be applied to all the data components, which will impose not only the computational cost but also the overhead for secure key management. Conversely, to reduce storage overhead, erasure code avoids data replication. Even though it requires some very limited computational costs due to coding/decoding operations, compared to the gain in data availability and inherit data content security, it can be fully justified.

The best known DSS schemes are designed based on the conventional (n, k) RS error-correction codes (such as OceanStore [9] and Total Recall [10]). When a node fails, a replacement node can be regenerated by connecting to any k benign nodes, which will first recover the whole file, and then regenerate the failure node. This approach is a waste of bandwidth because the whole file has to be downloaded to recover even a small fraction of it.

E. Regenerating Code-Based DSS

To improve bandwidth efficiency in repairing node failure, Dimakis *et al.* [11] introduced the concept of $\{n, k, d, \alpha, \beta, B\}$ linear regenerating code over the finite field based on network coding. A file of size B is stored in n storage nodes, each of which stores α symbols. A data collector (DC) can reconstruct the whole original file stored in the network by downloading α symbols from each of k randomly selected storage nodes. In the context of regenerating code, a replacement of a failed node can be regenerated by downloading $\beta \leq \alpha$ symbols from each of $d \geq k$ randomly selected storage nodes. Therefore, the total bandwidth required to regenerate a failed node is $\gamma = d\beta$, which could be far less than the whole file B.

For n storage peer nodes, the size of the file to be stored in the distributed storage network is B (symbols). Based on a cut-set bound on network-coding, the following theoretical bound was derived in [11]:

$$B \leq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\}. \tag{1}$$

In (1), there is a tradeoff between the choices of the parameters α and β, which corresponds to the minimum storage regeneration (MSR), where the storage parameter α is minimized as follows:

$$(\alpha_{\text{msr}}, \gamma_{\text{msr}}) = \left(\frac{B}{k}, \frac{Bd}{k(d-k+1)}\right) \tag{2}$$

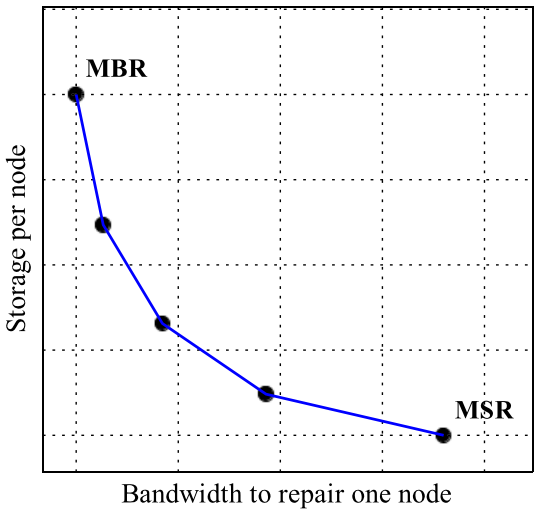


Fig. 3. Tradeoff between MSR and MBR.

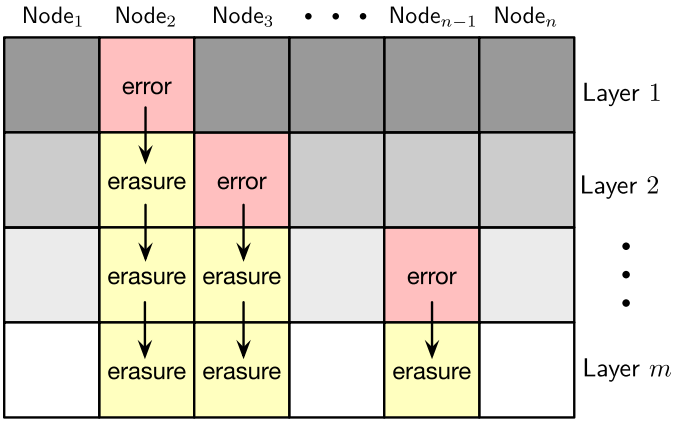


Fig. 4. Multi-layer RS code design.

and the minimum bandwidth regeneration (MBR), where the regeneration bandwidth γ is minimized as follows:

$$(\alpha_{\text{mbr}}, \gamma_{\text{mbr}}) = \left(\frac{2Bd}{2kd - k^2 + k}, \frac{2Bd}{2kd - k^2 + k}\right). \quad (3)$$

Fig. 3 shows a tradeoff between the MSR and MBR.

Existing DSS was largely constructed based on the MDS code [11], such as RS code. The error correction and node regenerating capability is *limited to* the theoretical error correction bound of the MDS codes.

Compared to MDS code-based schemes, *regenerating code* can reach the *optimal* theoretical tradeoff between the minimum *storage* regenerating and the minimum *bandwidth* regenerating. Moreover, in regenerating code, it is no longer needed to decode the original file in order to regenerate a failed peer node.

F. Adversarial Model

In P2P networks, it is possible that the storage nodes may be attacked by adversarial security attacks. The affected nodes may provide incorrect response to disrupt data reconstruction and regenerating, or becomes totally unavailable.

We assume that the goals of the attackers are twofold: 1) disrupt as many distributed storage nodes as possible, and 2) manipulate the content of as many storage nodes as possible. We will refer to these nodes as *spurious nodes*.

