Constructions of MSR Code for Optimal Node Repair in Distributed Storage

Jian Ren[1] Jian Li[2] Tongtong Li[1]
[1] Department of ECE, Michigan State University, East Lansing, MI 48824, USA.
email: {renjian, tongli}@msu.edu.
[2] School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China.
email: lijian@bjtu.edu.cn

2024 International Conference on Meta Computing (ICMC) | 979-8-3503-5599-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICMC60390.2024.00012

***Abstract*—Regenerating codes were designed to achieve an optimal trade-off between minimum node repair bandwidth and the amount of data stored per node for distributed data storage. There are two extreme points on the optimal trade-off curve, corresponding to minimum-storage regenerating (MSR) and minimum-bandwidth regenerating (MBR). The application scenarios of regenerating codes include providing reliable and efficient repair of failed nodes in distributed storage systems and peer-to-peer network storage systems. Storing data using regenerating code with node repair capability requires less redundancy while ensuring much high data availability than simple replication for the same level of reliability. Recently, Reed-Solomon (RS) code-based regenerating codes have been constructed under the product-matrix framework to achieve the maximum distance separable (MDS) property in code regeneration and reconstruction. Unfortunately, in cases where node compromising exists in the network, the storage capacity and data availability can both be significantly reduced. In this paper, we provide our detailed construction of the m-layer MSR code and analyze the theoretical bounds of m-layer codes in node repair capability and also storage while achieving minimum storage regeneration.**

***Index Terms*—regenerating code, Reed-Solomon code, optimal node repair.**

I. Introduction

Distributed storage (DS) is an important data storage and access paradigm that can achieve an efficient trade-off between storage efficiency and data availability, surpassing replication-based approaches significantly. An analysis conducted in [1] has demonstrated that, in addition to its flexibility, DS is also much more flexible and can double the storage efficiency compared to typical cloud storage solutions offered in a replicated fashion, while still achieving about the same level of data availability. Furthermore, erasure code-based DS can also provide secure and reliable data storage without requiring data encryption or any secure key management, making it an excellent solution for data centers where security is a critical requirement.

The most representative erasure coding-based technique works by encoding the data using an (n, k) Reed-Solomon (RS) code into n components in such a way that the original data can be reconstructed by any $k \leq n$ components. However, the originally stored data remains information theoretically secure for anyone who can access up to $k-1$ data components. In particular, it is infeasible to derive anything meaningful from any of the individual data components. As a result, the n data components can be stored anywhere, such as peer-to-peer networks and other types of on-demand network data storage, instead of storing the data in one single server. This approach offers an elegant trade-off between storage reliability and efficiency through adjusting of the parameter n. When individual components are stored distributively across multiple cloud storage servers, no costly data encryption and secure key management is required for any data component. The storage servers only need to ensure data availability and integrity. Furthermore, DS can increase data availability and reduce network congestion by distributing data across multiple nodes, which helps alleviate the load on individual network links and servers, leading to increased resiliency.

However, the data availability of DS heavily relies on the availability and integrity the data components. It is crucial to be able to detect and repair node failure to sustain the originally designed data availability. The concept of regenerating code was introduced to achieve optimal trade-off between minimum storage regenerating (MSR) and minimum bandwidth regenerating (MBR) [2], where a replacement node is allowed to connect to some individual nodes directly to regenerate a substitute of the failed node, instead of first recovering the original data then regenerating the failed component. Unfortunately, when malicious behaviors exist in the network, both the regeneration of the failed node and the reconstruction of the original file may fail [3]. In our previous work, we proposed a multi-layer (m-layer) code-based regenerating code [1], [4]–[6] to provide better error correction capability compared to the Reed-Solomon code based approach.

In this paper, we will the detailed design and present various design trade-offs and optimization of the multi-layer MSR code. The main contributions of this paper include:

- We propose the detailed construction of the m-layer code in encoding, and node regeneration and data construction with possible bogus nodes.
- We provide optimizations of the proposed m-layer code with respect to node repair and storage efficiency.

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

- We conduct theoretical and numerical analysis of the proposal m-layer MSR code. Our analysis and performance evaluation show that this code can double the node repair capability compared to that of the universally resilient regenerating code proposed in [3].
- The optimal construction of m-layer MSR regenerating code can also achieve optimal storage efficiency, which is much higher than that of the code proposed in [3].

The rest of this paper is organized as follows: Section II discusses the related work. The preliminary of this paper is presented in Section III. n Section IV, our proposed optimal m-layer code is presented. In Section V, various optimizations are discussed. We conduct performance analysis in Section VI. The paper is concluded in Section VII.

II. Related Work

When a storage node in the distributed storage network that employs the conventional (n, k) RS code fails, typically the replacement node first connects to k nodes and recovers the entire file, then regenerates the symbols stored in the failed node. Unfortunately, this approach is very bandwidth inefficient because the entire file has to be recovered to reproduce a fraction of it. To overcome this drawback, Dimakis *et al.* introduced a regenerating code based on network coding, denoted by $\{n, k, d, \alpha, \beta, B\}$ [2]. In the context of regenerating code, a data collector (DC) can reconstruct the original file stored in the network by downloading α symbols from each of k storage nodes. A replacement node can be regenerated by downloading β help symbols from each of d helper nodes. This gives the total bandwidth required to regenerate a failed node, $\gamma = d\beta$, which can be far less than the entire file B. In their paper, Dimakis *et al.* demonstrated that a trade-off exists between the bandwidth γ and per-node storage α. They found two optimal points on the optimal trade-off curve: minimum storage regeneration (MSR) and minimum bandwidth regeneration (MBR) points. Existing work has largely focused on the optimal design of regenerating codes [7]–[11], and implementation of the regenerating code.

The regenerating code can be divided into functional regeneration and exact regeneration. In the functional regeneration, the replacement node regenerates a new component that can functionally replace a failed component instead of being the same as the originally stored component [12]–[15]. In [12], the data regeneration was formulated as a multicast network coding problem. A random linear regenerating code for distributed storage systems was implemented in [13]. In [14], it has been proved that by allowing data exchange among the replacement nodes, a better trade-off between repair bandwidth γ and per node storage α can be achieved. In [15], a functional regenerating code with less computational complexity through binary operations was proposed. In the exact regeneration, the replacement node regenerates the exact symbols of a failed node [16]–[20]. In [16], the authors proposed to reduce the regeneration bandwidth through algebraic alignment. A code structure for exact regeneration using interference alignment technique was provided in [17]. In [18], an optimal exact constructions of MBR codes and MSR codes under product-matrix framework was presented. This is the first work that allows independent selection of the node number n in the network. In [19], repair performance of the Reed-Solomon codes was studied. A code construction that could achieve performance better than space-sharing between the minimum storage regenerating codes and the minimum bandwidth regenerating codes was proposed in [20].

However, none of these works considered code regeneration under possible node corruption or adversarial manipulation attacks in hostile networks. In fact, all these schemes would fail in both regeneration and reconstruction if some storage nodes could provide incorrect responses to the requests.

For general verification of the contents stored in distributed storage, several schemes have been proposed [21]–[24]. In [21], the authors amnalyzed the verification cost for both client read and write operations in workloads with idle periods. In [22], erasure coding and threshold cryptography were proposed to achieve storage efficiency and resilience. To check data integrity of the regenerating codes in hostile networks, CRC codes were adopted in [23]. Unfortunately, CRC checks can be easily manipulated by malicious nodes, resulting in regeneration and reconstruction failures. In [24], data integrity protection (DIP) was designed under a mobile Byzantine adversarial model to enable a client to verify the integrity of outsourced data against general or malicious corruptions in distributed storage.

There are some publications that discussed corrupted node detection and correction in regenerating codes [25]–[31]. In [25], the Byzantine fault tolerance of regenerating codes was studied. The amount of information that can be safely stored against passive eavesdropping and active adversarial attacks based on the regeneration structure was discussed in [26]. In [27], the universally secure regenerating code was developed to achieve information theoretic data confidentiality. However, the paper did not consider the extra computational cost and bandwidth for this code. In [28], the authors discussed the optimal trade-off between the storage space and the repair bandwidth in presence of two types of wiretapper. In [29], the authors revealed some general properties of MSR codes and a generally applicable upper bound on secrecy capacity with passive eavesdroppers in the storage network. A secure MSR coding scheme that could overcome the limitations of previous eavesdropper model was proposed in [30]. The achievable trade-off regions between the normalized storage capacity and repair bandwidth for the secure exact-repair regenerating codes against an eavesdropper were studied in [31].

There are many other research publications focusing on the security of the decentralized storage networks [32]–[36]. Nevertheless, since regenerating codes in these works are all an

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

extension of the maximum distance separable (MDS) code, the error correction capability is constrained by the MDS bound. Moreover, none of the schemes presented is able to determine whether the errors in network are successfully corrected.

III. Preliminary and Assumptions

A. Regenerating Code

Regenerating code was first introduced in [2]. It is a linear network code over the finite field $\mathbb{F}_q$ with 5 parameters $\{n, k, d, \alpha, \beta, B\}$, where q is a prime number or some power of a prime number. A file of size $B = \alpha k$ is stored in n storage nodes, each of which stores α symbols. The file can be reconstructed from any k randomly selected storage nodes. A replacement node can regenerate the contents of a failed node by downloading $\beta \leq \alpha$ symbols from each of $d \geq k$ randomly selected storage nodes, which makes the total bandwidth needed to regenerate a failed node $\gamma = d\beta$. In [2], the following theoretical bound was derived for regenerating codes:

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

From equation (1), a trade-off between the regeneration bandwidth γ and the per node storage α was derived, shown in Fig. 1. There are two special cases: the minimum storage regeneration (MSR) point, where the storage parameter α is minimized:

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

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

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

B. Adversarial Model

In this paper, our adversarial model is similar to the one adopted in [3]. We assume that some network nodes may be corrupted due to hardware failure or communication errors, and/or be controlled by malicious users. As a result, upon request, these nodes may provide incorrect responses to disrupt data regeneration and reconstruction. These incorrect responses are described as errors/erasures in [3]. We assume that malicious users can take full control of up to τ ($\tau \leq n$) storage nodes and perform possible collusion attacks. Other than the information stored in the compromised nodes, the adversary is unable to obtain the contents or the distribution of the encoding vectors in other intact nodes.

We will refer the corrupted and the compromised symbols as *bogus* symbols. We will also use corrupted nodes, bogus nodes, malicious nodes and compromised nodes interchangeably throughout the paper. The maximum number of bogus node from which errors can be corrected is referred to as the *node correction capability*.

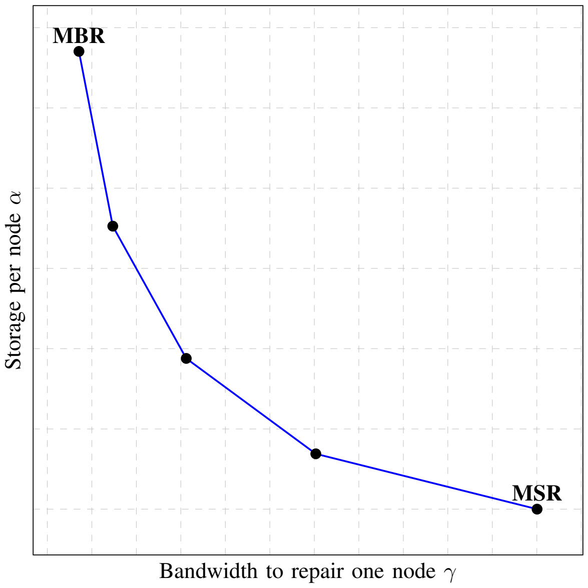

Fig. 1: MSR and MBR tradeoff in regenerating code.

In this paper, we assume the existence of a secure server that is responsible for encoding and distributing data to storage nodes. The secure server initialize replacement nodes. The DC and the secure server can be implemented on the same computer. We assume that this server will never be compromised.

IV. Proposed m-layer Regenerating Code (O-MSR)

In this section, we present the construction of the m-layer MSR code from m layers of MSR codes with symbols from the Galois field $\mathbb{F}_q$, where q is a prime number of some power of a prime number, and $n = q$ or $q-1$. For each $i = 1, \cdots, m$, the i^{th} layer MSR code L_i is represented by $(n, k_i, d_i, \alpha_i, \beta_i, B_i)$. It can be viewed as an $(n, d_i, n-d_i+1)$ MDS code, where $\alpha_i = k_i - 1$, $d_i = 2k_i - 2$, and $d_i \leq d_j$ for all $1 \leq i \leq j \leq m$.