In this system, the defender and the attacker compete to maximize their goals. We model the interaction between the defender and the attacker as a security game. In this game, the network manager—henceforth, the defender—determines the overall tradeoff between the storage efficiency and the storage robustness under malicious attacks.

We assume that the attacker has knowledge of the data allocation scheme. It observes the randomized allocation with an intention to disrupt or manipulate as many peer nodes as possible for any given budget to thwart data reconstruction and availability. Based on this information, the defender first generates n data components from B symbols to be stored in the P2P network using the regenerating code-based approach. It then randomly allocates the n data components across the entire P2P network so that the original B symbols can be recovered when any k untackled components are collected. Meanwhile, any spurious node can be repaired by collecting β symbols from d storage nodes.

To optimize data storage reliability, the defender will try to minimize the number of spurious nodes being attacked and controlled by the attackers. At the same time, the defender can adjust the design tradeoff parameters to ensure optimal data storage reliability and storage efficiency. In other words, the defender's strategy is to maximize the number of intact nodes to be at least d, where d is the node regenerating parameter.

III. Multi-Layer Code Design

In this section, we introduce our proposed multi-layer regenerating code for DSS and derive the *optimal* code construction. Since MSR and MBR are similar in description, we will only present the MSR code scenario.

A. Multi-Layer Codes and DSS

Multi-layer codes, or m-layer codes, split the data to be stored into m layers and n blocks and then distribute the blocks to $n_0 \leq n$ different storage peer nodes, as shown in Fig. 4. The m layers correspond to m RS codes. The code rate of the m-layer code increases from layer 1 to layer m. On the other hand, the error-correction capability decreases from layer 1 to layer m. Because of this structure, the error correction and node repairing of the m-layer code will proceed from layer 1 and then move toward layer m, layer by layer. Since each node stores data corresponds to all m layers, the nodes with errors detected in one layer, all the subsequent insiders layers stored in that node also become unreliable. Therefore, for the sake of computational efficiency, they will be treated as erasures, as illustrated in Fig. 4.

Let the parameter for the ith layer MSR code be viewed as an $(n-1, d_i, n-d_i)$ code, $i = 1, 2, \ldots, m$, where each component contains β symbols as described in the regenerating code setting, and $d_i \leq d_j$, $\forall 1 \leq i \leq j \leq m$. Then the first layer

code can detect and repair t_1 failure nodes

$$t_1 = \lfloor (n - d_1 - 1)/2 \rfloor = (n - d_1 - \varepsilon_1 - 1)/2 \tag{4}$$

where $\varepsilon_1 = 0$ if $n - d_1 - 1$ is even and $\varepsilon_1 = 1$ otherwise.

By treating the symbols from the t_1 nodes where errors have been found as erasures, then based on discussion in Section II-C, the second layer code can detect and repair t_2 failure nodes

$$\begin{aligned} t_2 &= \lfloor (n - d_2 - 1 - t_1)/2 \rfloor + t_1 \\ &= (n - d_2 - 1 - t_1 - \varepsilon_2)/2 + t_1 \\ &= (2(n - d_2 - \varepsilon_2 - 1) + (n - d_1 - \varepsilon_1 - 1))/4 \end{aligned} \tag{5}$$

assuming that $n - d_2 - 1 \geq t_1$, where $\varepsilon_2 = 0$ if $n - d_2 - 1 - t_1$ is even and $\varepsilon_2 = 1$ otherwise.

This process can be continued for $2 \leq i \leq m$, by treating the symbols from the t_{i-1} nodes where errors have been found in the layers $1, 2, \ldots, (i-1)$th layer as erasures, the ith layer code can detect and repair t_i failure nodes

$$\begin{aligned} t_i &= \lfloor (n - d_i - 1 - t_{i-1})/2 \rfloor + t_{i-1} \\ &= (n - d_i - 1 - t_{i-1} - \varepsilon_i)/2 + t_{i-1} \\ &= \left(\sum_{j=1}^{i} 2^{j-1} \big(n - d_j - \varepsilon_j - 1\big) \right) / 2^i \end{aligned} \tag{6}$$

assuming that $n - d_i - 1 \geq t_{i-1}$, where $\varepsilon_i = 0$ if $n - d_i - 1 - t_{i-1}$ is even and $\varepsilon_i = 1$ otherwise.

B. Multi-Layer Code Construction

The multi-layer DSS was first explored from Hermitian code [12]–[14]. A Hermitian curve $\mathcal{H}(q)\colon y^q + y = x^{q+1}$ is defined over $\mathbb{F}_{q^2}$. There are q^3 points on this curve, denoted as: $P_{i,j} = (\phi^{i-1}, \phi^{(i-1)(q+1)+1} + \theta_j)$ $(i = 0, 1, \ldots, q^2 - 1, j = 0, \ldots, q-1)$, where ϕ is the primitive element over $\mathbb{F}_{q^2}$ with $\phi^{-1} = 0$ and θ_j is the solution to $y^q + y = 0$.