A. Encoding

The MSR code $\{n, k_i, d_i, \alpha_i, \beta_i, B_i\}$ is encoded based on the product-matrix code framework proposed in [18]. According to equation (2), we have $\alpha_i = d_i/2$, $\beta_i = 1$ for one block of data with the size $B_i = k_i\alpha_i = (\alpha_i+1)\alpha_i$. The $d_i \times \alpha_i$ message matrix M_i is defined as

$$M_i = \begin{bmatrix} S_{i,1} \\ S_{i,2} \end{bmatrix}, \quad i = 1, \cdots, m, \tag{4}$$

where $S_{i,1}$ and $S_{i,2}$ are both $\alpha_i \times \alpha_i$ symmetric matrices, each of which will contain $\alpha_i(\alpha_i+1)$ data symbols. We further define the $n \times d_i$ encoding matrix

$$\Psi = [\Phi \quad \Lambda\Phi],$$

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

where

$$\Phi = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & g & g^2 & \dots & g^{\alpha_i - 1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & g^{n-1} & (g^{n-1})^2 & \dots & (g^{n-1})^{\alpha_i - 1} \end{bmatrix} \tag{5}$$

is an $n \times \alpha_i$ Vandermonde matrix and $\Lambda = \text{diag}[\lambda_1, \cdots, \lambda_n]$ is an $n \times n$ diagonal matrix such that $\lambda_i \in \mathbb{F}_q$ and $\lambda_i \neq \lambda_j$ for $1 \leq i, j \leq n, i \neq j$, g is a primitive element in $\mathbb{F}_q$, and any d_i rows of Ψ are linearly independent.

The codeword F_i is defined as

$$F_i = \Psi M_i = [\Phi \quad \Lambda\Phi] \begin{bmatrix} S_{i,1} \\ S_{i,2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\psi}_1 M_i \\ \vdots \\ \boldsymbol{\psi}_n M_i \end{bmatrix} = \begin{bmatrix} f_{i,1} \\ \vdots \\ f_{i,n} \end{bmatrix}. \tag{6}$$

Each row $f_{i,j} = \boldsymbol{\psi}_j M_i$ $(1 \leq i \leq n)$ of the codeword matrix F_i will be stored in storage node j, where the encoding vector $\boldsymbol{\psi}_j$ is the j^{th} row of Ψ.

To encode a file with size $B = \sum\limits_{i=1}^{m} \alpha_i(\alpha_i + 1)$ using the m-layer MSR code, the file will first be divided into message matrices M_i with file sizes $\alpha_i(\alpha_i + 1)$, for $i = 1, \cdots, m$, respectively. Then we encode message matrices M_i into the codeword matrices F_i based on equation (6).

To conceal the layer information and prevent malicious attackers from selectively corrupting the MSR code, the secure server will randomly concatenate all the matrices together to form the final $n \times (\alpha_1 + \cdots + \alpha_m)$ codeword matrix:

$$\mathcal{C} = [\mathsf{Perm}(F_1, \ldots, F_m)], \tag{7}$$

where Perm denotes a random matrix permutation operation. The secure sever will also record the order of the permutation for future code regeneration and reconstruction. For simplicity of the presentation, we will ignore the permutation operation in the subsequent sections of the paper. Then each row $\mathsf{f}_i = [f_{1,i}, \cdots, f_{m,i}]$, for $1 \leq i \leq n$, of the codeword matrix $\mathcal{C}$ will be stored in storage node i, where $f_{j,i}$ represents the i^{th} row of codeword F_j $(1 \leq j \leq m)$. The total number of symbols stored in each node is $\alpha_1 + \cdots + \alpha_m$.

We have the following Theorem.

Theorem 1. The encoding of the m-layer code presented from equations (4) to (7) can achieve the MSR point in defined in equation (2), since all the sublayer codes defined in equation (6) are RS-MSR codes [4], [5].

Remark 1. The permutation operation is designed to prevent adversaries from identifying individual layers. For application scenarios where node failure can only be caused by hardware failures or communication errors, we can directly concatenate all the codeword matrices without the permutation operation.

B. Node Repair and Regeneration

Suppose node z fails, the security server with the permutation information of the m layers of MSR code will initialize a replacement node z'. The replacement node z' will send regeneration requests to the rest of the $n - 1$ helper nodes. Upon receiving the regeneration request, helper node i will calculate and send out the help symbols

$$\mathbf{h}_i = [f_{1,i}\boldsymbol{\phi}_z^t, \cdots, f_{m,i}\boldsymbol{\phi}_z^t] = [\boldsymbol{\psi}_i M_1 \boldsymbol{\phi}_z^t, \cdots, \boldsymbol{\psi}_i M_m \boldsymbol{\phi}_z^t],$$

where $\boldsymbol{\phi}_z$ is the z^{th} row of Φ and $\boldsymbol{\phi}_z^t$ is the transpose of $\boldsymbol{\phi}_z$.

Let $\Psi_{i \to j} = \left[\boldsymbol{\psi}_i^t, \boldsymbol{\psi}_{i+1}^t \cdots, \boldsymbol{\psi}_j^t\right]^t$, for $1 \leq i \leq j \leq m$, and $\mathbf{x}_i^{(l)}$ be the vector containing the first l symbols of $M_i \boldsymbol{\phi}_z^t$, where $l = 1, \cdots, m$.

Suppose $\mathbf{h}_i' = \mathbf{h}_i + \mathbf{e}_i = [h_{1,i} + e_{1,i}, \cdots, h_{m,i} + e_{m,i}]$ is the response from helper node i, where $\mathbf{h}_i = [h_{1,i}, \cdots, h_{m,i}]$ and $\mathbf{e}_i = [e_{1,i}, \cdots, e_{m,i}]$. If $\mathbf{e}_i = [e_{1,i}, \cdots, e_{m,i}] \in \mathbb{F}_q^m \backslash \{\mathbf{0}\}$, then node i is corrupted since the response $\mathbf{h}_i$ has been modified.

Without loss of generality, we may assume that $i = n$, and define

$$\underline{\mathbf{h}}_l' = \underline{\mathbf{h}}_l + \underline{\mathbf{e}}_l = [h_{l,1} + e_{l,1}, \cdots, h_{l,n-1} + e_{l,n-1}],$$

where $\underline{\mathbf{h}}_l = [h_{l,1}, \cdots, h_{l,n-1}]$ and $\underline{\mathbf{e}}_l = [e_{l,1}, \cdots, e_{l,n-1}]$, for $l = 1, \cdots, m$.

To regenerate the symbols of the failed node z, we will regenerate the symbols layer by layer, starting from layer 1 to derive $h_{1,i}$ and $e_{1,i}$. We can successfully regenerate the symbol in node $h_{1,i}$ when the total number of received help symbols $h_{1,i}'$ being modified from the n helper nodes is less than $\lfloor (n - 2\alpha_1 - 1)/2 \rfloor$, where $\lfloor x \rfloor$ is the floor operation of x, which represents the greatest integer less than or equal to x. In other words, we can correct

$$t_1 = \lfloor (n - 2\alpha_1)/2 \rfloor = (n - 2\alpha_1 - \varepsilon_1)/2, \tag{8}$$

where $\varepsilon_1 = 0$ or 1 depending on whether n is even or odd.

Then we move to the j^{th} layer to derive $h_{j,i}$ and $e_{j,i}$, for $j = 2, \cdots, m$. By treating the symbols from the $\underline{\mathbf{h}}_j$ layer where errors have been found by $\underline{\mathbf{h}}_{j-1}$ as erasures, the j^{th} layer code $\underline{\mathbf{h}}_j$ can correct $t_j - 1$ erasures and $\lfloor (n - 2\alpha_i - t_{i-1})/2 \rfloor$ new errors [37]:

$$\begin{aligned} t_j &= \lfloor (n - 2\alpha_j - t_{j-1})/2 \rfloor + t_{j-1} \\ &= (n - 2\alpha_j - t_{j-1} - \varepsilon_j)/2 + t_{j-1} \\ &= \left(\sum_{l=1}^{j} 2^{l-1}(n - 2\alpha_l) - \sum_{l=1}^{j} 2^{l-1}\varepsilon_l \right) / 2^j, \end{aligned} \tag{9}$$

where $\varepsilon_i = 0$ or 1, with the restriction that $n - 2\alpha_i \geq t_{i-1}$, which can be written as:

$$-\sum_{l=1}^{j-1} 2^l \alpha_l + 2^j \alpha_j \leq n + \sum_{l=1}^{j-1} 2^{l-1}\varepsilon_l. \tag{10}$$

Without loss of generality, we assume $1 \leq i \leq n$. z' will perform Algorithm 1 to regenerate the contents of the failed node z.

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1. z' regenerates symbols of the failed node z.

```
Step 1: Let $\mathbf{h}_i' = [h_{1,i}+e_{1,i},\cdots,h_{m,i}+e_{m,i}]$, for $i = 1,\cdots,n-1$ be the response from helper nodes $i$. Define $\underline{\mathbf{h}}_i' = [h_{1,i}+e_{1,i},\cdots,h_{n-1,i}+e_{n-1,i}]$.
Step 2: Decode $\underline{\mathbf{h}}_1'$ using the decoding algorithm for RS code to $\underline{\mathbf{h}}_1$ [38], where $\underline{\mathbf{h}}_1' = [h_{1,1}',\cdots,h_{n-1,1}']^t$ can be viewed as an MDS code with parameters $(n, 2\alpha_1, n-2\alpha_1)$ since $\Psi_{1|1\to(n-1)}\cdot\mathbf{x}^{(n-1)} = \underline{\mathbf{h}}_1'$. Suppose the error locations are $\otimes_{1,1},\cdots,\otimes_{1,j_1}$.
Step 3: Decode $\underline{\mathbf{h}}_i'$ using the erasure decoding algorithm for RS code with parameters described in equation (9) for $i = 2,\cdots,m$, where $\underline{\mathbf{h}}_i' = [h_{1,i}',\cdots,h_{n-1,i}']^t$ can be viewed as an MDS code with parameters $(n-1, d_i, n-d_1)$ since $\Psi_{i|1\to(n-1)}\cdot\mathbf{x}^{(n-1)} = \underline{\mathbf{h}}_i'$. Suppose the error locations are $\otimes_{j,1},\cdots,\otimes_{j,j_i}$.
Step 4: Compute $\mathsf{f}_z = [f_{1,z},\cdots,f_{m,z}] = [\Phi_z S_{1,1} + \lambda_z\Phi_z S_{1,2},\cdots,\Phi_z S_{m,1} + \lambda_z\Phi_z S_{m,2}]$ as described in [18].
```

Proposition 1. For the m-layer regenerating code, if $\alpha_i \le \alpha_j$, for all $1 \le i \le j \le m$ and $n - 2\alpha_i \ge t_{i-1}$ as defined in equation (9), then the m-layer code can correct errors from $\lfloor (n-2\alpha_m)/2 \rfloor$ corrupted nodes in data regenerating, where $\lfloor x \rfloor$ is the floor operation.

C. Node Repair and Data Reconstruction

When the DC needs to reconstruct the original file, it will send reconstruction requests to n storage nodes. Upon receiving the request, node i will send out the symbol vector $\mathbf{h}_i' = \mathbf{h}_i + \mathbf{e}_i = [h_{1,i}+e_{1,i},\cdots,h_{m,i}+e_{m,i}]$ to the DC. If $\mathbf{e}_i \in \mathbb{F}_q^m\backslash\{\mathbf{0}\}$, then node i is corrupted since the response $\mathbf{h}_i' \ne \mathbf{h}_i$.

The DC will reconstruct the file as follows: Let

$$R' = \begin{bmatrix} f_{1,1} & \cdots & f_{m,1} \\ \vdots & \ddots & \vdots \\ f_{1,n} & \cdots & f_{m,n} \end{bmatrix} = [\Phi \quad \Lambda\Phi] \begin{bmatrix} S_{1,1}' & \cdots & S_{m,1}' \\ S_{1,2}' & \cdots & S_{m,2}' \end{bmatrix},$$

we have

$$R' = [\Phi S_{1,1}' + \Lambda\Phi S_{1,2}',\cdots,\Phi S_{m,1}' + \Lambda\Phi S_{m,2}'].$$

Let $R'\Phi^t = [\widehat{R}_1',\cdots,\widehat{R}_m']$, and $C^{(i)} = \Phi S_{i,1}'\Phi^t$, $D^{(i)} = \Phi S_{i,2}'\Phi^t$, $i = 1,\cdots,m$, then

$$C^{(i)} + \Lambda D^{(i)} = \widehat{R}_i'.$$

Since $C^{(i)}, D^{(i)}$ are both symmetric, we can solve the non-diagonal elements of $C^{(l)}, D^{(l)}$ as follows:

$$\begin{cases} C_{j,k}^{(i)} + \lambda_i \cdot D_{j,k}^{(i)} = \widehat{R}_{j,k}' \\ C_{j,k}^{(i)} + \lambda_j \cdot D_{j,k}^{(i)} = \widehat{R}_{k,j}'. \end{cases}$$

Because matrices $C^{(i)}$ and $D^{(i)}$ have the same structure, we only need to focus on $C^{(i)}$, which corresponds to $S_{i,1}'$.

It is straightforward to see that if node i is corrupted and there are errors in the i^{th} row of R', there will be errors in the i^{th} row of $\widehat{R}'$. As a result, there will be errors in the i^{th} row and i^{th} column of $C^{(i)}$.