Let $\mathcal{G}_j = \{(y^j f_j)(P_{0,0}), \ldots, (y^j f_j)(P_{0,q-1}), \ldots, (y^j f_j)(P_{q^2-1,0}), \ldots, (y^j f_j)(P_{q^2-1,q-1})\}$, $j = 0, \ldots, q-1$, be an RS-code, where f_j is a polynomial satisfying

$$\deg f_j < \kappa(j) = \max\{t \mid tq + j(q-1) \leq m\} + 1.$$

According to [12], $\mathcal{G}_j$ can be decomposed as a concatenation of q RS-codes of length q^2. That is

$$\mathcal{H}_m = \mathcal{G}_0 \oplus \mathcal{G}_1 \oplus \cdots \oplus \mathcal{G}_{q-1}.$$

Alternatively, it can be viewed as a multiple layer RS code, with code parameters

$$\Big(q^2 - 1, \kappa(i), q^2 - \kappa(i)\Big),\ i = 0, 1, \ldots, q-1. \tag{7}$$

For a Hermitian code $\mathcal{H}_m$ over $\mathbb{F}_{q^2}$, a message matrix $M_{\dim(\mathcal{H}_m) \times A} = [M_1, \ldots, M_A]$ is encoded columnwise. The codeword matrix is $\mathcal{H}_m(M) = [\mathcal{H}_m(M_1), \mathcal{H}_m(M_2), \ldots, \mathcal{H}_m(M_A)]$, where $\mathcal{H}_m(M_i)$ has the following form ($\varrho \in L(mQ)$):

$$\big[\varrho(P_{0,0}), \ldots, \varrho(P_{0,q-1}), \ldots, \varrho(P_{q^2-1,0}), \ldots, \varrho(P_{q^2-1,q-1})\big]^T.$$

Algorithm 1 H-MSR Encoding Algorithm [14]

1: Encode the data matrices S, T defined above using a Hermitian code $\mathcal{H}_m$ over $GF(q^2)$ with parameters $\kappa(j)$ $(0 \leq j \leq q-1))$ and m $(m \geq q^2 - 1)$.
2: Calculate the $q^3 \times A$ codeword matrix $Y = \mathcal{H}_m(S) + \Gamma \mathcal{H}_m(T)$.
3: Divide Y into q^2 submatrices $Y_0, \cdots, Y_{q^2-1}$ of the size $q \times A$. Store the submatrix in (up to) q^2 storage nodes.

For Hermitian code-based MSR (H-MSR) construction, let $\alpha_0, \ldots, \alpha_{q-1}$ be a strictly decreasing integer sequence satisfying $0 < \alpha_i \leq k(i)$, $0 \leq i \leq q-1$, and $A = \mathsf{lcm}(\alpha_0, \ldots, \alpha_{q-1})$. Arrange $B = A \cdot \sum_{i=0}^{q-1} (\alpha_i + 1)$ symbols into two matrices S and T as follows:

$$S = \begin{bmatrix} S_0 \\ \vdots \\ S_{q-1} \end{bmatrix}, T = \begin{bmatrix} T_0 \\ \vdots \\ T_{q-1} \end{bmatrix}$$

where $S_i = [S_{\alpha_i,1} \quad S_{\alpha_i,2} \quad \cdots \quad S_{\alpha_i,A/\alpha_i}]$, $T_i = [T_{\alpha_i,1} \quad T_{\alpha_i,2} \quad \cdots \quad T_{\alpha_i,A/\alpha_i}]$, $S_{\alpha_i,j}, T_{\alpha_i,j}$ $(0 \leq i \leq q-1, 1 \leq j \leq A/\alpha_i)$ are symmetric matrices of size $\alpha_i \times \alpha_i$. Then by processing the data symbols using Algorithm 1, we can achieve the MSR point in the distributed storage [13], [14], where $\Gamma = \mathsf{diag}(\Lambda_0, \Lambda_1, \ldots, \Lambda_{q^2-1})$, and the q diagonal elements in Λ_i are all identical to $\lambda_i \in \mathbb{F}_{q^2}$. Since H-MSR code can be viewed as q layers of RS codes with parameters

$$\Big(q^2 - 1, d_i, q^2 - d_i\Big),\ i = 0, 1, \ldots, q-1, \alpha_l \leq \kappa(l) \tag{8}$$

where $d_i = 2\alpha_i, \alpha \leq \kappa(i)$, we can choose the sequence α_i to be strictly decreasing so that d_i is also strictly decreasing. For the q RS codes, the minimum distance of the $(q^2 - 1, d_{q-1}, q^2 - d_{q-1})$ code is the largest. It can correct up to τ_{q-1} errors

$$\tau_{q-1} = \Big\lfloor \Big(q^2 - d_{q-1} - 1\Big)/2 \Big\rfloor$$

where $\lfloor x \rfloor$ is the floor function of x.

Next, the code $(q^2 - 1, d_l, q_2 - d_l), l = q-2, \ldots, 0$ will be decoded sequentially, which can correct at least $\tau_l = \tau_{q-1}$ errors when $q^2 - d_0 - 1 \geq \tau_{q-1}$. Therefore, the total number of errors that the H-MSR can *detect* and *repair* is

$$\tau_{\mathsf{H-MSR}} = q\tau_{q-1} = q\Big\lfloor \Big(q^2 - d_{q-1} - 1\Big)/2 \Big\rfloor.$$

In comparison, for RS-MSR code with the same rate can repair

$$\tau_{\mathsf{RS-MSR}} = \left\lfloor \left(q^3 - q - \sum_{l=0}^{q-1} d_l \right) /2 \right\rfloor. \tag{9}$$

Therefore

$$\tau_{\mathsf{H-MSR}} - \tau_{\mathsf{RS-MSR}} \geq \frac{q(q-1)}{4} > 0.$$

Moreover, the proposed H-MSR code has complexity $\mathcal{O}(n^{5/3})$ for node regenerating and reconstruction, which is lower than the RS-MSR complexity $\mathcal{O}(n^2)$ in both scenarios.