We can view each column of $C^{(i)}$ as an $(n, \alpha_i, n-\alpha_i+1)$ MDS code because Φ is a Vandermonde matrix. The length of the code is n since the diagonal elements of $C^{(i)}$ is unknown. Suppose node j is a legitimate node, we can decode the MDS code to recover the j^{th} column of $C^{(i)}$ and locate the corrupted nodes. The decoding algorithm is similar to Algorithm 1 for symbol regenerating. We can correct

$$t_1 = \lfloor (n-\alpha_1)/2 \rfloor = (n-\alpha_1-\varepsilon_1)/2, \tag{11}$$

where $\varepsilon_1 = 0$ or 1 depending on whether $n - \alpha_1$ even or odd.

For $i = 2,\cdots,m$, by treating the symbols that the t_{i-1} node where errors are found by $C^{(i-1)}$ as erasures, the i^{th} layer can correct t_{i-1} erasures and $\lfloor (n-\alpha_i-t_{i-1})/2 \rfloor$ new errors:

$$\begin{aligned} t_i &= \lfloor (n-\alpha_i-t_{i-1})/2 \rfloor + t_{i-1} \\ &= (n-\alpha_i-t_{i-1}-\varepsilon_i)/2 + t_{i-1} \\ &= \left(\sum_{j=1}^{i} 2^{j-1}(n-\alpha_j) - \sum_{j=1}^{i} 2^{j-1}\varepsilon_j \right)/2^i, \end{aligned} \tag{12}$$

In this way, $C^{(i)}$, $i = 1,\cdots,m$, can be recovered. So the DC can reconstruct $S_{i,1}$ using the method similar to [4], [5]. For $S_{i,2}$, the recovering process is similar.

Proposition 2. For the m-layer regenerating code, if $\alpha_i \le \alpha_j$, for $1 \le i \le j \le m$, and $n - \alpha_i \ge t_{i-1}$, then the m-layer code can correct errors from $\lfloor (n-\alpha)/2 \rfloor$ corrupted nodes in data reconstruction.

V. Optimizations

The optimization problem for bogus node repair has been discussed in [1], [6]. In this paper, we will continue the research along this line and provide refined considerations.

A. Optimizing Node Repair and Regeneration Capability

According to Section IV-B, the i^{th} layer MSR code F_i can be viewed as an $(n, d_i, n-d_i+1)$ MDS code for $1 \le i \le m$ during regenerating. In the optimization, we can set the summation of the α_i's of all the layers to a constant α_0:

$$\sum_{i=1}^{m} \alpha_i = \alpha_0,$$

and

$$\alpha_{i-1} - \alpha_i \le 0, \text{ for } 2 \le i \le m.$$

We can then maximize the number of the bogus node regenerating capability of the m-layer MSR code by maximizing

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

t_m. With all the constrains listed above, the optimization problem can be written as:

$$\begin{aligned}
&\text{maximize } t_m = \frac{1}{2^m}\left(\sum_{i=1}^{m} 2^{i-1}(n-2\alpha_i) - \sum_{i=1}^{m} 2^{i-1}\varepsilon_i\right), \\
&\text{subject to } \sum_{i=1}^{m} \alpha_i = \alpha_0, \\
&\qquad \alpha_{i-1} - \alpha_i \le 0, \quad i = 2, \cdots, m, \\
&\qquad 0 \le \varepsilon_i \le 1, \quad i = 1, \cdots, m, \\
&\qquad -\sum_{j=1}^{i-1} 2^j \alpha_j + 2^j \alpha_i \le n + \sum_{j=1}^{i-1} 2^{j-1}\varepsilon_j, \\
&\qquad\quad 2 \le i \le m.
\end{aligned} \tag{13}$$

For this optimization, we can introduce slack variables $\varepsilon_i, i = 1, \cdots, m$ to convert it into a linear programming optimization. The optimization result can be summarized as follows:

Theorem 2. For the m-layer MSR regeneration code, when

$$\alpha_i = \text{round}(\alpha_0/m) = \widetilde{\alpha}, \text{ for } 1 \le i \le m, \tag{14}$$

it can achieve maximum node repair capability, which is

$$t_m = \left((2^m - 1)(n - 2\widetilde{\alpha}) - \sum_{i=1}^{m} 2^{i-1}\varepsilon_i\right)/2^m \tag{15}$$

corrupted nodes.

In fact, by selecting n to be even, we can make $n - 2\widetilde{\alpha}$ even, we can ensure that $\varepsilon_i = 0$ for $i = 1, \cdots, m$, thereby maximizing the corrupt node detection and regenerating capability while maintaining the given storage efficiency as follows:

$$\begin{aligned}
t_m &= \left((2^m - 1)(n - 2\widetilde{\alpha})\right)/2^m \\
&= \left(1 - \frac{1}{2^m}\right)(n - 2\widehat{\alpha}).
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
\lim_{m\to\infty} t_m &= \lim_{m\to\infty}\left(1 - \frac{1}{2^m}\right)(n - 2\widehat{\alpha}) \\
&= n - 2\widehat{\alpha}.
\end{aligned} \tag{16}$$

In the worst case scenario, when $\varepsilon_i = 1, i = 1, \cdots, m$, then

$$t_m = \left((2^m - 1)(n - 2\widetilde{\alpha} - 1)\right)/2^m.$$

Similar to equation (16), we have

$$\begin{aligned}
\lim_{m\to\infty} t_m &= \lim_{m\to\infty}\left(1 - \frac{1}{2^m}\right)(n - 2\widehat{\alpha} - 1) \\
&= n - 2\widehat{\alpha} - 1.
\end{aligned} \tag{17}$$

In both scenarios, the node repair capability for the O-MSR doubles that of the RS-MSR, which is summarized in the following theorem.

Theorem 3. The node repair capability of the m-layer O-MSR code converges to twice that of the RS-MSR as m increases.

Theorem 3 states that the overhead required to repair random node failure for the O-MSR code approaches only the overhead required for erasure correction, which is only half the overhead required to repair random errors.

From now on, the MSR code generated this way will be referred to as the *optimal m-layer MSR regenerating code*, and denoted as *O-MSR*.

B. Optimizing Code Rate Given Node Repair Capability

In the previous optimization, we set the code rate of the O-MSR code to a constant and then maximize the node repair node capability in node regeneration. Alternatively, we can also optimize the storage efficiency of the MSR code, given the node repair capability in node regeneration defined through t_m in equation (9), to a constant value t_0, while maximizing the code rate of the MSR regenerating code. The optimization problem can be presented as follows:

$$\begin{aligned}
&\text{maximize } \sum_{i=1}^{m} \alpha_i, \\
&\text{subject to } -\sum_{j=1}^{i-1} 2^j \alpha_j + 2^j \alpha_i \le n + \sum_{j=1}^{i-1} 2^{j-1}\varepsilon_j, \\
&\qquad 2 \le i \le m, \\
&\qquad \alpha_{i-1} - \alpha_i \le 0, \quad 2 \le i \le m, \\
&\qquad t_m = \sum_{j=1}^{m} 2^{j-1-m}(n - 2\alpha_j) - \sum_{j=1}^{m} 2^{j-1-m}\varepsilon_j.
\end{aligned} \tag{18}$$

The optimization result shows that when m is sufficiently large, then easy layer can roughly store t_0 more symbols.

Remark 2. We can also optimize the proposed O-MSR performance based on any particular attack scenarios to achieve optimum performance.

C. Overhead of MSR Code

The concept of the *overhead* of an MSR code δ_{MSR} is defined as the amount of information that needs to be transferred compared to the fragment size B/k. In our proposed O-MSR design, we have

$$\delta_{\mathsf{MSR}} = \frac{d\beta_{\mathsf{MSR}}}{\alpha} = \frac{2\alpha}{\alpha} = 2. \tag{19}$$

The performance gain of the O-MSR for node regenerating, compared to data reconstruction and then regenerate the node is defined as:

$$\mathsf{gain} = \frac{B}{\gamma_{\mathsf{MSR}}} = \frac{k(d-k+1)}{d} = \frac{k}{2}. \tag{20}$$

This is true regardless of the k and d values and node repair capability of the MSR code as long as they are selected to satisfy the following requirements:

$$1 \le d_i = 2k_i - 2 \le n.$$

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Comparison of node repair capability for $n = 16$, with the number of layers m varying from 1 to 4.

Layers / k/d			Gain	Node repair capability				δ_S
				$m=1$	$m=2$	$m=3$	$m=4$	
$k=2$	$\alpha=1$	$d=2$	1	7	10	12	13	1/8
$k=3$	$\alpha=2$	$d=4$	3/2	6	9	10	11	3/16
$k=4$	$\alpha=3$	$d=6$	2	5	7	8	9	1/4
$k=5$	$\alpha=4$	$d=8$	5/2	4	6	7	7	5/16
$k=6$	$\alpha=5$	$d=10$	3	3	5	5	5	3/8
$k=7$	$\alpha=6$	$d=12$	7/2	2	3	3	3	7/16
$k=8$	$\alpha=7$	$d=14$	4	1	1	1	1	1/2

Therefore, we have

$$2 \leq k_i \leq \frac{n}{2} + 1, \quad 1 \leq \alpha_i \leq \frac{n}{2},$$

for $i = 1, \cdots, m$. The corresponding MDS codes for node repair and node reconstruction are $(n, d_i, n - d_i + 1)$ and $(n, \alpha_i, n - \alpha_i + 1)$, respectively.

We can also analyze the *storage efficiency* δ_S, which is defined as the ratio between the actual size of data to be stored and the total storage space needed by the encoded data in [6]. Then we have:

$$\delta_S = \frac{B}{n\alpha} = \frac{\sum_{i=1}^{m} \alpha_i(\alpha_i + 1)}{n \sum_{i=1}^{m} \alpha_i}.$$

Table I compares the node repair capability for $n = q = 2^4$, with the number of layers m varying from 1 to 4 and $1 \leq k_i \leq 7, i = 1, \cdots, m$. Table I also provides the number of symbols that will be stored in each node and the total number of symbols stored. From the table, it is evident that as m increases, the node repair capability approaches twice that of when $m = 1$. This trend works also holds true to other n and m cases.

We also observed that as n increases, the number of layers required to reach the optimal node repair capability may also increase.

VI. Performance Analysis

In this section, we analyze the performance of the m-layer O-MSR code and compare it with the performance of the RS-MSR code regarding the repair capability and the storage efficiency.

A. Number of Layers and Node Repair Capability

To demonstrate the improvement in node repair capability with the increasing number of layers, we select $q = 2^6$, $n = q - 1 = 63$, and the code dimension $k = 30$ for a single layer. We compare the maximum number fo bogus node from which errors can be corrected when the number of layers m increase from 1 to 6 while maintaining the overall code rate. The node repair capability is shown in Fig. 2. From this figure, we can

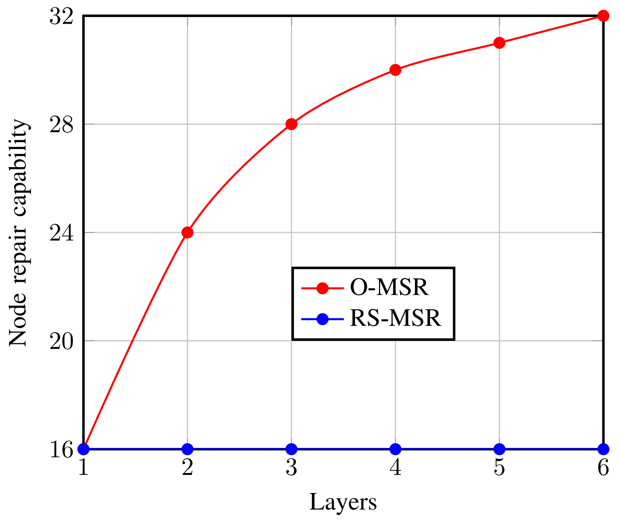

Fig. 2: Node repair capability increases with the number of layers.

see that when $m = 6$, the m-layer O-MSR can double the corrupted node detection and corrected capability compared to the single layer RS-MSR, which represents the theoretical optimal upper bound.

Fig. 3 compares the bogus node regeneration capability between O-MSR and RS-MSR when the total number of symbols stored in the m-layer code is 32 symbols and in the single-layer RS-MSR code is $\lfloor 32/m \rfloor$, respectively, with m ranging from 1 to 16. This simulation again shows that the performance of the m-layer code doubles that of the single layer RS-MSR code.

B. Code Length and Node Repair Capability

In this simulation, we choose $n = q^2$, $m = 6$, and code rates of $3/4$ and $1/2$. As q increases from 4 to 16, in both cases, the node regenerating capability for the O-MSR code increases, reaching nearly double that of the RS-MSR code, as demonstrated in Fig. 4. We also found that this trend holds for all code rates.

From Fig. 4, we can also observe that the node regenerating capability for O-MSR at a code rate of 3/4 nearly overlaps with the RS-MSR at a code rate of 1/2. This observation suggests that by maintaining the same node regenerating capability, O-MSR can increase storage efficiency by 50%. This alternative approach demonstrates the performance improvement achievable with O-MSR codes.