C. Optimal Regenerating Code

Motivated by the Hermitian code structure, a natural research task is to find out how to select the layers that can optimize the overall performance [15].

Similar to message encoding in Algorithm 1, to encode a file with size B, we select $d = 2\alpha$ and divide the file into θ blocks of data with size $\widetilde{B}$, where $\theta = \lceil B/\widetilde{B} \rceil$. Then the θ blocks of data will be encoded into codeword matrices $\mathsf{F}_1, \ldots, \mathsf{F}_\theta$ and form the final $n \times \alpha\theta$ codeword matrix: $\mathcal{C} = [\mathsf{F}_1, \ldots, \mathsf{F}_\theta]$. Each row $\mathbf{c}_i = [\mathbf{f}_{1,i}, \ldots, \mathbf{f}_{\theta,i}]$, $1 \le i \le n$, of the codeword matrix $\mathcal{C}$ will be stored in storage node i, where $\mathbf{f}_{j,i}$ is the ith row of F_j, $1 \le j \le \theta$.

The design of optimal multi-layer code is equivalent to maximizing the number of failure nodes that can be regenerated, which is determined by t_m. The problem can be formulated as the following optimization problem:

$$\begin{aligned} \text{maximize: } & t_m, \text{ where } t_i = \left\lfloor \frac{n - d_i - t_{i-1}}{2} \right\rfloor + t_{i-1} \\ & i = 2, \ldots, m, t_0 = 0 \qquad (10a) \\ \text{constraints: } & \sum_{i=1}^{m} d_i = d. \\ & d_{i-1} \le d_i, \ 2 \le i \le m \\ & n - d_i - t_{i-1} - 1 \ge 0, \ i = 2, \ldots, m \\ & n + d_i - 2d_{i+1}, \ge 0, \ i = 1, \ldots, m-1. \qquad (10b) \end{aligned}$$

This is a linear optimization. By introducing slack variables ε_i, $i = 1, 2, \ldots, m$, we can easily find that the *optimal regenerating code* can be achieved when $d_i = \text{round}(d/m) = \tilde{d}$ as shown in [15]. In this case, as shown in (6), the maximum number of failure nodes that the m-layer optimal regenerating code (O-MSR) can repair is at most

$$\begin{aligned} t_m &= \left(\sum_{i=1}^{m} 2^{i-1} \left(n - \tilde{d} - 1 - \varepsilon_i \right) \right) / 2^m \\ &= \left(1 - \frac{1}{2^m} \right) \left(n - \tilde{d} - 1 \right) - \frac{1}{2^m} \sum_{i=1}^{m} 2^{i-1} \varepsilon_i. \qquad (11) \end{aligned}$$

By carefully selecting the d so that $n - \tilde{d} - 1$ is even, then we have $\varepsilon_i = 0$ for $i = 1, \ldots, m$, which can maximize the error correction capability for the given storage efficiency. In this case, from (11), we have

$$\begin{aligned} \lim_{m \to \infty} t_m &= \lim_{m \to \infty} \left(\sum_{i=1}^{m} 2^{i-1} \left(n - \tilde{d} - \varepsilon_i - 1 \right) \right) / 2^m \\ &= n - \tilde{d} - 1. \qquad (12) \end{aligned}$$

In this case, for the single layer RS code, we have

$$\begin{aligned} t_{\mathsf{RS-MSR}} &= \left\lfloor \left(n - \tilde{d} - 1 \right) / 2 \right\rfloor \\ &= \left(n - \tilde{d} - 1 \right) / 2. \qquad (13) \end{aligned}$$

In the worst case when $\varepsilon_i = 1$ for $i = 1, \ldots, m$, we have

$$\begin{aligned} \lim_{m \to \infty} t_m &= \lim_{m \to \infty} \left(\sum_{i=1}^{m} 2^{i-1} (n - \tilde{d} - \varepsilon_i - 1) \right) / 2^m \\ &= n - \tilde{d} - 2. \qquad (14) \end{aligned}$$

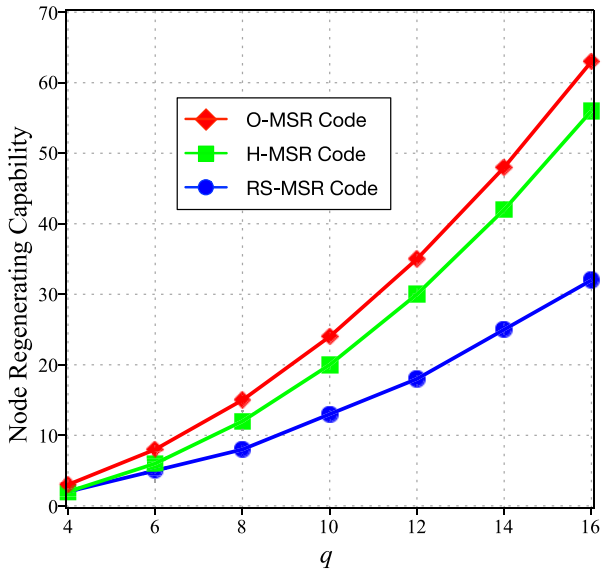


Fig. 5. Comparison of node regenerating capability of the m-layer code with the H-MSR and RS-MSR codes with code rate = 3/4 and $m = q$.