C. Storage Capacity Improvement over Varying Node Repair Capability

We also conducted simulations to compare the reliable data storage capacity under varying given node failure and repair capability. Specifically, we compared the storage capacity of m-layer O-MSR codes and RS-MSR codes in two different

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

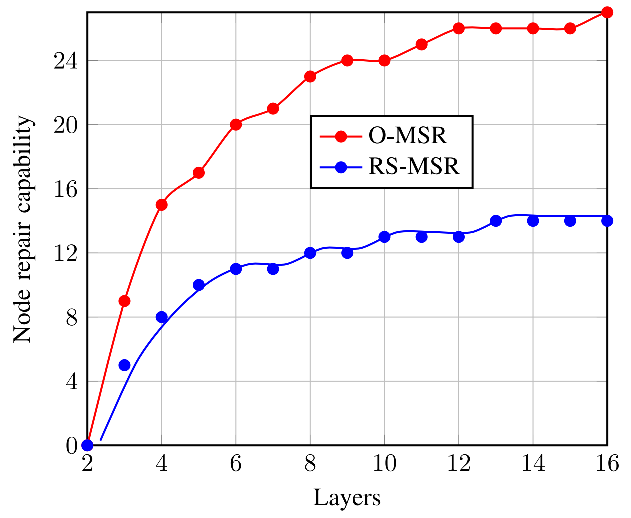

Fig. 3: Comparison of bogus node regenerating capability across varying layers between O-MSR and RS-MSR, where each single-layer node stores $\lfloor 32/m \rfloor$ symbols and m variess from 1 to 16.

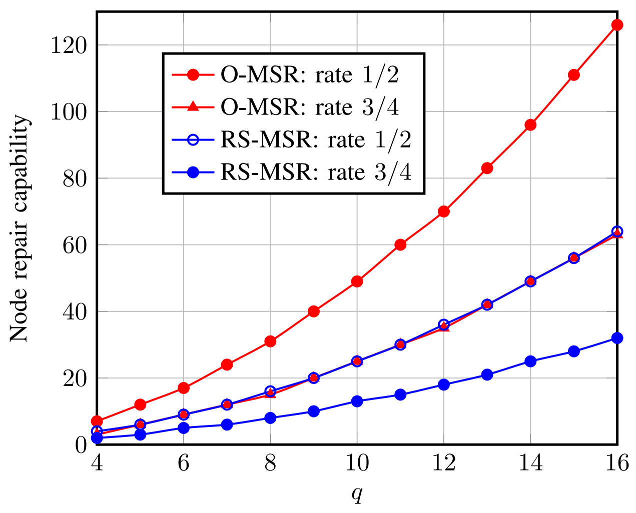

Fig. 4: Comparison of bogus node repair capability for $m = 6$, $n = q^2$ with q values from 4 to 16.

settings: (i) with $n = q = 2^5$, $m = 4$, and the number of nodes that need to be repaired varies from 3 to 15, and (ii) with $n = q = 2^6$, and $m = 4$, and the number of nodes that need to be repaired varies from 3 to 31. The results are illustrated in Fig. 6. It can observed from Fig. 6 that as the number of node failures approaches 50% of the nodes, the reliable data storage capacity of RS-MSR approaches 0, whereas the m-layer O-MSR can still reliably store about 50% of data symbols in each layer, which is consistent with the theoretical results provided in Section V-B.

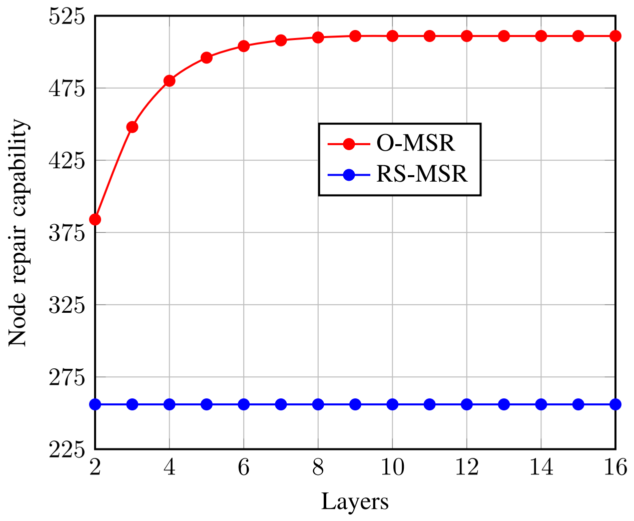

Fig. 5: Bogus node repair capability comparison.

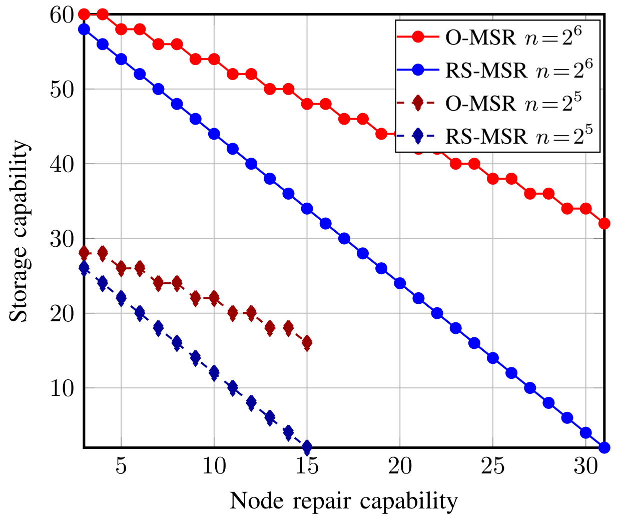

Fig. 6: Node repair capability increases with the number of layers.

VII. Conclusion

In this paper, we present the detailed construction of our proposed m-layer minimum storage regenerating (MSR) code for distributed storage. We discuss optimizations of the construction, focusing on both node repair capability and storage efficiency. Theoretical analysis shows that the optimal m-layer MSR code can nearly double the capacity in node repair for node regeneration while maintaining the same storage efficiency. Alternatively, we can increase the data storage efficiency by nearly 50% when the number of failed nodes approaches close to 50%.

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.

Acknowledgment

This work was supported in part by NSF under Grant CCF-1919154 and Grant ECCS-1923409.

References

[1] J. Ren, J. Li, T. Li, and M. Mutka, "Feasible region of secure and distributed data storage in adversarial networks," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8980–8988, 2022.

[2] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, pp. 4539 – 4551, 2010.