In this case, for the single layer RS code, the number of nodes that can be regenerated is

$$\begin{aligned} t_{\mathsf{RS-MSR}} &= \left\lfloor \left(n - \tilde{d} - 1 \right) / 2 \right\rfloor \\ &= \left(n - \tilde{d} - 2 \right) / 2. \qquad (15) \end{aligned}$$

Therefore, we have the following theorem.

Theorem 1: Let $t_{\mathsf{O-MSR}}^{(m)}$ be the number of nodes that can be regenerated by an m-layer optimal regenerating code and $t_{\mathsf{RS-MSR}}$ be the number of nodes that can be regenerated from a single layer RS-MSR code with comparable parameters, then

$$\lim_{m \to \infty} t_{\mathsf{O-MSR}}^{(m)} = 2 t_{\mathsf{RS-MSR}}. \qquad (16)$$

Theorem 1 proves that the overhead required to correct random node failure for the m-layer O-MSR code approaches one half of the RS-MSR code under comparable security parameters, which is the overhead required to correct erasures.

Fig. 5 compares the performance of our proposed multi-layer codes O-MSR and H-MSR codes with the RS-MSR code in which we select $m = q$ since H-MSR is only defined for this case.

IV. Optimal Storage in Adversarial Networks

In P2P networks, all the peer nodes can be vulnerable to security attacks. The data stored in these nodes may be either partially or completely damaged and become malicious.

For DSS, all the nodes are symmetric and with equal importance in data reconstruction and node regenerating. When a node is being attacked, the data stored in the node could be maliciously manipulated, controlled, or even completely damaged. These nodes will be referred to as *spurious nodes* and being treated in the same way without making any distinction.

In practice, the probability for each node to be attacked may vary. To simplify the analysis, we assume that the probability for each peer node to be attacked is p. Then the total number of

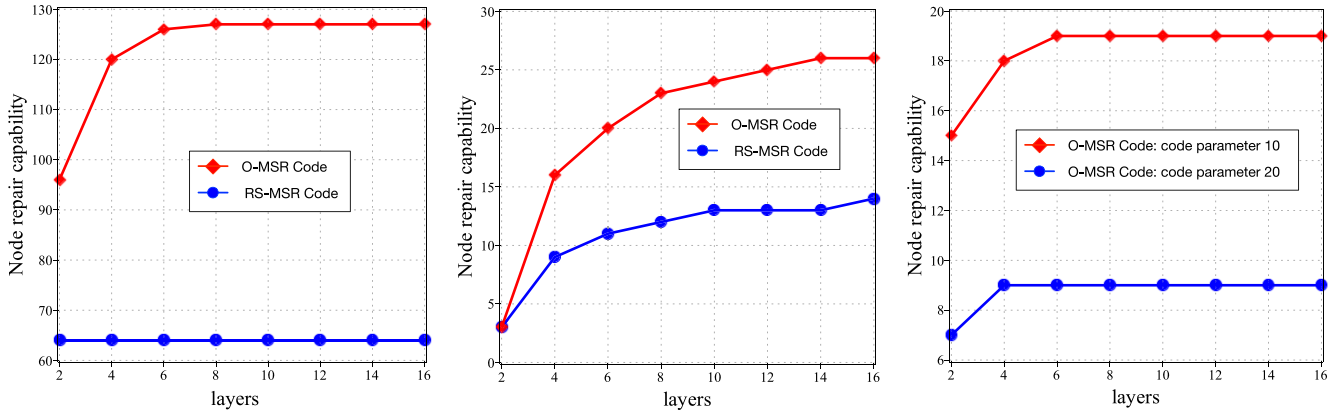


Fig. 6. Comparison between the maximum number of malicious nodes that can be regenerated between O-MSR and RS-MSR with comparable parameters. (a) $n = 256$, code rate $= 1/2$. (b) $n = 30, d = 50$, and $2 \le m \le 16$. (c) $n = 256, d = 10 \times m$, and $d = 20 \times m$.

spurious peer nodes is no more than $\mathsf{SpuriousNode} = \lceil np \rceil$, where $\lceil x \rceil$ denotes the ceiling of x. It is clear that there is a tradeoff between the attack probability p and the storage efficiency to ensure reliable data recovery. As p and the number of SpuriousNode increase, we have to decrease the storage efficiency in order to maintain the same data availability and storage reliability. Alternatively, we can increase the total number of storage nodes n. In other words, as the attacker becomes more powerful and is capable of manipulating more peer nodes, the data storage efficiency must be lowered down to ensure data storage reliability and availability. On the other hand, reliable data storage efficiency can be increased.

Unlike the scenarios described in the previous section, the goal of DSS is twofold. First, determine the optimal, reliable, and efficient data storage schemes for any given attack scenario. Second, design storage schemes that can achieve the optimal theoretical storage performance, which may vary dramatically in different networks. As a result, no single storage scheme can be optimal for all attack models.