[3] K. Rashmi, N. Shah, K. Ramchandran, and P. Kumar, "Regenerating codes for errors and erasures in distributed storage," in *International Symposium on Information Theory (ISIT) 2012*, pp. 1202–1206, 2012.

[4] J. Li, T. Li, and J. Ren, "Beyond the MDS bound in distributed cloud storage," in *IEEE INFOCOM 2014*, (Toronto, CA.), April 27-May 2 2014.

[5] J. Li, T. Li, and J. Ren, "Beyond the MDS bound in distributed storage," *IEEE Transactions on Information Theory*, vol. 66, no. 7, pp. 3957–3975, 2020.

[6] J. Li, T. Li, and J. Ren, "Optimal construction of regenerating code through rate-matching in hostile networks," *IEEE Transactions on Information Theory*, vol. 63, pp. 4414–4429, July 2017.

[7] Y. Wu, "A construction of systematic MDS codes with minimum repair bandwidth," *IEEE Transactions on Information Theory*, vol. 57, no. 6, pp. 3738–3741, 2011.

[8] D. Papailiopoulos, J. Luo, A. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *INFOCOM, 2012 Proceedings IEEE*, pp. 2801–2805, 2012.

[9] D. Papailiopoulos, A. Dimakis, and V. Cadambe, "Repair optimal erasure codes through hadamard designs," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3021–3037, 2013.

[10] A. Wang and Z. Zhang, "Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage," in *INFOCOM, 2013 Proceedings IEEE*, pp. 400–404, 2013.

[11] C. Tian and T. Liu, "Multilevel diversity coding with regeneration," *IEEE Transactions on Information Theory*, vol. 62, no. 9, pp. 4833–4847, 2016.

[12] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *45th Annu. Allerton Conf. Control, Computing, and Communication*, 2007.

[13] A. Duminuco and E. Biersack, "A practical study of regenerating codes for peer-to-peer backup systems," in *ICDCS '09. 29th IEEE International Conference on Distributed Computing Systems, 2009*, pp. 376 – 384, June 2009.

[14] K. Shum, "Cooperative regenerating codes for distributed storage systems," in *2011 IEEE International Conference on Communications (ICC)*, pp. 1–5, 2011.

[15] H. Hou, K. W. Shum, M. Chen, and H. Li, "Basic codes: Low-complexity regenerating codes for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3053–3069, 2016.

[16] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *IEEE International Symposium on Information Theory, 2009. ISIT 2009.*, pp. 2276–2280, 2009.

[17] N. Shah, K. Rashmi, P. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, pp. 2134 – 2158, 2012.

[18] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, pp. 5227–5239, 2011.

[19] V. Guruswami and M. Wootters, "Repairing reed-solomon codes," *http://arxiv.org/abs/1509.04764*, 2016.

[20] C. Tian, B. Sasidharan, V. Aggarwal, V. A. Vaishampayan, and P. V. Kumar, "Layered exact-repair regenerating codes via embedded error correction and block designs," *IEEE Transactions on Information Theory*, vol. 61, no. 4, pp. 1933–1947, 2015.

[21] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie, "Lazy verification in fault-tolerant distributed storage systems," in *SRDS 2005. 24th IEEE Symposium on Reliable Distributed Systems, 2005*, pp. 179–190, 2005.

[22] C. Cachin and S. Tessaro, "Optimal resilience for erasure-coded byzantine distributed storage," in *DSN 2006. International Conference on Dependable Systems and Networks, 2006*, pp. 115–124, 2006.

[23] Y. Han, R. Zheng, and W. H. Mow, "Exact regenerating codes for byzantine fault tolerance in distributed storage," in *Proceedings IEEE INFOCOM*, pp. 2498 – 2506, 2012.

[24] H. Chen and P. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage," in *2012 IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, pp. 51–60, 2012.

[25] F. Oggier and A. Datta, "Byzantine fault tolerance of regenerating codes," in *2011 IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 112–121, 2011.

[26] S. Pawar, S. El Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," *IEEE Transactions on Information Theory*, vol. 57, pp. 6734 – 6753, 2011.

[27] N. B. Shah, K. V. Rashmi, K. Ramchandran, and P. V. Kumar, "Privacy-preserving and secure distributed storage codes," *http://www.eecs.berkeley.edu/~nihar/publications/privacy_security.pdf/*.

[28] R. Tandon, S. Amuru, T. C. Clancy, and R. M. Buehrer, "Toward optimal secure distributed storage systems with exact repair," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3477–3492, 2016.

[29] K. Huang, U. Parampalli, and M. Xian, "On secrecy capacity of minimum storage regenerating codes," *IEEE Transactions on Information Theory*, vol. 63, pp. 1510–1524, March 2017.

[30] A. S. Rawat, "Secrecy capacity of minimum storage regenerating codes," in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 1406–1410, June 2017.

[31] S. Shao, T. Liu, C. Tian, and C. Shen, "On the tradeoff region of secure exact-repair regenerating codes," *IEEE Transactions on Information Theory*, vol. 63, pp. 7253–7266, Nov 2017.

[32] M. Xu, J. Zhang, H. Guo, X. Cheng, D. Yu, Q. Hu, Y. Li, and Y. Wu, "FileDES: A secure, scalable and succinct decentralized encrypted storage network," *Cryptology ePrint Archive*, 2024.

[33] H. Guo, M. Xu, J. Zhang, C. Liu, D. He, and X. Cheng, "FileDAG: A multi-version decentralized storage network built on dag-based blockchain," *IEEE Transactions on Computers*, vol. 72, no. 11, p. 3191–3202, 2023.

[34] H. Guo, M. Xu, J. Zhang, C. Liu, R. Ranjan, D. Yu, and X. Cheng, "BFT-DSN: A byzantine fault tolerant decentralized storage network," *IEEE Transactions on Computers, Early Access*.

[35] J. Li, T. Li, and J. Ren, "Secure regenerating code," in *Proc. IEEE Global Telecommunications Conference (Globecom)*, (Austin, TX.), 8-12 December 2014.

[36] J. Li, T. Li, and J. Ren, "Rate-matched regenerating code in hostile networks," in *Proc. IEEE International Conference on Communications (ICC)*, (London, UK), 2015.

[37] J. Ren, "On the structure of hermitian codes and decoding for burst errors," *IEEE Transactions on Information Theory*, vol. 50, pp. 2850–2854, 2004.

[38] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*. Prentice Hall.

Authorized licensed use limited to: Michigan State University. Downloaded on November 10,2025 at 21:42:12 UTC from IEEE Xplore. Restrictions apply.