The design of optimal and reliable data storage includes two major steps. First, evaluate the security game between the defender and the attacker, the available resources, and defending mechanisms. Ultimately, we hope to be able to determine the total number of spurious nodes SpuriousNode. Second, based on the aforementioned results, the defender determines the optimal parameters that can optimize the reliable data storage efficiency while ensuring data availability for the given attack scenario. In other words, our goal is to design the m-layer code that can maximize $\sum_{i=1}^{m} d_i = d$ under the constraints $t_m \ge$ the number of SpuriousNode, where

$$t_1 = \left\lfloor \frac{n - d_1 - 1}{2} \right\rfloor,$$
$$t_i = \left\lfloor \frac{n - d_i - t_{i-1}}{2} \right\rfloor + t_{i-1}, \quad i = 1, 2, \ldots, m.$$

The optimal storage under security attacks can be formulated as

$$\text{maximize:} \quad \sum_{i=1}^{m} d_i = d. \tag{17a}$$

$$\begin{aligned} \text{constraints:} \quad & n - d_i - t_{i-1} \ge 0 \\ & \text{where } t_i = \left\lfloor \frac{n - d_i - t_{i-1}}{2} \right\rfloor + t_{i-1} \\ & i = 2, \ldots, m, \text{ and } t_0 = 0 \\ & d_i \le d_{i+1}, \quad 1 \le i \le m - 1 \\ & n - d_i - t_{i-1} - 1 \ge 0, \ i = 2, \ldots, m \\ & t_m \ge \#\text{ of SpuriousNode}. \end{aligned} \tag{17b}$$

It is an integer linear optimization with $t_1 \le t_2 \le \cdots \le t_m$. Similar to (10), the optimal solution can be solved very efficiently. Based on the attack possibilities, we can then determine the corresponding optimal reliable data storage efficiency and the feasible data storage region for all attack probability p.

V. Simulation Results

In this section, we conduct comprehensive simulations to compare the storage capacity and feasible region between the O-MSR code with the existing RS-MSR code in adversarial networks.

In Fig. 6, we compare the maximum number of malicious nodes that can be regenerated by the single layer RS-MSR code and multi-layer O-MSR code with comparable parameters. In Fig. 6(a), we select $n = 256$, code rate $= 1/2$ for both RS-MSR and O-MSR with m varies from 2 to 16. In Fig. 6(b), we compare the number of nodes that can be repaired by m-layer O-MSR with $n = 30, d = 50$ and single layer $(n, \text{round}(d/m))$ RS MSR code for $2 \le m \le 16$. In Fig. 6(c), we compare the number of nodes that can be repaired by the O-MSR code with different code rates.

In Fig. 7, we compare the data storage capacity with respect to various adversarial attack probabilities for n from 0 to 262 144 symbols. For O-MSR, we select the number of layers $m = \lfloor \sqrt{n} \rfloor$. It should be made clear that for any fixed code length n, the performance for the O-MSR code increases with m. Therefore, the gap between the m-layer O-MSR code and the single layer RS-MSR code will further increase. This is shown in Fig. 8(a).

In Fig. 8(a), we show that the reliable data storage capacity for various adversarial attack possibilities, where the total storage

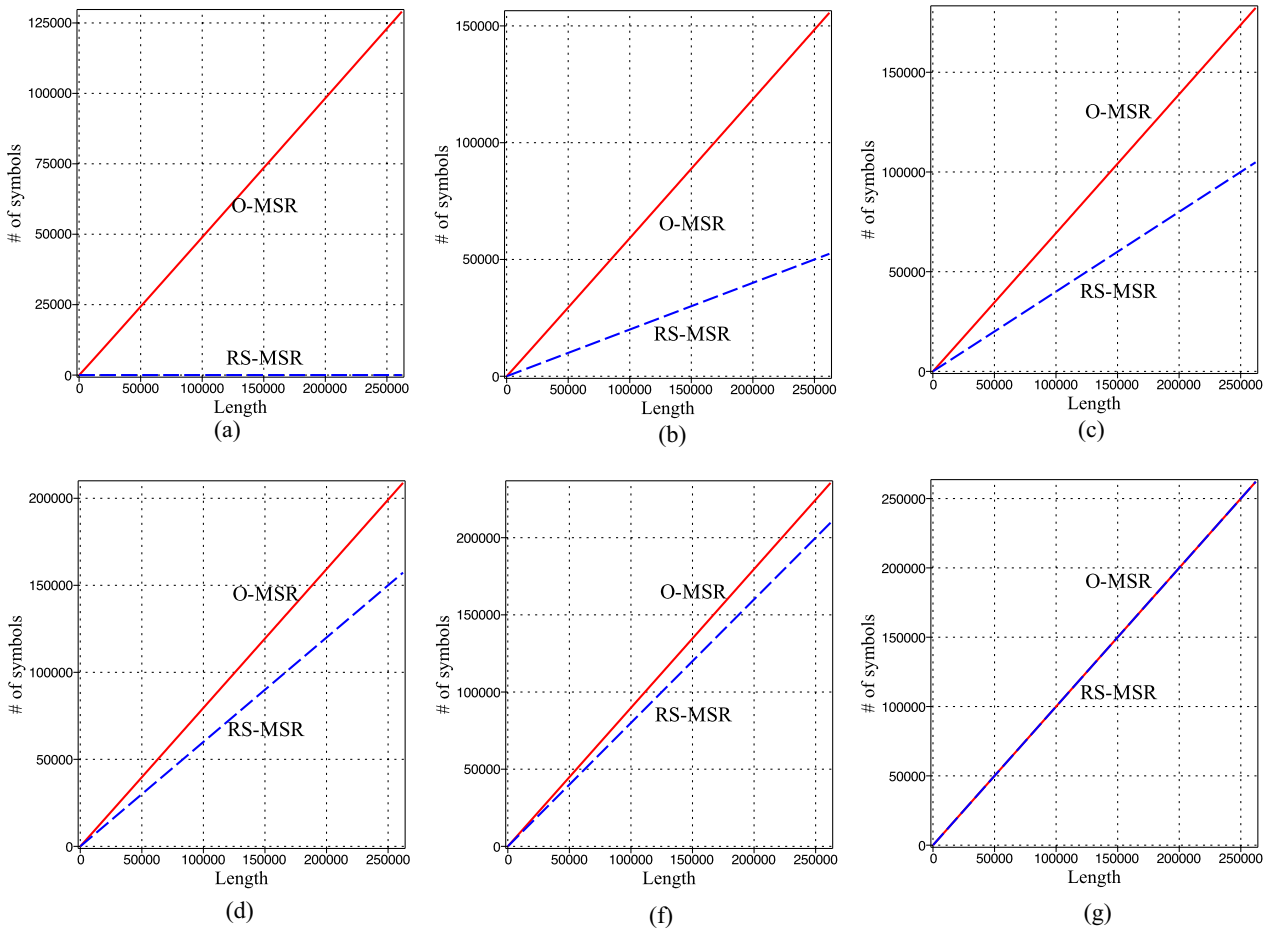


Fig. 7. Secure storage capacity comparison for $m = \lfloor \sqrt{n} \rfloor$. (a) $p = 50\%$. (b) $p = 40\%$. (c) $p = 30\%$. (d) $p = 20\%$. (e) $p = 10\%$. (f) $p = 0\%$.

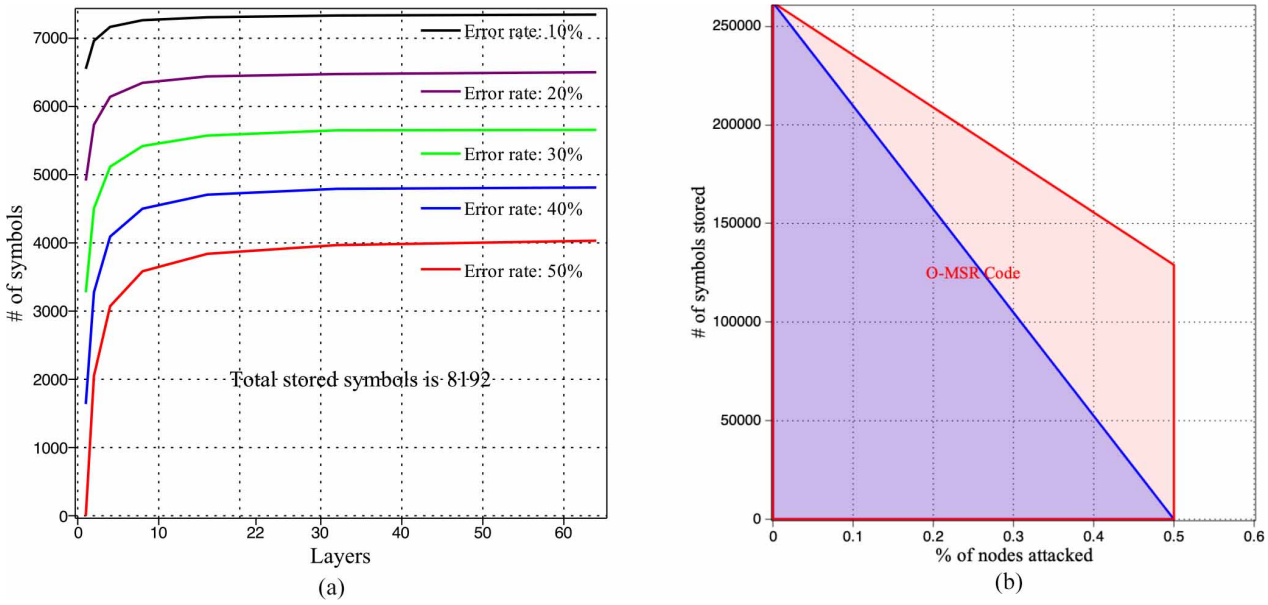


Fig. 8. Storage capacity and feasible region comparison for $m = \lfloor \sqrt{n} \rfloor$. (a) For various m and p. (b) Feasible region comparison.

size is $n = 8192$ symbols, the number of layers increases from 1 to 64. Fig. 8(b) compares the feasible data storage region for various attack probabilities between O-MSR and RS-MSR, where the total number of symbols stored is 262 144.

VI. Conclusion

We first present limitations of the existing storage system, especially the decentralized storage system. We then ana-

8 lyze our proposed multi-layer code design and applications

to distributed storage systems. We also derive the theoretical performance bound of the optimal multi-layer regenerating code. Finally, we develop an optimal DSS in adversarial attacks along with comprehensive simulation results.

REFERENCES

[1] Alied Market Research. (Apr. 2020). *Cloud Storage Market Statistics: 2027*. [Online]. Available: https://www.alliedmarketresearch.com/cloud-storage-market

[2] *Amazon Web Services (AWS)—Cloud Computing Services*. [Online]. Available: https://aws.amazon.com

[3] J. Benet. *IPFS—Content Addressed, Versioned, P2P File System (Draft 3)*. Accessed: Jul. 24, 2014. [Online]. Available: https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[4] Protocol Labs. (Aug. 14, 2017). *Filecoin: A Decentralized Storage Network*. [Online]. Available: https://filecoin.io/filecoin.pdf

[5] E. Zaghloul, T. Li, M. Mutka, and J. Ren, "Bitcoin and blockchain: Security and privacy," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10288–10313, Oct. 2020.

[6] *Uncensorable Wikipedia on IPFS*. Accessed: May 4, 2017. [Online]. Available: https://blog.ipfs.io/24-uncensorable-wikipedia/

[7] *How the Catalan Government Uses IPFS to Sidestep Spain's Legal Block*. Accessed: Sep. 29, 2017. [Online]. Available: https://news.ycombinator.com/item?id=15367531

[8] J. Zittrain, K. Albert, and L. Lessig, "Perma: Scoping and addressing the problem of link and reference rot in legal citations," *Legal Inf. Manage.*, vol. 14, no. 2, pp. 88–99, 2014.

[9] S. Rhea *et al.*, "Maintenance-free global data storage," *IEEE Internet Comput.*, vol. 5, no. 5, pp. 40–49, Sep./Oct. 2001.

[10] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. Symp. Netw. Syst. Design Implement.*, 2004, pp. 337–350.

[11] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, pp. 4539–4551, Sep. 2010.

[12] J. Ren, "On the structure of Hermitian codes and decoding for burst errors," *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2850–2854, Nov. 2004.

[13] J. Li, T. Li, and J. Ren, "Beyond the MDS bound in distributed cloud storage," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 307–315.

[14] J. Li, T. Li, and J. Ren, "Beyond the MDS bound in distributed cloud storage," *IEEE Trans. Inf. Theory*, vol. 66, no. 7, pp. 3957–3975, Jul. 2020.

[15] J. Li, T. Li, and J. Ren, "Optimal construction of regenerating code through rate-matching in hostile networks," *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4414–4429, Jul. 2017.

Jian Ren (Senior Member, IEEE) received the B.S. and M.S. degrees in mathematics from Shaanxi Normal University, Xi'an, China, in 1988 and 1991, respectively, and the Ph.D. degree in EE from Xidian University, Xi'an, in 1994.

He is a Professor with the Department of ECE, Michigan State University, East Lansing, MI, USA. His current research interests include cybersecurity, cloud computing security, distributed data sharing and storage, decentralized data management, blockchain-based e-voting and AI security, and Internet of Things.

Prof. Ren is a recipient of the U.S. National Science Foundation CAREER Award in 2009. He served as the TPC Chair of IEEE ICNC'17, the General Chair of ICNC'18, and the Executive Chair of ICNC'19 and ICNC'20. He serves as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE INTERNET OF THINGS JOURNAL, and *ACM Transactions on Sensor Networks*, and a Senior Associate Editor for *IET Communications*.

Jian Li received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 2005 and 2008, respectively, and the Ph.D. degree in electrical engineering from Michigan State University, East Lansing, MI, USA, in 2015.

He is an Associate Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing. His current research interests include network security, cloud storage, wireless sensor network in Internet of things, privacy-preserving communications, and cognitive networks.

Tongtong Li (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from Auburn University, Auburn, AL, USA, in 2000.

From 2000 to 2002, she was with Bell Laboratories, Denver, CO, USA, and had been working on the design and implementation of 3G and 4G systems. Since 2002, she has been with Michigan State University, East Lansing, MI, USA, where she is currently a Professor. Her research interests fall into the areas of wireless and wired communications, wireless security, information theory, and statistical signal processing, with applications in neuroscience.

Prof. Li is a recipient of a National Science Foundation CAREER Award in 2008 for her research on efficient and reliable wireless communications. She served as an Associate Editor for IEEE SIGNAL PROCESSING LETTERS from 2007 to 2009 and IEEE TRANSACTIONS ON SIGNAL PROCESSING from 2012 to 2016, and an Editorial Board Member for EURASIP JOURNAL ON WIRELESS COMMUNICATIONS AND NETWORKING from 2004 to 2011.

Matt W. Mutka (Fellow, IEEE) received the B.S. degree in electrical engineering from the University of Missouri–Rolla, Rolla, MO, USA, in 1979, the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1980, and the Ph.D. degree in computer sciences from the University of Wisconsin–Madison, Madison, WI, USA, in 1988.

He is a Professor with the faculty of the Department of Computer Science and Engineering, Michigan State University (MSU), East Lansing, MI, USA. He is currently serving as a Program Director with the National Science Foundation, Alexandria, VI, USA, within the Division of Computer and Networks Systems of the Directorate for Computer and Information Science and Engineering. He has been a Visiting Scholar with the University of Helsinki, Helsinki, Finland, and a member of Technical Staff with Bell Laboratories, Denver, CO, USA. His current research interests include mobile computing, sensor networking, and wireless networking.

Prof. Mutka was honored with the MSU Distinguished Faculty Award. He served as the Chairperson of the MSU Department of Computer Science and Engineering from 2007 to 2017.